UNIVERSITY OF CALIFORNIA, SAN DIEGO

Automatic music tagging with time series models

A dissertation submitted in partial satisfaction of the
requirements for the degree of Doctor of Philosophy

in

Electrical Engineering (Intelligent Systems, Robotics and Control)

by

Emanuele Coviello

Committee in charge:

Professor Gert Lanckriet, Chair
Professor Antoni B. Chan
Professor Sanjoy Dasgupta
Professor Bhaskar D. Rao
Professor Lawrence Saul
Professor Nuno Vasconcelos

2014

The Dissertation of Emanuele Coviello is approved and is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2014

DEDICATION

To Liz, because only for you did I come back to San Diego, and manage to do all of this.

And to nonna Milvia. If there existed such a thing as a doctorate in being a extraordinary grandmother, you'd for sure have one.[1]

[1] E a nonna Milvia. Se ci fosse un dottorato in nonna, tu di sicuro ne avresti uno.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

I thank everyone that in one way or another helped me getting to this point. In particular:

Gert and Antoni for their invaluable support and mentorship. Gert for being welcoming when I popped into his office without notification that first time, and for making the impossible possible. Antoni, for always being there, no matter of the over 7000 miles that separate our offices, and for showing me how it is done.

Adeel Mumtaz, for being an enthusiastic collaborator.

The guys at the Computer Audition Laboratory. Yonatan for his witty jokes, and Daryl for always being up for a hour-long discussion. The girls, Kat and Janani, and the forefathers: Luke, Douglas and Brian for paving the road for all of us.

The students of the "text and image modeling" team, Nikhil, Jose and Gabriel, and professor Roger Levy.

Riccardo Miotto, for showing me that one can achieve way above his initial expectations.

Bhaskar Rao, for getting me started on research.

Nuno Vasconcelos, for his lessons, the first version of the hierarchical EM, and for preparing Antoni.

Sanjoy Dasgupta, and Lawrence Saul, for their teachings and for being always available.

Francesco Rossetto, for his superlative guide though my first steps in research.

Michele Zorzi, for being always supportive, especially after unplanned changes of route.

The professors of the University of Padova for getting me in the right shape.

Malcolm Slaney, for the inspiring advices on research and the insightful comments on my papers.

Ben Rubinstein, Ariel Fuxman and Patrick Pantel, for their mentorship at MSR.

Steve Checkoway for making the template I used for writing this dissertation, and saving me hours of work.

The several colleagues I met at conferences, in particular, Philippe Hamel, Jeff Scott and Erik Schmidt, Gautham Mysore, Eric Battenberg, Eric Humphrey, Nicola Montecchio, Nicola Orio, Matt Hoffman, and many others, for the good times together and for increasing my motivation to publish.

Everyone that stepped over the 4th floor of Calit-2 in the past five year or so, for creating a fun working environment, and most of all for being patient about my (and Lorenzo's) loud Italian-speaking. In particular, Giorgio Quer, for letting me "borrow" his coffee, Alon Orlitsky, for his great humor, and Hirakendu Das for his technical support.

And on top of all, my brother Lorenzo, for always setting the bar high and at the same time reminding me about the real priorities in life—food and swimming.

Gioele, for being a patient listener.

My parents and grandparents, for the genes they passed, the values they taught, and the opportunities the gave me.

Liz, for putting up with and marrying me during my PhD.

And Peanut, for always being next to me, dozing while I am working, from early in the morning till late at night.

Chapter 2, in full, is a reprint of the material as it appears in the IEEE Transactions on Audio, Speech and Language Processing, 19(5):1343-1359, July 2011, E. Coviello, A. Chan, and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 3, in full, is a reprint of the material as it appears in the Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012, Porto (Portugal), 8-12 October 2012, E. Coviello, Y. Vaizman, A.B. Chan and

G. Lanckriet. The dissertation author was co-primary investigator and co-author of this paper.

Chapter 4, in full, is a reprint of the material as it appears in the Journal of Machine Learning Research, 15, pp. 697-747, 2014, E. Coviello, A.B. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 5, in full, is a reprint of the material as it appears in the Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami (USA). 24-28 October 2011, E. Coviello, R. Miotto, and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 6, in full, is a reprint of the material as it appears in the IEEE Transactions on Audio, Speech and Language Processing, 21(9):2554-2569, December 2013, K. Ellis, E. Coviello, A. Chan and G. Lanckriet. The dissertation author was a co-author of this paper.

Chapter 7, in full, is a reprint of the material as it appears in the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2012, pp. 1979-1986, Providence (USA), 18-20 June 2012, E. Coviello, A. Mumtaz, A. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 8, in full, is a reprint of the material as it appears in the Proceedings of The 30th International Conference on Machine Learning, JMLR W&CP 28 (3) :468-476, Atlanta, Georgia (USA), 16-21 June 2013, E. Coviello, A. Mumtaz, A.B. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

VITA

2006	Bachelor of Science in Information Engineering, Università degli Studi di Padova
2008	Master of Science in Telecommunication Engineering, Università degli Studi di Padova
2014	Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control), University of California, San Diego

PUBLICATIONS

E. Coviello, A.B. Chan and G. Lanckriet. "Clustering Hidden Markov Models with Variational HEM." *Journal of Machine Learning Research*, 15, pp. 697-747, 2014.

J. Costa Pereira, E. Coviello, G. Doyle, N. Rasiwasia, G.R.G. Lanckriet, R. Levy, N. Vasconcelos. "On the Role of Correlation and Abstraction in Cross-Modal Multimedia Retrieval." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):521-535, March 2014.

K. Ellis, E. Coviello, A. Chan and G. Lanckriet. "A Bag of Systems Representation for Music Auto-tagging." *IEEE Transactions on Audio, Speech and Language Processing*, 21(9):2554-2569, December 2013.

T. Chuk, A.C.W. Ng, E. Coviello, A.B. Chan and J.H. Hsiao. "Understanding Eye Movements in Face Recognition with Hidden Markov Models." In *Proceedings of the 35th Annual Conference of the Cognitive Science Society*, CogSci 2013, Berlin (Germany), 31 July-3 August 2013

A. Mumtaz, E. Coviello, A. Chan and G. Lanckriet. "Clustering Dynamic Textures with the Hierarchical EM Algorithm for Modeling Video." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(7):1606-1621, July 2013.

E. Coviello, A. Mumtaz, A.B. Chan and G. Lanckriet. "That was Fast! Speeding Up NN Search of High Dimensional Distributions." In *Proceedings of The 30th International Conference on Machine Learning*, JMLR W&CP 28 (3) :468-476, Atlanta, Georgia (USA), 16-21 June 2013.

E. Coviello, A.B. Chan and G. Lanckriet. "The Variational Hierarchical EM Algorithm for Clustering Hidden Markov Models." In *Advances in Neural Information Processing*

Systems 25, NIPS 2012, pp. 413-421, Lake Tahoe, Nevada (USA), 3-8 December 2012.

E. Coviello, Y. Vaizman, A.B. Chan and G. Lanckriet. "Multivariate Autoregressive Mixture Models for Music." In *Proceedings of the 13th International Society for Music Information Retrieval Conference*, ISMIR 2012, Porto (Portugal), 8-12 October 2012.

E. Coviello, A. Mumtaz, A. Chan and G. Lanckriet. "Growing a Bag of Systems Tree for Fast and Accurate Classification." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2012, pp. 1979-1986, Providence (USA), 18-20 June 2012.

K. Ellis, E. Coviello, and G. Lanckriet. "Semantic Annotation and Retrieval of Music Using a Bag of Systems Representation." In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR 2011, Miami (USA). 24-28 October 2011.

E. Coviello, R. Miotto, and G. Lanckriet. "Combining Content-Based Auto-Taggers with Decision-Fusion." In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR 2011, Miami (USA). 24-28 October 2011.

E. Coviello, A. Chan, and G. Lanckriet. "Time Series Models for Semantic Music Annotation." *IEEE Transactions on Audio, Speech and Language Processing*, 19(5):1343-1359, July 2011.

N. Rasiwasia, J.C. Pereira, E. Coviello, G. Doyle, G.R.G. Lanckriet, R. Levy, N. Vasconcelos. "A New Approach to Cross-Modal Multimedia Retrieval." In *Proceedings of the international conference on Multimedia*, pp. 251-260. ACM, Firenze (Italy), 25-29 October 2010.

E. Coviello, L. Barrington, A. Chan, and G. Lanckriet. "Automatic Music Tagging with Time Series Models." In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, ISMIR 2010, Utrecht (Netherlands). 9-13 August 2010.

A. Chan, E. Coviello, and G. Lanckriet. "Clustering Dynamic Textures with the Hierarchical EM Algorithm." In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, CVPR 2010, pp. 1979-1986, San Francisco (USA), 13-18 June 2010.

A. B. Chan, E. Coviello, and G. Lanckriet. "Derivation of the Hierarchical EM Algorithm for Dynamic Textures." City University of Hong Kong, Tech. Rep., 2010.

Emanuele Coviello, Abhijeet Bhorkar, Francesco Rossetto, Bhaskar D. Rao and Michele Zorzi. "A Robust Approach to Carrier Sense for MIMO Ad Hoc Networks." In *Proceedings of the IEEE International Conference on Communications* ICC 2009, Dresden (Germany), 15-17 June 2009.

ABSTRACT OF THE DISSERTATION

Automatic music tagging with time series models

by

Emanuele Coviello

Doctor of Philosophy in Electrical Engineering (Intelligent Systems, Robotics and Control)

University of California, San Diego, 2014

Professor Gert Lanckriet, Chair

As music distribution has evolved form physical media to digital content, tens of millions of songs are instantly available to consumers through several online services. In order to help users search, browse and discover songs from these extensive collections, music information retrieval systems have been developed to assist in automatically analyzing, indexing and recommending musical content.

This dissertation proposes machine learning methods for content-based automatic tagging of music, and evaluates their performance on music annotation and retrieval tasks. The proposed methods rely on time-series models of the musical signal, to account for

longer term temporal dynamics of music in addition to timbral textures, and allow to leverage different types of models and information at multiple time scales in a single system. Efficient algorithms for estimation and deployment are proposed for all the considered methods.

Chapter 1

Introduction

In the last decade, continuous technological advancements have revolutionized music production, distribution and sharing. Today, online services like iTunes,[1] Spotify[2] or YouTube[3] bring in millions of visitors by offering instant access to tens of million of songs and music videos. The size of these collections, constantly enriched by new clips (from established artists as well as less known performers), can be quite disorienting for the user, and can inhibit finding desired content or discovering new music. For example, it is not straightforward to find a "live jazz performance with saxophone on a groovy bass line" unless having already in mind a specific song title or artist name, or simply decide what to listen next. This have determined a growing interest in music information retrieval (MIR), which aims at building technology to help users search, browse and discover music, and, ultimately, to help content providers monetize their catalogues more effectively.

Commercial MIR systems for search and recommendation can be broadly categorized into two groups, i) systems based on collaborative filtering and ii) systems based on meta-data and semantic annotations. Collaborative filtering relies on users' listening or purchase histories, to make guesses on users's tastes and provide recommendations.

[1] http://www.apple.com/itunes
[2] http://www.spotify.com
[3] http://www.youtube.com

For examples, Last.fm[4] and iTunes are known to use collaborative filtering for their recommendations [16].

Services based on semantic annotations rely on tags—short textual bits that describe specific characteristics of music pieces, ranging from genre and instrumentation, to mood and usage. Tags bridge the gap between music and human semantics, and enable music search and indexing based on transparent textual descriptions as in standard text search engine (e.g., by matching the tags in a query to the tags associated to the songs in a catalogue), or query-by-example recommendation based on semantic similarity (as opposed to acoustic similarity) to a query song. Historically, these systems have initially relied on available metadata (e.g. artist biographies, genre annotation, blogs or critical reviews), or manual labeling from expert musicologist. While solutions based on expect annotators provide accurate annotations, they are expensive and hard to scale. For example, the catalogue of the Music Genome Project[5] is only a fraction of iTunes catalogue. More recently, crowd sourcing solutions, such as TagATune [98] and HerdIt [14], have attempted to scale up manual labeling by inviting non-expert users to engaging or rewarding games. However, these efforts have not become viral, and by themselves can cover only a small portion of the songs available in modern music collections.[6]

In practice, meta-data based solutions and collaborative filters work well on popular music. Indeed, for popular artists and popular songs, a wealth of information is easily accessible over the internet, and can be accurately indexed through standard text search technologies, making meta-data based technique successful. Similarly, popular content is consumed by a large number of users, and is hence well represented in a collaborative filter.

Unfortunately, neither of these methodologies handles well the “long tail” of less

[4] http://www.last.fm

[5] http://www.pandora.com/about/mgp

[6] TagATune labeled clips represent less than 0.15% of the iTunes’ collection.

popular or novel content. For example, relevant meta-data for less popular content is generally scarce and hard to retrieve, and in many case may be spammy or even absent (e.g., the only available piece of information being an audio file uploaded on a streaming service). Similarly, unpopular content is poorly represented in a collaborative filter as the corresponding historical data is typically limited. As a consequence of this *cold start* problem, the long-tail of less popular content will not be recommended even when it would be rated favorably [59].

This motivates the development of content-based auto-tagging systems [159, 117, 81, 106, 55], i.e., intelligent algorithms that, by analyzing and understanding the acoustic content of songs, can automatically index them with semantic tags, which can then be used to improve the search experience and speed up the discovery of desired content. Several auto-tagging algorithms follow a common recipe based on signal processing and machine learning, and are trained on a corpus of songs annotated with respect to the tags. First, each song is is represented as a sequence a of audio features. Successively, a series of statistical models, one for each tag, is estimated to lean the most predictive patterns for the tags. The tag models are then used to analyze new songs. In particular, the audio features of a new song are compared against each tag model, and the tags whose models fit beast are selected to annotate the song.

Traditionally, auto-taggers uses a bag of feature (BoF) representation of songs where audio features are treated as independent, which ignores the temporal order or dynamics between them. This type of representation fails to account for the longer-term musical dynamics (e.g., tempo and beat) or temporal structures (e.g., riffs and arpeggios), which are clearly important characteristics of a musical signal.

This dissertation investigates the use of time-series models for automatic tagging of music. By gluing together consecutive features into sequences, time-series models capture both instantaneous timbal information as well as temporal dynamics of music.

1.1 Summary of contributions

This dissertation is divided into two parts. The first part (chapters 2 to 4) investigates the use of a variety of time series models to model tags directly. The second part (chapters 5 to 8) proposes a high-level music descriptor that can leverage the benefits of different-models and different-time scales into a single auto-tagger.

Chapter 2 proposes the use of dynamic texture mixture (DTM) models for music auto-tagging, and shows improvements especially on tags with temporal dynamics that unfold in the time span of a few seconds. In order to allow efficient estimation, chapter 2 also discusses an efficient hierarchical expectation maximization (HEM) algorithm for estimating hidden state models.

Chapter 3 investigates the uses of auto regressive mixture models for music auto-tagging, and studies the tradeoff between performance and efficiency for a variety of time-series models of different complexities.

Chapter 4 presents a music auto-tagger based on hidden Markov models (HMMs) mixtures. In particular, chapter 4 proposes the variational HEM algorithm for clustering HMMs, and compares its performance against existing techniques on a wide range of applications, also outside the music arena.

Chapter 5 introduces *decision fusion*, a framework that combines the benefits of different content-based auto-tagger, by modeling the co-occurrence of their tag predictions.

Chapter 6 proposes the bag of systems (BoS) representation of music, a high-level descriptor of musical content that uses generative models as musical codewords. This type of representation allows to combine different types of generative models and different time scales, and can leverage a large corpus of (unlabeled) music to build a richer descriptor.

Chapter 7 introduces the BoS Tree, an efficient data structure to index a large BoS codebook efficiently, based on a bottom-up hierarchy of codewords. Chapter 7 demonstrates the advantages of the BoS Tree on modeling videos as well.

Finally, chapter 8 proposes a branch and bound search, which is used to navigate the BoS Tree effectively. More generally, chapter 8 shows that the proposed branch and bound method allows for fast and accurate approximate nearest neighbor search on a variety of problems involving high dimensional data.

Chapter 2

Time Series Models for Semantic Music Annotation

2.1 Introduction

Recent technologies fueled new trends in music production, distribution and sharing. As a consequence, an already large corpus of millions of musical pieces is constantly enriched with new songs (by established artists as well as less known performers), all of which are instantly available to millions of consumers through online distribution channels, personal listening devices, etc. This age of music proliferation created a strong need for music search and discovery engines, to help users find 'Mellow Beatles songs' on a nostalgic night, or satisfy their sudden desire for 'psychedelic rock with distorted guitar and deep male vocals,' without knowing appropriate artists or song titles. A key scientific challenge in creating this search technology is the development of intelligent algorithms, trained to map the human perception of music within the coded confine of computers, to assist in automatically analyzing, indexing and recommending from this extensive corpus of musical content [69].

This paper concerns *automatic tagging* of music with descriptive keywords (e.g., genres, emotions, instruments, usages, etc.), based on the content of the song. Music annotations can be used for a variety of purposes, such as searching for songs

exhibiting specific qualities (e.g., "jazz songs with female vocals and saxophone"), or retrieval of semantically similar songs (e.g., generating play-lists based on songs with similar annotations). Since semantics is a compact, popular medium to describe an auditory experience, it is essential that a music search and discovery system supports these semantics-based retrieval mechanisms, to recommend content from a large audio database.

State-of-the-art music "auto-taggers" represent a song as a "bag of audio features" (e.g., [159, 117, 81, 106, 55]). The bag-of-features representation extracts audio features from the song at regular time intervals, but then treats these features independently, ignoring the temporal order or dynamics between them. Hence, this representation fails to account for the longer-term musical dynamics (e.g., tempo and beat) or temporal structures (e.g., riffs and arpeggios), which are clearly important characteristics of a musical signal.

To address this limitation, one approach is to encode some temporal information in the features ([29, 107, 159, 81, 106, 55]) and keep using existing, time-independent models. For example, some of the previous approaches augment the "bag of audio features" representation with the audio features' first and second derivatives. While this can slightly enrich the representation at a short-time scale, it is clear that a more principled approach is required to model dynamics at a longer-term scale (seconds instead of milliseconds).

Therefore, in this paper, we explore the dynamic texture (DT) model [54], a generative *time series model* that captures longer-term time dependencies, for automatic tagging of musical content. The DT model represents a time series of audio features as a sample from a *linear dynamical system* (LDS), which is similar to the hidden Markov model (HMM) that has proven robust in music identification [140]. The difference is that HMMs quantize the audio signal into a fixed number of discrete "phonemes", while the

DT has a continuous state space, offering a more flexible model for music.

Since musical time series often show significant structural changes within a single song and have dynamics that are only locally homogeneous, a single DT would be insufficiently rich to model individual songs and, therefore, the typical musical content associated with semantic tags. To address this at the song-level, Barrington et al. [13] propose to model the audio fragments from a *single song* as samples from a dynamic texture mixture (DTM) model [32], for the task of automatic music *segmentation*. Their results demonstrated that the DTM provides an accurate segmentation of music into homogeneous, perceptually similar segments (corresponding to what a human listener would label as 'chorus', 'verse', 'bridge', etc.) by capturing *temporal* as well as *textural* aspects of the musical signal.

In this paper, we adopt the DTM model to propose a novel approach to the task of automatic music *annotation* that accounts for both the timbral content and the temporal dynamics that are predictive of a semantic *tag*. We first model each song in a music database as a DTM, capturing longer-term time dependencies and instantaneous spectral content at the *song-level*. Second, the characteristic temporal and timbral aspects of musical content commonly associated with a semantic tag are identified by learning a *tag-level* DTM that summarizes the common features of a (potentially large) set of song-level DTMs for the tag (as opposed to the tag-level Gaussian mixture models by Turnbull et al. [159], which do not capture temporal dynamics). Given all song-level DTMs associated with a particular tag, the common information is summarized by clustering similar song-level DTs using a novel, efficient hierarchical EM (HEM-DTM) algorithm. This gives rise to a tag-level DTM with few mixture components.

Experimental results show that the proposed time series model improves annotation and retrieval, in particular for tags with temporal dynamics that unfold in the time span of a few seconds.

In summary, this paper brings together a DTM model for music, a generative framework for music annotation and retrieval, and an efficient HEM-DTM algorithm. We will focus our discussion on the latter two. For the former, we provide an introduction and refer to our earlier work [13] for more details. The remainder of this paper is organized as follows. After an overview of related work on auto-tagging of music in Section 2.2, we introduce the DTM model in Section 2.3. Next, in Sections 2.4 and 2.5, we present an annotation and retrieval system for time series data, based on an efficient hierarchical EM algorithm for dynamic texture mixtures (HEM-DTM). In Sections 2.6 and 2.7, we present experiments using HEM-DTM for music annotation and retrieval. Finally, Section 2.8 illustrates qualitatively how variations in the acoustic characteristics of semantic tags affect the parameters of the corresponding DTM models.

2.2 Related Work

The prohibitive cost of manual labeling makes automated semantic understanding of audio content a core challenge in designing fully functional retrieval systems ([57, 117, 81, 107, 168, 27, 148, 159, 106, 55, 15, 126, 60, 150, 29, 160, 167]). To automatically annotate music with *semantic tags*, based on audio content, various discriminative machine learning algorithms have been proposed (e.g., multiple-instance [106], multiple-kernel [15], and stacked [117] SVMs, boosting [55], nearest-neighbor ([126, 60]), embedding methods [150], locally-sensitive hashing [29] and regularized least-squares [167]). The discriminative framework, however, can suffer from poorly- or weakly-labeled training data (e.g., positive examples considered as negatives due to incomplete annotations).

To overcome this problem, unsupervised learning algorithms have been considered (e.g., K-means[21], vector quantization[140]), ignoring any labels and determining the classes automatically. The learned clusters, however, are not guaranteed to have any

connection with the underlying semantic tags of interest.

The labeling problem is compounded since often only a subset of the song's features actually manifests the tag the entire song is labeled with (e.g., a song labeled with "saxophone" may only have a handful of features describing content where a saxophone is playing). This suggests a *generative modeling* approach, which is better suited at handling weakly-labeled data and estimating concept distributions that naturally emerge around concept-relevant audio content, while down-weighting irrelevant outliers. More details on how generative models accommodate weakly-labeled data by taking a multiple instance learning approach is provided by Carneiro et al. [28]. Moreover, generative models provide class-conditional probabilities, which naturally allows us to rank tags probabilistically for a song. Generative models have been applied to various music information retrieval problems. This includes Gaussian mixture models (GMMs) ([10, 160, 159]), hidden Markov models (HMMs) [140], hierarchical Dirichlet processes (HDPs) [80], and a codeword Bernoulli average model (CBA) [81]. Generative models used for automatic music annotation (e.g., GMMs and CBA) usually model the spectral content (and, sometimes, its first and second instantaneous derivatives) of short-time windows. These models ignore longer-term temporal dynamics of the musical signal. In this paper, we adopt dynamic texture mixture models for automatic music annotation. These generative time-series models capture both instantaneous spectral content, as well as longer-term temporal dynamics. Compared to HMMs, they have a continuous rather than discrete state space. Therefore, they do not require to quantize the rich sound of a musical signal into discrete "phonemes", making them an attractive model for music.

2.3 Dynamic Texture Mixture Model

In this section, we review the dynamic texture (DT) and dynamic texture mixture (DTM) models for modeling short audio fragments and whole songs.

2.3.1 The Dynamic Texture Model

A dynamic texture [54] (DT) is a generative model that takes into account both the instantaneous acoustics and the temporal dynamics of audio sequences (or audio fragments) [13]. The model consists of two random variables: y_t, which encodes the acoustic component (audio feature vector) at time t, and x_t, a hidden state variable which encodes the dynamics (evolution) of the acoustic component over time. The two variables are modeled as a *linear dynamical system*,

$$\begin{aligned} x_t &= Ax_{t-1} + v_t, \\ y_t &= Cx_t + w_t + \bar{y}, \end{aligned}$$

where $x_t \in \mathbb{R}^n$ and $y_t \in \mathbb{R}^m$ are real vectors (typically $n \ll m$). Using such a model, we assume that the dynamics of the audio can be summarized by a more parsimonious *hidden state process* x_t ($n < m$), which evolves as a first order Gauss-Markov process, and each *observation variable* y_t is dependent only on the current hidden state x_t.

The *state transition matrix* $A \in \mathbb{R}^{n \times n}$ encodes the dynamics or evolution of the hidden state variable (e.g., the evolution of the audio track), and the *observation matrix* $C \in \mathbb{R}^{m \times n}$ encodes the basis functions for representing the audio fragment. The vector $\bar{y} \in \mathbb{R}^m$ is the mean of the dynamic texture (i.e., the mean audio feature vector). The *driving noise process* v_t is zero-mean Gaussian distributed with covariance Q, i.e., $v_t \sim \mathcal{N}(0, Q)$, with $Q \in \mathbb{S}_+^n$, the set of symmetric, positive definite matrices of dimension $n \times n$. w_t is the *observation noise* and is also zero-mean Gaussian, with covariance R, i.e., $w_t \sim \mathcal{N}(0, R)$, with $R \in \mathbb{S}_+^m$. Finally, the *initial condition* is distributed as $x_1 \sim \mathcal{N}(\mu, S)$, where $\mu \in \mathbb{R}^n$ is the mean of the initial state, and $S \in \mathbb{S}_+^n$ the covariance. The DT is specified by the parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$.

Intuitively, the columns of C can be interpreted as the principal components (or

basis functions) of the audio feature vectors over time. Hence, each audio feature vector y_t can be represented as a linear combination of principal components, with corresponding weights given by the current hidden state x_t. In this way, the DT can be interpreted as a time-varying PCA representation of an audio feature vector time series. Figure 2.1(a) shows the graphical model of the DT, as it represents a short audio fragment.

2.3.2 The Dynamic Texture Mixture Model

A song is a combination of heterogeneous audio fragments with significant structural variations, and hence cannot be represented with a single DT model. To address this lack of global homogeneity, Barrington et al. [13] proposed to represent audio fragments, extracted from a song, as samples from a dynamic texture mixture (DTM) [32], effectively modeling the heterogeneous structure of the song. The DTM model [32] introduces an assignment random variable $z \sim \text{multinomial}(\pi_1, \cdots, \pi_K)$, which selects one of K dynamic texture components as the source of an audio fragment. Each mixture component is parameterized by

$$\Theta_z = \{A_z, C_z, Q_z, R_z, \mu_z, S_z, \bar{y}_z\}, \tag{2.1}$$

and the DTM model is parameterized by $\Theta = \{\pi_z, \Theta_z\}_{z=1}^{K}$.

Given a set of audio fragments extracted from a song, the maximum-likelihood parameters of the DTM can be estimated with recourse to the expectation-maximization (EM) algorithm, which is an iterative optimization method that alternates between estimating the hidden variables with the current parameters, and computing new parameters given the estimated hidden variables (the "complete data"). The EM algorithm for DTM alternates between estimating second-order statistics of the hidden states, conditioned on each audio fragment, with the Kalman smoothing filter (E-step), and computing

new parameters given these statistics (M-step). More details are provided by Chan and Vasconcelos [32].

Figure 2.1(b) illustrates the DTM representation of a song, where each DT component models homogeneous parts of the song. Previous work by Barrington et al. [13] has successfully used the DTM for the task of segmenting the structure of a song into acoustically similar sections (e.g., intro, verse, chorus, bridge, solo, outro). In this work, we propose that the DTM can also be used as a *tag-level annotation model* for music annotation and retrieval.

2.4 Music Annotation and Retrieval with DTMs

In this section we formulate the related tasks of annotation and retrieval of audio data as a supervised multi-class labeling (SML) problem [28] in the context of time series DTM models.

2.4.1 Notation

A song $\mathscr{Y}$ is represented as a collection of T overlapping time series, i.e., $\mathscr{Y} = \{y^1_{1:\tau}, \ldots, y^T_{1:\tau}\}$, where each $y^t_{1:\tau}$, called an audio *fragment*, represents τ sequential audio feature vectors extracted by passing a short-time window over the audio signal. The number of audio fragments, T, depends on the length of the song. The semantic content of a song with respect to a vocabulary $\mathscr{V}$ of size $|\mathscr{V}|$ is represented in an annotation vector $\mathbf{c} = [c_1, \ldots, c_{|\mathscr{V}|}]$, where $c_k > 0$ only if there is a positive association between the song and the tag w_k, otherwise $c_k = 0$. Each *semantic weight*, c_k, represents the degree of association between the song and the tag w_k. The data set $\mathscr{D}$ is a collection of $|\mathscr{D}|$ song-annotation pairs $(\mathscr{Y}_d, \mathbf{c}_d)$.

2.4.2 Music Annotation

We treat annotation as a supervised multi-class problem [28, 159] in which each class is a tag w, from a vocabulary $\mathscr{V}$ of unique tags (e.g., "bass guitar", "hip hop", "boring"). Each tag w_k is modeled with a probability distribution over the space of audio fragments, i.e., $p(y^t_{1:\tau}|w_k)$ for $k = 1, \ldots, |\mathscr{V}|$, which is a DTM. The annotation task is to find the subset $\mathscr{W} = \{w_1, \ldots, w_{\mathscr{A}}\} \subseteq \mathscr{V}$ of $\mathscr{A}$ tags that best describe a novel song $\mathscr{Y}$.

Given the audio fragments of a novel song $\mathscr{Y}$, the most relevant tags are the ones with highest posterior probability, computed using Bayes' rule:

$$p(w_k|\mathscr{Y}) = \frac{p(\mathscr{Y}|w_k)\, p(w_k)}{p(\mathscr{Y})}, \tag{2.2}$$

where $p(w_k)$ is the prior of the k^{th} tag and $p(\mathscr{Y})$ the song prior. To promote annotation using a diverse set of tags, we assume a uniform prior, i.e., $p(w_k) = 1/|\mathscr{V}|$ for $k = 1, \ldots, |\mathscr{V}|$. To estimate the likelihood term in (5.2), $p(\mathscr{Y}|w_k)$, we assume that song fragments, $y^t_{1:\tau}$, are conditionally independent (given w_k). To compensate for the inaccuracy of this naïve Bayes assumption and keep the posterior from being too "peaked", one common solution is to estimate the likelihood term with the geometric average [159] (in this case, the geometric average of the individual audio fragment likelihoods):

$$p(\mathscr{Y}|w_k) = \prod_{t=1}^{T} \left(p(y^t_{1:\tau}|w_k)\right)^{\frac{1}{T\tau}}. \tag{2.3}$$

Note that, besides normalizing by T, we also normalize by the length of the audio fragment, τ, due to the high dimension of the probability distribution of the DTM time series model. The likelihood terms $p(y^t_{1:\tau}|w_k)$ of the DTM tag models can be computed efficiently with the "innovations" form of the likelihood using the Kalman filter [32], [147].

Unlike bag-of-features models that discard any dependency between audio feature vectors, (2.3) only assumes independence between different *sequences* of audio feature vectors (i.e., audio fragments, describing seconds of audio). Correlation *within* a single sequence is directly accounted for by the time series model.

The probability that the song $\mathscr{Y}$ can be described by the tag w_k is

$$p(w_k|\mathscr{Y}) = \frac{\prod_{t=1}^{T}\left(p(y_{1:\tau}^t|w_k)\right)^{\frac{1}{T\tau}}}{\sum_{l=1}^{|\mathscr{V}|}\prod_{t=1}^{T}\left(p(y_{1:\tau}^t|w_l)\right)^{\frac{1}{T\tau}}}\,, \tag{2.4}$$

where the song prior $p(\mathscr{Y}) = \sum_{l=1}^{|\mathscr{V}|} p(\mathscr{Y}|w_l)p(w_l)$. Finally, the song can be represented as a semantic multinomial, $\mathbf{p} = [p_1, \ldots, p_{|\mathscr{V}|}]$, where each $p_k = p(w_k|\mathscr{Y})$ represents the relevance of the k^{th} tag for the song, and $\sum_{i=1}^{|\mathscr{V}|} p_i = 1$. We annotate a song with the most likely tags according to $\mathbf{p}$, i.e., we select the tags with the largest probability.

2.4.3 Music Retrieval

Given a tag-based query, songs in the database can be retrieved based on their relevance to this semantic query[1]. In particular, we determine a song's relevance to a query with tag w_k based on the posterior probability of the tag, $p(w_k|\mathscr{Y})$, in (2.4). Hence, retrieval involves rank-ordering the songs in the database, based on the k^{th} entry (p_k) of the semantic multinomials $\mathbf{p}$.

Note that the songs could also be ranked by the likelihood of the song given the query, i.e., $p(\mathscr{Y}|w_k)$. However, this tends not to work well in practice because it favors generic songs that are most similar to the song prior $p(\mathscr{Y})$, resulting in the same retrieval result for any query w_k. Normalizing by the song prior $p(\mathscr{Y})$ fixes this problem, yielding the ranking based on semantic multinomials (assuming a uniform tag prior) described above.

[1]Note that although this work focuses on single-tag queries, our representation easily extends to multiple-tag queries [158].

2.5 Learning DTM Tag Models with the Hierarchical EM Algorithm

In this paper, we represent the tag models with dynamic texture mixture models. In other words, the tag distribution $p(y^t_{1:\tau}|w_k)$ is modeled with the probability density of the DTM, which is estimated from the set of training songs associated with the particular tag. One approach to estimation is to extract all the audio fragments from the relevant training songs, and then run the EM algorithm [32] directly on this data to learn the tag-level DTM. This approach, however, requires storing many audio fragments in memory (RAM) for running the EM algorithm. For even modest-sized databases, the memory requirements can exceed the RAM capacity of most computers.

To allow efficient training in both computation time and memory requirements, the learning procedure is split into two steps. First, a song-level DTM model is learned for each song in the training set using the standard EM algorithm [32]. Next, a tag-level model is formed by pooling together all the song-level DTMs associated with a tag, to form a large mixture. However, a drawback of this model aggregation approach is that the number of DTs in the DTM tag model grows linearly with the size of the training data, making inference computationally inefficient when using large training sets. To alleviate this problem, the DTM tag models formed by model aggregation are reduced to a representative DTM with fewer components by using the hierarchical EM (HEM) algorithm presented in this section. The HEM algorithm clusters together similar DTs in the song-level DTMs, thus summarizing the common information in songs associated with a particular tag. The new DTM tag model allows for more efficient inference, due to fewer mixture components, while maintaining a reliable representation of the tag-level model.

Because the database is first processed at the song level, the computation can be

easily done in parallel (over the songs) and the memory requirement is greatly reduced to that of processing a single song. The memory requirement for computing the tag-level models is also reduced, since each song is succinctly modeled by the parameters of a DTM. Such a reduction in computational complexity also ensures that the tag-level models can be learned from cheaper, weakly-labeled data (i.e., missing labels, labels without segmentation data) by pooling over large amounts of audio data to amplify the appropriate attributes.

In summary, adopting DTM, or time series models in general, as a tag model for SML annotation requires an appropriate HEM algorithm for efficiently learning the tag-level models from the song-level models. In the remainder of the section, we present the HEM algorithm for DTM.

2.5.1 Learning DTM Tag Models

The process for learning a tag-level DTM model from song-level DTMs is illustrated in Figure 2.2. First, all the song-level DTMs with a particular tag are pooled together into a single large DTM. Next, the common information is summarized by clustering similar DT components together, forming a new *tag-level* DTM with fewer mixture components.

The DT components are clustered using the hierarchical expectation maximization (HEM) algorithm [164]. At a high level, this is done by generating *virtual samples* from each of the song-level component models, merging all the samples, and then running the standard EM algorithm on the merged samples to form the reduced tag-level mixture. Mathematically, however, using the virtual samples is equivalent to marginalizing over the distribution of song-level models. Hence, the tag model can be learned directly and efficiently from the parameters of the song-level models, without generating any virtual samples.

The hierarchical expectation-maximization (HEM) algorithm was originally proposed by Vasconcelos and Lippman [164] to reduce a Gaussian mixture model (GMM) with a large number of mixture components into a representative GMM with fewer components, and has been successful in learning GMMs from large datasets for the annotation and retrieval of images [28] and music [159]. We next present an HEM algorithm for mixtures with components that are *dynamic textures* [33].

2.5.2 HEM Formulation

Formally, let $\Theta^{(s)} = \{\pi_i^{(s)}, \Theta_i^{(s)}\}_{i=1}^{K^{(s)}}$ denote the combined *song-level* DTM (i.e., after pooling all song-level DTMs for a certain tag) with $K^{(s)}$ components, where $\Theta_i^{(s)}$ are the parameters for the i^{th} DT component, and $\pi_i^{(s)}$ the corresponding component weights, which are normalized to sum to 1 (i.e., $\sum_i \pi_i^{(s)} = 1$). The likelihood of observing an audio fragment $y_{1:\tau}$ with length τ from the combined song-level DTM $\Theta^{(s)}$ is given by

$$p(y_{1:\tau}|\Theta^{(s)}) = \sum_{i=1}^{K^{(s)}} \pi_i^{(s)} p(y_{1:\tau}|z^{(s)} = i, \Theta^{(s)}), \tag{2.5}$$

where $z^{(s)} \sim \text{multinomial}(\pi_1^{(s)}, \cdots \pi_{K^{(s)}}^{(s)})$ is the hidden variable that indexes the mixture components. $p(y_{1:\tau}|z^{(s)} = i, \Theta^{(s)})$ is the likelihood of the audio fragment $y_{1:\tau}$ under the i^{th} DT mixture component.

The goal is to find a tag-level *annotation* DTM, $\Theta^{(a)} = \{\pi_j^{(a)}, \Theta_j^{(a)}\}_{j=1}^{K^{(a)}}$, which represents (7.6) using fewer number of mixture components, $K^{(a)}$, (i.e., $K^{(a)} < K^{(s)}$). The likelihood of observing an audio fragment $y_{1:\tau}$ from the tag-level DTM $\Theta^{(a)}$ is

$$p(y_{1:\tau}|\Theta^{(a)}) = \sum_{j=1}^{K^{(a)}} \pi_j^{(a)} p(y_{1:\tau}|z^{(a)} = j, \Theta^{(a)}), \tag{2.6}$$

where $z^{(a)} \sim \text{multinomial}(\pi_1^{(a)}, \cdots, \pi_{K^{(a)}}^{(a)})$ is the hidden variable for indexing components

in $\Theta^{(a)}$. Note that we will always use i and j to index the components of the song-level model, $\Theta^{(s)}$, and the tag-level model, $\Theta^{(a)}$, respectively. To reduce clutter, we will also use the short-hand $\Theta_i^{(s)}$ and $\Theta_j^{(a)}$ to denote the i^{th} component of $\Theta^{(s)}$ and the j^{th} component of $\Theta^{(a)}$, respectively. For example, we denote $p(y_{1:\tau}|z^{(s)}=i,\Theta^{(s)})$ as $p(y_{1:\tau}|\Theta_i^{(s)})$.

2.5.3 Parameter Estimation

To obtain the tag-level model, HEM [164] considers a set of N virtual observations drawn from the song-level model $\Theta^{(s)}$, such that $N_i = N\pi_i^{(s)}$ samples are drawn from the i^{th} component. We denote the set of N_i virtual audio samples for the i^{th} component as $Y_i = \{y_{1:\tau}^{(i,m)}\}_{m=1}^{N_i}$, where $y_{1:\tau}^{(i,m)} \sim \Theta_i^{(s)}$ is a single audio sample and τ is the length of the virtual audio samples (a parameter we can choose). The entire set of N samples is denoted as $Y = \{Y_i\}_{i=1}^{K^{(s)}}$. To obtain a consistent hierarchical clustering, we also assume that all the samples in a set Y_i are eventually assigned to the same tag-level component $\Theta_j^{(a)}$. We denote this as $z_i^{(a)} = j$. The parameters of the tag-level model can then be estimated by maximizing the likelihood of the virtual audio samples,

$$\Theta^{(a)*} = \arg\max_{\Theta^{(a)}} \log p(Y|\Theta^{(a)}), \tag{2.7}$$

where

$$\log p(Y|\Theta^{(a)}) = \log \prod_{i=1}^{K^{(s)}} p(Y_i|\Theta^{(a)}) \tag{2.8}$$

$$= \log \prod_{i=1}^{K^{(s)}} \sum_{j=1}^{K^{(a)}} \pi_j^{(a)} \int p(Y_i, X_i|z_i^{(a)}=j,\Theta^{(a)})dX_i$$

$$= \log \prod_{i=1}^{K^{(s)}} \sum_{j=1}^{K^{(a)}} \pi_j^{(a)} \int p(Y_i, X_i|\Theta_j^{(a)})dX_i \tag{2.9}$$

and $X_i = \{x_{1:\tau}^{(i,m)}\}_{m=1}^{N_i}$ are the hidden state variables corresponding to Y_i. Computing the log-likelihood in (3.5) requires marginalizing over the hidden assignment variables $z_i^{(a)}$ and hidden state variables X_i. Hence, (2.7) can also be solved with recourse to the EM algorithm [49]. In particular, each iteration consists of

$$\begin{aligned}
&\text{E-Step: } \mathscr{Q}(\Theta^{(a)}, \hat{\Theta}^{(a)}) = \mathbb{E}_{X,Z|Y,\hat{\Theta}^{(a)}}[\log p(X,Y,Z|\Theta^{(a)})], \\
&\text{M-Step: } \hat{\Theta}^{(a)^*} = \arg\max_{\Theta^{(a)}} \mathscr{Q}(\Theta^{(a)}, \hat{\Theta}^{(a)}),
\end{aligned}$$

where $\hat{\Theta}^{(a)}$ is the current estimate of the tag-level model, $p(X,Y,Z|\Theta^{(a)})$ is the "complete-data" likelihood, and $\mathbb{E}_{X,Z|Y,\hat{\Theta}^{(a)}}$ is the conditional expectation with respect to the current model parameters.

As is common with the EM formulation, we introduce a hidden assignment variable $\mathbf{z}_{i,j}$, which is an indicator variable for when the audio sample set Y_i is assigned to the j^{th} component of $\Theta^{(a)}$, i.e., when $z_i^{(a)} = j$. The complete-data log-likelihood is then

$$\begin{aligned}
&\log p(X,Y,Z|\Theta^{(a)}) \\
&= \sum_{i=1}^{K^{(s)}} \sum_{j=1}^{K^{(a)}} \mathbf{z}_{i,j} \log \pi_j^{(a)} + \mathbf{z}_{i,j} \log p(Y_i, X_i|\Theta_j^{(a)}).
\end{aligned} \tag{2.10}$$

The $\mathscr{Q}$ function is then obtained by taking the conditional expectation of (3.6), and using the law of large numbers to remove the dependency on the virtual samples. The result is a $\mathscr{Q}$ function that depends only on the parameters of the song-level DTs $\Theta_i^{(s)}$. For the detailed derivation of HEM for DTM, we refer the reader to our earlier work [33, 34].

The HEM algorithm for DTM is summarized in Algorithm 1. In the E-step, the expectations in (3.13) are computed for each song-level component $\Theta_i^{(s)}$ and current tag-level component $\hat{\Theta}_j^{(a)}$. These expectations can be computed using "suboptimal filter analysis" or "sensitivity analysis" [65] on the Kalman smoothing filter (see Appendix

A and [33]). Next, the probability of assigning the song-level component $\Theta_i^{(s)}$ to the tag-level component $\Theta_j^{(a)}$ is computed according to (3.14), and the expectations are aggregated over all the song-level DTs in (3.16). In the M-step, the parameters for each tag-level component $\hat{\Theta}_j^{(a)}$ are recomputed according to the update equations in (2.14). Note that the E- and M-steps for HEM-DTM are related to the standard EM algorithm for DTM. In particular, the song-level DT components $\Theta_i^{(s)}$ take the role of the "data-points" in standard EM. This is manifested in the E-step of HEM as the expectation with respect to $\Theta_i^{(s)}$, which averages over the possible values of the "data-points". Given the aggregate expectations, the parameter updates in the M-step of HEM and EM are identical.

2.6 Music Datasets

In this section we describe the music collection and the audio features used in our experiments.

2.6.1 CAL500 Database

The CAL500 [159] dataset consists of 502 popular Western songs from the last 50 years from 502 different artists. Each song has been annotated by at least 3 humans, using a semantic vocabulary of 149 tags that describe genres, instruments, vocal characteristics, emotions, acoustic characteristics, and song usages. CAL500 provides hard binary annotations, which are 1 when a tag applies to the song and 0 when the tag does not apply. We find empirically that accurately fitting the HEM-DTM model requires a significant number of training examples, due to the large number of parameters in the model. Hence, we restrict our attention to the 78 tags with at least 50 positively associated songs.

2.6.2 Swat10k Database

Swat10k [156] is a collection of over ten thousand songs from 4,597 different artists, weakly labeled from a vocabulary of 18 genre tags, 135 sub-genre tags and 475 other acoustic tags. The song-tag associations are mined from Pandora's website. Each song is labeled with 2 to 25 tags. As for CAL500, we restrict our attention to the tags (125 genre tags and 326 acoustic tags) with at least 50 positively associated songs.

2.6.3 Audio Features

Mel-frequency cepstral coefficients (MFCCs) [103] are a popular feature for content-based music analysis, which concisely summarize the short-time content of an acoustic waveform by using the discrete cosine transform (DCT) to decorrelate the bins of a Mel-frequency spectrum[2]. In Section 2.3.1 we noted how 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 y_t. In this case, the columns of the observation matrix C (a learned PCA matrix) are analogous to the DCT basis functions, and the hidden states x_t are the coefficients (analogous to the MFCCs). The advantage of learning the PCA representation, rather than using the standard DCT basis, is that different basis functions (C matrices) can be learned to best represent the particular song or semantic tag of interest. Hence, the DT can focus on the frequency structure that is relevant for modeling the particular tag. Another advantage of learning the basis functions is that it may allow a much smaller-sized state transition matrix: using the DCT basis functions instead of the learned ones may require more basis functions to capture the timbral information and hence a higher-dimensional state vector. Estimating a smaller-sized state transition matrix is more efficient and expected to be less prone to overfitting. The benefits of learning the basis functions will be validated in

[2]This decorrelation is usually convenient in that it reduces the number of parameters to be estimated.

Section 2.7.3 (see, e.g., Table 2.7). 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 our experiments, we use 34 Mel-frequency bins, computed from half overlapping, 46ms windows of audio. The Mel-frequency bins are represented in a dB scale, which accurately accounts for the human auditory response to acoustic stimuli. Each audio fragment is described by a time series $y^t_{1:\tau}$ of $\tau = 450$ sequential audio feature vectors, which corresponds to 10 seconds. Song-level DTM models are learned from a dense sampling of audio fragments of 10 seconds, extracted every 1 second.

2.7 Experimental Evaluation

In this section we present results on music annotation and retrieval using the DTM model.

2.7.1 Experimental Setup

We set the state-space dimension $n = 7$, as in the work by Barrington et al. [13]. Song-level DTMs are learned with $K^{(s)} = 16$ components to capture enough of the temporal diversity present in each song, using EM-DTM [32]. Tag-level DTMs are learned by pooling together all song-level models associated with a given tag and reducing the result to a DTM with $K^{(a)} = 2$ components with HEM-DTM. We keep $K^{(a)}$ low to prevent HEM-DTM from overfitting (compared to HEM-GMM, HEM-DTM requires estimating significantly more parameters per mixture component). Section 2.7.3 illustrates that the system is fairly robust for reasonable variations in these parameters.

The EM-DTM algorithm to estimate song-level DTMs follows an iterative "component splitting" procedure. First, a one-component mixture is estimated by initializing parameters randomly and running EM till convergence. Then, the number of components

is increased by splitting this component and EM is run to convergence again. This process of splitting components and re-running EM for a mixture with more components is repeated until the desired number of components is obtained. When splitting a component, new components are initialized by replicating the component and slightly perturbing — randomly and differently for each new component — the poles of the state transition matrix, A. We follow a growing schedule of $\{1, 2, 4, 8, 16\}$ mixture components. The single component of the initial mixture is learned from a set of randomly selected fragments of the song, using the method proposed by Doretto et al. [54]. This "component splitting" procedure for EM-DTM was found to be quite robust to different initializations. More details can be found in earlier work by Chan et al. [32]. The tag-level DTMs (with $K^{(a)} = 2$ components) are learned by running 10 trials of the HEM-DTM algorithm. Each trial is initialized by randomly selecting 2 mixture components from the aggregated song-level mixtures. The final parameter estimates are obtained from the trial that achieves the highest likelihood. This procedure proved robust as well.

To investigate the advantage of the DTM's temporal representation, we compare the auto-tagging performance of HEM-DTM to the hierarchically trained Gaussian mixture models (HEM-GMM) [159], the CBA model [81], the boosting approach [55], and the SVM approach [106]. We follow the original procedure for training HEM-GMM and CBA, with the modification that the CBA codebook is constructed using only songs from the training set. We report performance also for direct-EM model estimation (EM-DTM), which learns each tag-level DTM model using the standard EM algorithm for DTM [32] directly on a subsampled set of all audio fragments associated with the tag. Empirically, we found that due to RAM requirements a single run of EM-DTM only manages to process about 1% of the data (i.e., audio fragments) that HEM-DTM can process, when estimating a tag model from approximately 200 training examples, on a modern laptop with 4GB of RAM. In contrast, HEM-DTM, through the estimation of

intermediate models, can pool over a much richer training data set, both in the number of songs and in the density of audio fragments sampled within each song. Finally, we compare to model aggregation DTM (AGG-DTM), which estimates each tag-level model by aggregating all the song-level DTM models associated with the tag. A drawback of this technique is that the number of DTs in the tag-level DTM models grows linearly with the size of the training set, resulting in drawn out delays in the evaluation stage. All reported metrics are the results of 5-fold cross validation where each song appeared in the test set exactly once.

2.7.2 Evaluation of Annotation and Retrieval

Annotation performance is measured following the procedure described by Turnbull et al. [159]. Test set songs are annotated with the 10 most likely tags in their semantic multinomial (Eq. 2.4). Annotation accuracy is reported by computing precision, recall and F-score for each tag[3], and then averaging over all tags. Per-tag precision is the probability that the model correctly uses the tag when annotating a song. Per-tag recall is the probability that the model annotates a song that should have been annotated with the tag. Precision, recall and F-score measure for a tag w are defined as:

$$\mathrm{P} = \frac{|w_C|}{|w_A|},\ \mathrm{R} = \frac{|w_C|}{|w_H|},\ \mathrm{F} = 2\left((\mathrm{P})^{-1} + (\mathrm{R})^{-1}\right)^{-1}, \tag{2.15}$$

where $|w_H|$ is the number of tracks that have w in the ground truth, $|w_A|$ is the number of times our annotation system uses w when automatically tagging a song, and $|w_C|$ is the number of times w is correctly used. In case a tag is never selected for annotation, the corresponding precision (that otherwise would be undefined) is set to the tag prior from

[3]We compute annotation metrics on a *per-tag* basis, as our goal is to build an automatic tagging algorithm with high stability over a wide range of semantic tags. *Per-song* metrics may get artificially inflated by consistently annotating songs with a small set of highly frequent tags, while ignoring less common tags.

Table 2.1. Annotation and retrieval results for various algorithms on CAL500.

	Annotation			Retrieval		
Model	P	R	F-score	AROC	MAP	P10
HEM-GMM [159]	**0.49**	0.23	0.26	0.66	0.45	0.47
CBA [81]	0.41	0.24	0.25	**0.69**	0.47	0.49
Boosting [55]	0.37	0.17	0.20	**0.69**	0.47	0.49
SVM [106]	0.38	0.24	0.25	0.66	0.45	0.48
HEM-DTM	0.47	**0.25**	**0.30**	**0.69**	**0.48**	**0.53**
AGG-DTM	0.34	0.23	0.21	**0.69**	0.47	0.50
EM -DTM	0.46	0.24	0.28	0.65	0.44	0.49

the training set, which equals the performance of a random classifier.

To evaluate retrieval performance, we rank-order test songs for each single-tag query in our vocabulary, as described in Section 2.4. We report mean average precision (MAP), area under the receiver operating characteristic curve (AROC) and top-10 precision (P10), averaged over all the query tags. The ROC curve is a plot of true positive rate versus false positive rate as we move down the ranked list. The AROC is obtained by integrating the ROC curve, and it is upper bounded by 1. Random guessing would result in an AROC of 0.5. The top-10 precision is the fraction true positives in the top-10 of the ranking. MAP averages the precision at each point in the ranking where a song is correctly retrieved.

2.7.3 Results on CAL500

Annotation and retrieval results on the CAL500 data set are presented in Table 5.1. For all metrics, except for precision, the best performance is observed with HEM-DTM. For retrieval, while some other methods show a comparable AROC score, HEM-DTM clearly improves the top of the ranked list compared to any other method. The higher precision-at-10 score demonstrates this. The results also show that sub-sampling of the training set for direct-EM estimation (EM-DTM) degrades the performance, compared

Table 2.2. Annotation and retrieval performance as a function of $K^{(s)}$ and $K^{(a)}$, respectively.

(a) $K^{(a)} = 2$				
$K^{(s)}$	2	4	8	16
MAP	0.4748	0.4768	0.4804	0.4834
F-score	0.2785	0.2886	0.2963	0.3002
(b) $K^{(s)} = 16$				
$K^{(a)}$	2	4	8	16
MAP	0.4834	0.4796	0.4820	0.4798
F-score	0.3002	0.2940	0.2942	0.2872

to HEM estimation (HEM-DTM). Aggregating the song-level DTMs associated with a tag (AGG-DTM) is also inferior. Figure 6.7 plots the precision-recall curves, for annotation, for all methods. At low recall (shorter, more selective annotations), (H)EM-DTM outperforms any other method. At higher recall, HEM-GMM catches up. In future work, we will investigate whether combining DTM- and GMM-based annotations can make for a more accurate auto-tagger.

To illustrate how different values of $K^{(s)}$ and $K^{(a)}$ (the number of components in the song and tag mixture models, respectively) affect the system's performance, we vary $K^{(s)}$ in $\{2, 4, 8, 16\}$ while fixing $K^{(a)} = 2$, and, vice versa, vary $K^{(a)}$ in $\{2, 4, 8, 16\}$ while fixing $K^{(s)} = 16$. Annotation and retrieval results are reported in Table 2.2, showing that performance is fairly robust within a reasonable parameter range.

We expect DTMs to be particularly beneficial for tags with characteristic temporal dynamics (e.g., tempo, rhythm, etc.) that unfold in the time span of a few seconds. Tags that are modeled adequately already by instantaneous spectral characteristics within a window of 50ms (e.g., timbre) may not benefit much, as well as tags that might require a global, structured song model.

To illustrate this point, Table 6.7 lists annotation (F-score) and retrieval (MAP)

results for a subset of the CAL500 vocabulary.

DTMs prove suitable for tags with significant temporal structure, e.g., vocal characteristics and instruments such as electric or acoustic guitar, by capturing the attack/sustain/decay/release profile of the instruments. DTMs also capture the temporal characteristics of a "fast" song — expected to unfold "fast", i.e., within a few seconds — and significantly improve upon GMMs, which cannot model these characteristics. For "slow" songs, on the other hand, DTMs are not picking up any additional information that GMMs do not capture already. The same is observed when predicting tags such as "light beat" and "mellow", already well described by timbre information (as evidenced by the high GMM performance), or "weak" and "sad", where neither DTMs nor GMMs are capturing strongly predictive acoustic characteristics. While, for some tags, this may indicate that "timbre tells all", for others, capturing more specific characteristics might require modeling structure at a much longer time scale or higher level. This will be a topic of future research.

It should also be noted that the increased modeling power of DTMs, compared to GMMs, requires more training data to reliably estimate them. This is discussed in more detail later in this section. Especially when training data is more noisy (e.g., for more subjective tags), significantly more examples will be required to make stand out the salient attributes HEM-DTM is trying to capture. This may explain why DTMs improve over GMMs for "positive feelings" (over 170 examples in CAL500) but not for "negative feelings (less than 80 examples). The same consideration holds for the usage tags "driving" and "going to sleep", which respectively appear 141 and 56 times in CAL500. So, while DTMs can capture a superset of the information modeled by GMMs, they may still perform worse for some tags, for this reason. Another factor to keep in mind when observing worse DTM than GMM performance is the more limited modeling power of DTMs when no clear temporal dynamics are present. Indeed, the absence of

Table 2.3. Annotation and retrieval results for some tags with HEM-DTM and HEM-GMM.

Tag	HEM-DTM		HEM-GMM [159]	
	F-score	MAP	F-score	MAP
HEM-DTM outperforms HEM-GMM				
female lead vocals	0.58	0.69	0.42	0.44
male lead vocals	0.44	0.87	0.08	0.81
backing vocals	0.30	0.47	0.09	0.44
emotional vocals	0.37	0.42	0.05	0.27
vocal harmonies	0.21	0.21	0.10	0.15
pop	0.32	0.36	0.31	0.35
acoustic guitar	0.44	0.44	0.31	0.43
electric guitar	0.32	0.37	0.14	0.33
fast	0.40	0.48	0.20	0.42
positive feelings	0.30	0.51	0.10	0.48
driving	0.29	0.38	0.29	0.33
HEM-DTM under-performs HEM-GMM				
light beat	0.36	0.58	0.53	0.61
mellow	0.34	0.41	0.37	0.49
slow	0.45	0.60	0.44	0.62
weak	0.22	0.26	0.26	0.25
sad	0.13	0.23	0.28	0.30
negative feelings	0.27	0.29	0.32	0.30
going to sleep	0.27	0.33	0.35	0.36

clear regularities in the temporal dynamics will result in a degenerate linear dynamical system (e.g., $A = 0$), reducing each DT component to a Gaussian component. Clearly, a DTM with 2 Gaussian components is a less rich timbre model than a GMM with 16 mixture components (as proposed by Turnbull et al. [159]). Estimating a DTM with as many mixture components, on the other hand, is prone to overfitting. This would result in a poorer timbre model as well.

Table 2.4 reports automatic 10-tag annotations for some songs from the CAL500 music collection, with HEM-DTM and HEM-GMM. Table 2.5 and 2.6 show the top-10

retrieval results for the queries “acoustic guitar” and “female lead vocals” respectively, both for HEM-DTM and HEM-GMM. For “acoustic guitar”, it is noted that both GMM and DTM make some “acceptable” mistakes. For example, “Golden brown”, by The Stranglers, has a harpsichord, and Aaron Neville’s “Tell it like it is” has clean electric guitar.

We investigate how the size of the training set affects the quality of the resulting models. As suggested earlier, reliably estimating the more powerful but also more complex DTM models is expected to require more training examples, compared to estimating GMM models. Figure 2.4 illustrates this. In Figure 2.4(a), we consider all 149 CAL500 tags and plot the relative retrieval performance of HEM-DTM, compared to HEM-GMM, for tag subsets of different minimal cardinality. The cardinality of a tag is defined as the number of examples in the data set that are associated with the tag. The minimal cardinality of a set of tags is determined by its tag with lowest cardinality. The plot shows that DTM modeling provides a bigger performance boost, over GMMs, when more examples are available for a tag. This is confirmed in Figure 2.4(b). This experiment is restricted to the 10 CAL500 tags that have cardinality of 150 or more. For each tag, the size of the training set is varied from 25 to 150, by random subsampling. Finally, the average retrieval performance (over these 10 tags) is reported as a function of the training set size, both for HEM-DTM and HEM-GMM. Initially, a larger training set benefits both methods. However, while GMM performance levels off beyond 100 training examples, DTM performance keeps improving. Additional training examples keep leveraging more of the DTM’s extra modeling potential, widening the gap between DTM and GMM performance.

Finally, we validate our claim that learning the observation matrix C (i.e., the basis functions for the Mel-spectrum), rather than using the standard DCT basis, is beneficial as it combines a better representation of the features of the Mel-spectrum with a more

compact model of the temporal dynamics that are characteristic for a particular song or semantic tag of interest. In Table 2.7, we compare HEM-DTM, with a learned C-matrix, with HEM-DTM-DCT, where we modify the DT model to fix the observation matrix C to be the DCT basis functions. We report annotation and retrieval performance for an experimental setup similar to the one in Table 5.1, with $n = 7$, $K^{(s)} = 16$ and $K^{(a)} = 2$. For HEM-DTM-DCT, the first $n = 7$ DCT bases (ordered by frequency) are selected. We also analyze the effect of a higher-dimensional DCT basis for HEM-DTM-DCT, by increasing n to 13. HEM-DTM outperforms both HEM-DTM-DCT variants, which illustrates that learning the observation matrix C improves the performance over using a standard DCT basis. The small difference in performance between HEM-DTM-DCT for $n = 7$ and $n = 13$, respectively, suggests that overfitting on the (higher-dimensional) hidden state process may be neutralizing the benefits of a larger (fixed) basis, which allows to better represent the Mel-frequency spectrum, for $n = 13$.

2.7.4 Results on Swat10k

HEM-DTM scales well to larger music collections, like this data set. It efficiently estimates tag models from a large number of examples by breaking the problem down into intermediate steps. The annotation and retrieval results on Swat10k, presented in Table 2.8, demonstrate that this procedure to estimate DTMs is also accurate. On Swat10k, DTMs outperform GMMs for every performance metric reported[4], except for precision on the "acoustic" tags. The annotation results are obtained by annotating songs with the two most likely "genre" tags (ideally one main genre and one sub-genre), and with the 10 most likely acoustic tags. Precision-recall curves are shown in Figure 2.5,

[4]Swat10k is weakly labeled, i.e., song annotations are incomplete. Given enough positive training examples, this doesn't affect the estimation of *generative* models (see, e.g., [159, 28]), like GMM and DTM. For evaluation purposes, while this still allows relative comparisons, it will reduce the absolute value of some performance metrics, e.g., MAP and P10 that evaluate positive song-tag associations at the top of the ranking.

confirming the overall dominance of HEM-DTM over HEM-GMM for the annotation task. In summary, for both Swat10k tag categories, DTMs successfully capture temporal dynamics over a few seconds as well as instantaneous timbre information, providing more accurate models.

2.8 Discussion on the DTM model's parameters

illustrate qualitatively how variations in the acoustic characteristics of semantic tags are reflected in different DTM model parameters. We show how the dynamics of a musical tag affect the state transition matrix A and how the structure of the observation matrix C specializes for different tags. We also present some two-dimensional embeddings of tag models, showing that qualitatively similar musical tags give rise to qualitatively similar DTM models.

2.8.1 State Transition Matrix: Temporal Dynamics

Doretto et al. [53] describe the link between the location of the poles[5] of the state transition matrix, A, and the dynamics of the LDS. The higher a normalized frequency (i.e., the wider the angle between each of the conjugate poles and the positive real axis), the faster and more distinct the associated dynamics. On the other hand, if all the poles are on the positive real axis, there are no dynamics connected with the modes of the system. Secondly, the distances of the poles from the origin control the durations of the corresponding modes of the system. Poles closer to the origin require stronger excitement for their mode to persist in the system.

Figure 2.6(a) sketches the poles of the mixture components of the DTM models for the tags "Fast" and "Slow" respectively, from the experiment of Section 2.7. The

[5]Consider the decomposition $A = P\Lambda P^{-1}$, where Λ is a diagonal matrix containing the eigenvalues of A, and P is an orthogonal matrix whose columns are the corresponding eigenvectors. The eigenvalues are the poles of the system. The eigenvectors determine the corresponding modes, describing the system's characteristic oscillations.

location of the poles in the polar plane is in agreement with the intuition that the former is characterized by faster dynamics while the latter coincides with smoother variations. Figure 2.6(b) shows the location of the poles for some of the tags in the upper portion of Table 6.7, for which HEM-DTM shows improvements over HEM-GMM. HEM-DTM associates some distinct normalized frequencies with these tags. Finally, Figure 2.6(c) (top row) plots the poles for different types of guitar (electric, distorted electric and bass) and piano. The figure illustrates that acoustically similar instruments have similar dynamics. For example, the locations of the poles of acoustic guitar and clean electric guitar are fairly similar. Also, the various types of guitar show similar dynamics, while "Piano" is characterized by noticeably different normalized frequencies.

2.8.2 Observation Matrix: Instantaneous Spectral Content

While the state transition matrix, A, encodes rhythm and tempo, the observation matrix, C, accounts for instantaneous timbre. In particular, a DT model generates features into the subspace[6] spanned by the columns of the observation matrix.

The bottom row of Figure 2.6(c) displays the first three basis vectors for the guitar tags and the piano tag in the top row. Each semantic tag is modeled by distinct basis functions that fit its particular music qualities and timbre. In contrast, the DCT basis functions used for MFCCs are fixed a priori. When modeling Mel-frequency spectra as the output of a DTM model, dimensionality reduction and model estimation are coupled together. On the other hand, for MFCCs, the DCT is performed before model estimation and is not adapted to specific audio sequences.

2.8.3 The DTM: Timbre and Dynamics

For a more intuitive interpretation, Figure 2.7 represents the audio feature subspaces — whose first three basis functions are depicted in the bottom row of Figure 2.6(c) — for the previous guitar and piano tags as a point in a two-dimensional embedding. The relative positions of the points reflect the Martin distance ([1, 38]) between the DTs for the corresponding tags, which is related to the difference in principal angles of the observation matrices [170].

The figure indicates that DTM tag models corresponding to qualitatively similar tags generate audio features in similar subspaces. For example, note that the guitar-type models are well separated from the piano model along the horizontal axis, while smaller variations along the vertical coordinate are observed between the different types of guitars.

Tag-level DTMs (combining state transitions with an observation model) simultaneously model both the timbre and dynamics of a tag. In this subsection, we qualitatively examine how similar/different the resulting models are for different tags. In particular, t-SNE [162] is used to embed tag models in a two-dimensional space, based on the Kullback-Leibler (KL) divergence between DTMs. The KL divergence between two DTs can be computed efficiently with a recursive formula [35]. The KL divergence between two mixture models, not analytically tractable in exact form, can be approximated efficiently (see, e.g., [79]).

Figure 2.8 shows two-dimensional embeddings for different groups of tags, learned from CAL500. Figure 2.8(a) displays the embedding for "emotion" and "acoustic characteristics" tags. Qualitatively, the resulting embedding is consistent with the semantic meaning of the different tags. For example, the top-left protuberance of the cloud gathers tags associated with smooth acoustic sensations: from low-energy and

[6] This is exactly true when the observation noise is ignored or negligible, i.e., $R \to 0$.

romancing sounds, to relaxing, slow sonorities. In the bottom of the cloud prevails a sense of happiness, that, marching to the right of the plot, turns into energy, excitement and heavy beats.

Similarly, Figure 2.8(b) provides the two-dimensional embedding for tags of the category "genre". In the center of the cloud, there is a strong rock and pop influence, usually characterized by the sound of electric guitars, the beat of drums and melodies sung by male lead vocalists. Moving toward the top-left of the graph, music fills up with emotions typical of blues and finally evolves to more sophisticated jazzy sounds. In the bottom, we find hip hop, electronica and dance music, more synthetic sonorities often diffused in night clubs.

2.9 Conclusions

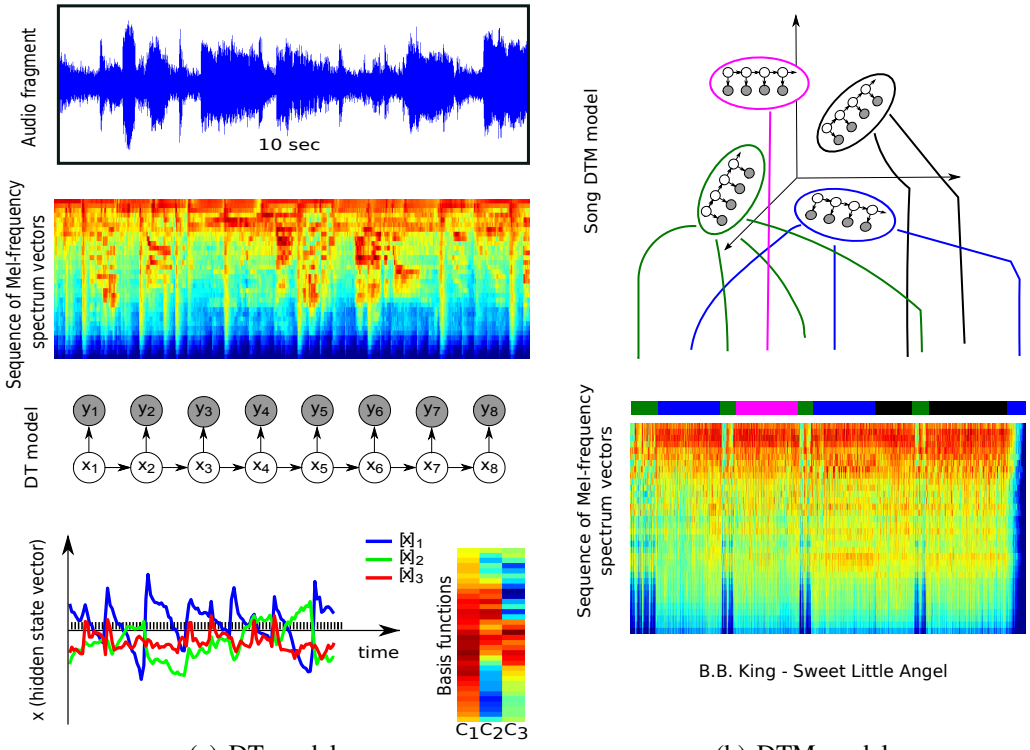
In this work, we have proposed a novel approach to automatic music annotation and retrieval that captures temporal (e.g., rhythmical) aspects as well as timbral content. In particular, our approach uses the dynamic texture mixture model, a generative time series model for musical content, as a tag-level annotation model. To learn the tag-level DTMs, we use a two-step procedure: 1) learn song-level DTMs from individual songs using the EM algorithm (EM-DTM); 2) learn the tag-level DTMs using a hierarchical EM algorithm (HEM-DTM) to summarize the common information shared by song-level DTMs associated with a tag. This hierarchical learning procedure is efficient and easily parallelizable, allowing DTM tag models to be learned from large sets of weakly-labeled songs (e.g., up to 2200 songs per tag in our experiments).

Experimental results demonstrate that the new DTM tag model improves accuracy over current bag-of-features approaches (e.g., GMMs, shown on the first line of Table 5.1), which do not model the temporal dynamics in a musical signal. In particular, we see significant improvements in tags with temporal structures that span several seconds, e.g.,

vocal characteristics, instrumentation, and genre. This leads to more accurate annotation at low recall, and improvements in retrieval at the top of the ranked-list. While, in theory, DTMs are a more general model than GMMs (as a DTM with degenerate temporal dynamics is equivalent to a GMM), we observe that in some cases GMMs are favorable. For musical characteristics that do not have distinctive long-term temporal dynamics, a GMM with more mixture components may be better suited to model pure timbre information, since it apriori ignores longer-term dynamics. A DTM with the same number of components, on the other hand, may overfit to the temporal dynamics of the training data, resulting in a poorer timbre model. Preventing overfitting by using a DTM with less mixture components apriori limits its flexibility as a pure timbre model though. This suggests that further gains are possible by using both DTM (longer-term temporal) and GMM (short-term timbre) annotation models. Future work will address this topic by developing criteria for selecting a suitable annotation model for a specific tag, or by combining results from multiple annotation models using the probabilistic formalism inherent in the generative models. Finally, our experiments show that DTM tag models perform significantly better when more training data is available. As an alternative to supplying more data, future work will consider learning DTM tag models using a Bayesian approach, e.g., by specifying suitable (data-driven) prior distributions on the DTM parameters, thus reducing the amount of training data required to accurately model the musical tag.

2.10 Acknowledgements

Chapter 2, in full, is a reprint of the material as it appears in the IEEE Transactions on Audio, Speech and Language Processing, 19(5):1343-1359, July 2011, E. Coviello, A. Chan, and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.



(a) DT model (b) DTM model

Figure 2.1. Dynamic texture music model: (a) a single DT represents a short audio fragment; (b) a DT mixture represents the heterogeneous structure of a song, with individual mixture components modeling homogeneous sections. The different orientations (and, locations) of the DT components in the top part of (b) are to visually suggest that each DT is characterized by a distinct set of parameters, to produce a specific type of audio fragments.

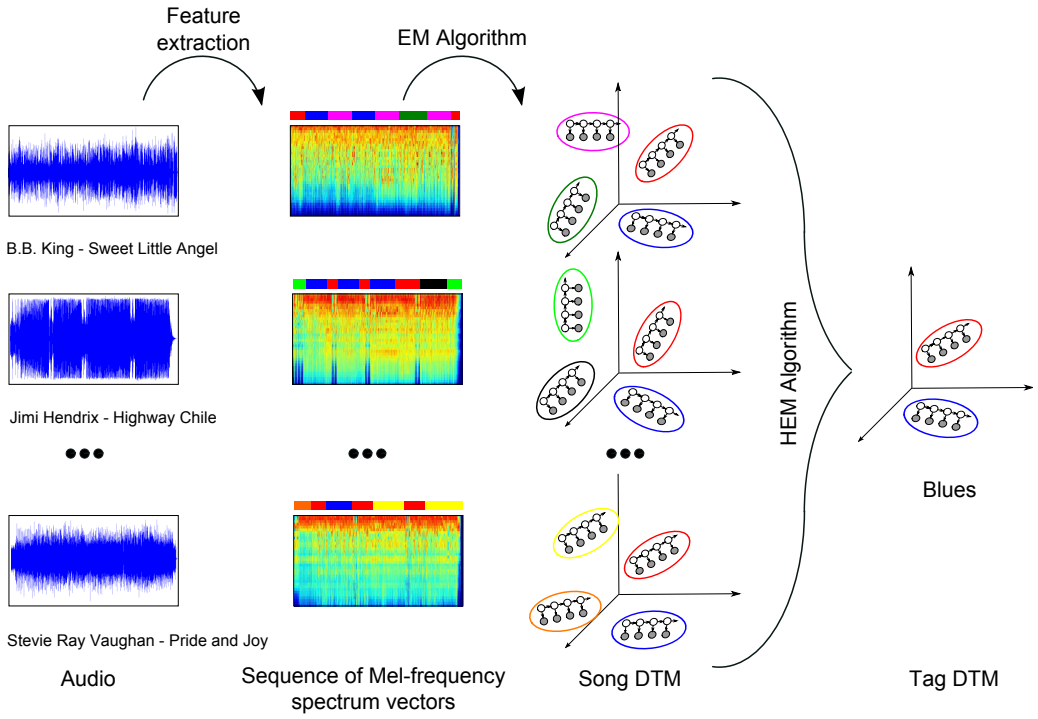


Figure 2.2. Learning a DTM tag model: first song-level DTMs are learned with EM for all songs associated with a tag, e.g., "Blues". Then, the song-level models are aggregated using HEM to find common features between the songs.

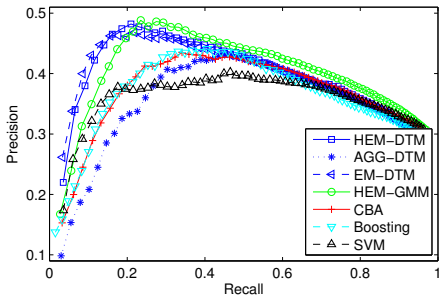


Figure 2.3. Precision-recall curves for different methods. (H)EM-DTM dominates at low recall. GMMs catch up at higher recall.

Algorithm 1. HEM algorithm for DTM

1: **Input**: combined song-level DTM $\{\pi_i^{(s)}, \Theta_i^{(s)}\}_{i=1}^{K^{(s)}}$, number of virtual samples N.
2: Initialize tag-level DTM $\{\hat{\pi}_j^{(a)}, \hat{\Theta}_j^{(a)}\}_{j=1}^{K^{(a)}}$.
3: **repeat**
4: {E-step} Compute expectations using sensitivity analysis for each $\Theta_i^{(s)}$ and $\hat{\Theta}_j^{(a)}$

$$\begin{aligned}
\hat{x}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}\left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(a)}}[x_t]\right], \\
\hat{P}_{t,t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}\left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(a)}}[x_t x_t^T]\right], \\
\hat{P}_{t,t-1|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}\left[\mathbb{E}_{x|y,\hat{\Theta}_j^{(a)}}[x_t x_{t-1}^T]\right], \\
\hat{W}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}\left[(y_t - \hat{\hat{y}}_j^{(a)})\mathbb{E}_{x|y,\hat{\Theta}_j^{(a)}}[x_t]^T\right], \\
\hat{U}_{t|j}^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}\left[(y_t - \hat{\hat{y}}_j^{(a)})(y_t - \hat{\hat{y}}_j^{(a)})^T\right], \\
\hat{u}_t^{(i)} &= \mathbb{E}_{y|\Theta_i^{(s)}}[y_t], \\
\ell_{i|j} &= \mathbb{E}_{\Theta_i^{(s)}}[\log p(y_{1:\tau}|\hat{\Theta}_j^{(a)})].
\end{aligned} \tag{2.11}$$

5: {E-step} Compute assignment probability and weighting:

$$\hat{\mathbf{z}}_{i,j} = \frac{\hat{\pi}_j^{(a)} \exp\left(N_i \ell_{i|j}\right)}{\sum_{j'=1}^{K^{(a)}} \hat{\pi}_{j'}^{(a)} \exp\left(N_i \ell_{i|j'}\right)}, \qquad \hat{\mathbf{w}}_{i,j} = \hat{\mathbf{z}}_{i,j} N_i = \hat{\mathbf{z}}_{i,j} \pi_i^{(s)} N. \tag{2.12}$$

6: {E-step} Compute aggregate expectations for each $\hat{\Theta}_j^{(a)}$:

$$\begin{aligned}
\hat{N}_j &= \textstyle\sum_i \hat{\mathbf{z}}_{i,j}, & \eta_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \hat{P}_{1,1|j}^{(i)}, \\
\hat{M}_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j}, & \gamma_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{u}_t^{(i)}, \\
\xi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \hat{x}_{1|j}^{(i)}, & \beta_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{x}_{t|j}^{(i)}, \\
\Phi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{P}_{t,t|j}^{(i)}, & \Psi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t,t-1|j}^{(i)}, \\
\varphi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t,t|j}^{(i)}, & \phi_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=2}^{\tau} \hat{P}_{t-1,t-1|j}^{(i)}, \\
\Lambda_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{U}_{t|j}^{(i)}, & \Gamma_j &= \textstyle\sum_i \hat{\mathbf{w}}_{i,j} \sum_{t=1}^{\tau} \hat{W}_{t|j}^{(i)}.
\end{aligned} \tag{2.13}$$

7: {M-step} Recompute parameters for each component $\hat{\Theta}_j^{(a)}$:

$$\begin{aligned}
\hat{C}_j^{(a)} &= \Gamma_j \Phi_j^{-1}, & \hat{R}_j^{(a)} &= \tfrac{1}{\tau \hat{M}_j}(\Lambda_j - \hat{C}_j^{(a)} \Gamma_j), & \hat{A}_j^{(a)} &= \Psi_j \phi_j^{-1}, \\
\hat{Q}_j^{(a)} &= \tfrac{1}{(\tau-1)\hat{M}_j}(\varphi_j - \hat{A}_j^{(a)} \Psi_j^T), & \hat{\mu}_j^{(a)} &= \tfrac{1}{\hat{M}_j} \xi_j, & \hat{S}_j^{(a)} &= \tfrac{1}{\hat{M}_j} \eta_j - \hat{\mu}_j^{(a)} (\hat{\mu}_j^{(a)})^T, \\
\hat{\pi}_j^{(a)} &= \tfrac{\hat{N}_j}{K^{(s)}}, & \hat{\hat{y}}_j^{(a)} &= \tfrac{1}{\tau \hat{M}_j}(\gamma_j - \hat{C}_j^{(a)} \beta_j).
\end{aligned} \tag{2.14}$$

8: **until** convergence
9: **Output**: tag-level DTM $\{\hat{\pi}_j^{(a)}, \hat{\Theta}_j^{(a)}\}_{j=1}^{K^{(a)}}$.

Table 2.4. Automatic 10-tag annotations for different songs. CAL500 ground truth annotations are marked in bold.

Kool and the Gang - "Funky stuff"	
HEM-DTM	**danceable**, **positive feelings**, driving, **party**, **cheerful**, **happy**, **energy**, **light**, **catchy**, **light-hearted**
HEM-GMM	weak, alternative, synthesized, **danceable**, major, cold, **indifferent**, driving, synthesizer, **unromantic**
Stevie Ray Vaughan - "Pride and joy"	
HEM-DTM	**drum set**, **romantic**, emotional vocals, **recommended**, **positive**, **like**, **cheerful**, emotional, **light**, alternative
HEM-GMM	weak, alternative, synthesized, synthesizer, not happy, unromantic, negative feelings, major, indifferent, cold
The Beach Boys - "I get around"	
HEM-DTM	**synthesizer**, **synthesized**, party, **electronica**, danceable, **heavy beat**, arousing, rough, **pop**, fast
HEM-GMM	**synthesized**, **electronica**, **synthesizer**, **pop**, party, cold, happy, danceable, **heavy beat**, **uptight**
The Police - "Every little thing she does is magic"	
HEM-DTM	**classic rock**, **driving**, **energy**, **fast**, **male lead vocals**, **electric guitar**, **electric**, indifferent, powerful, rough
HEM-GMM	boring, major, acoustic, **driving**, not likeable, female lead vocals, **recording quality**, cold, **synthesized**, pop , guitar
R.E.M. - "Camera"	
HEM-DTM	romantic, piano, **touching**, tender, pop, **pleasant**, **positive**, backing vocals, light beat, **calming**
HEM-GMM	weak, alternative, major, piano, synthesized, synthesizer, pop, female lead vocals, backing vocals, not likeable

Table 2.5. Top-10 retrieved songs for "acoustic guitar". Songs with acoustic guitar are marked in bold.

Rank	HEM-DTM	
1	The Zombies	"Beechwood park"
2	**James Taylor**	**"Fire and rain"**
3	**Arlo Guthrie**	**"Alice's restaurant massacree"**
4	**Crosby, Stills, Nash & Young**	**"Teach your children"**
5	**American Music Club**	**"Jesus hands"**
6	**Donovan**	**"Catch the wind"**
7	Smokey Robinson & The Miracles	"Ooo baby baby"
8	Aaron Neville	"Tell it like it is"
9	The Animals	"I'm crying"
10	**10cc**	**"For you and I"**
Rank	HEM-GMM	
1	The Stranglers	"Golden brown"
2	**Crosby, Stills, Nash & Young**	**"Teach your children"**
3	Pet Shop Boys	"Being boring"
4	The Police	"Every little thing she does is magic"
5	Belle and Sebastian	"Like Dylan in the movies"
6	Counting Crows	"Speedway"
7	The Beautiful South	"One last love song"
8	Beth Quist	"Survival"
9	Neutral Milk Hotel	"Where you'll find me now"
10	**James Taylor**	**"Fire and rain"**

Table 2.6. Top-10 retrieved songs for "female lead vocals". Songs with female lead vocals are marked in bold.

Rank	HEM-DTM	
1	**Artemis**	**"Don't look back"**
2	**The Ronettes**	**"Walking in the rain"**
3	**Alicia Keys**	**"Fallin"**
4	**Syreeta**	**"What love has joined together"**
5	**Dionne Warwick**	**"Walk on by"**
6	**Gloria Gaynor**	**"I will survive"**
7	**Anita Baker**	**"Caught up in the rapture"**
8	**The Andrews Sisters**	**"Boogie woogie bugle boy"**
9	**The Buggles**	**"Video killed the radio star"**
10	Antonio Carlos Jobim	"Wave"
Rank	**HEM-GMM**	
1	**Artemis**	**"Don't look back"**
2	**Britney Spears**	**"I'm a slave for you"**
3	**Alicia keys**	**"Fallin"**
4	**Dionne Warwick**	**"Walk on by"**
5	**Syreeta**	**"What love has joined together"**
6	The Beach Boys	"I get around"
7	Kourosh Zolani	"Peaceful planet"
8	**Natalie Imbruglia**	**"Torn"**
9	**Cat Power**	**"He war"**
10	The Animals	"I'm crying"

Table 2.7. Annotation and retrieval results for HEM-DTM and HEM-DTM-DCT ($n = 7$ and $n = 13$).

	Annotation			Retrieval		
Model	P	R	F-score	AROC	MAP	P10
HEM-DTM	**0.47**	**0.25**	**0.30**	**0.69**	**0.48**	**0.53**
HEM-DTM-DCT (n=7)	0.43	0.21	0.21	0.67	0.45	0.47
HEM-DTM-DCT (n=13)	0.42	0.21	0.21	0.68	0.45	0.47

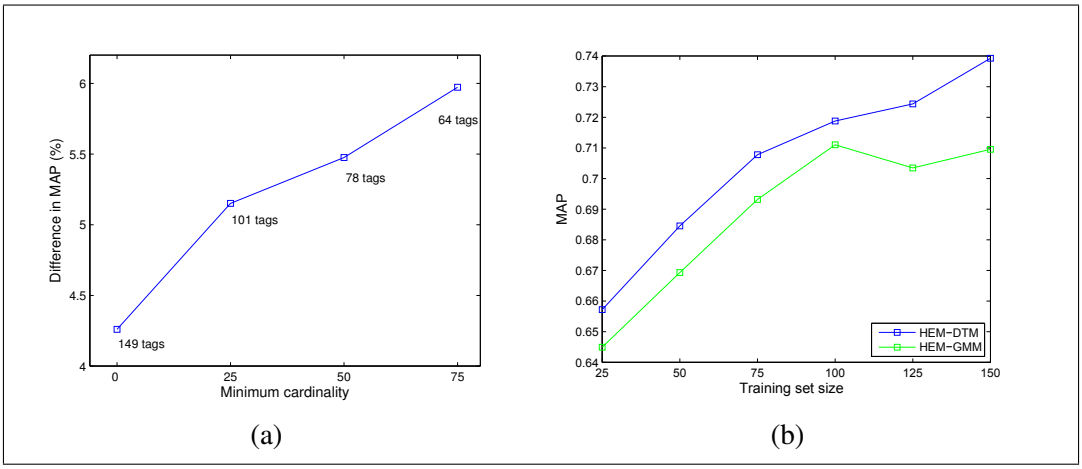
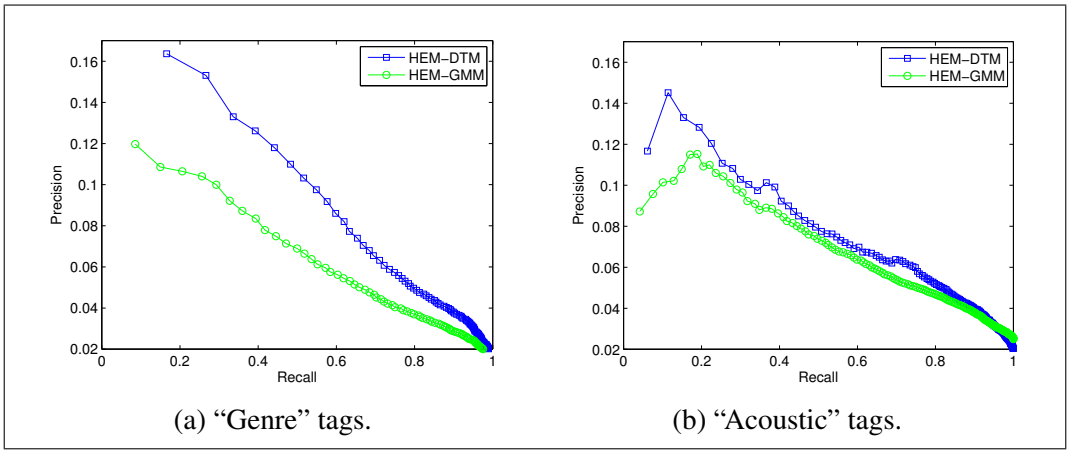


Figure 2.4. (a) Retrieval performance of HEM-DTM, relative to HEM-GMM, as a function of the minimal cardinality of tag subsets. More precisely, for each point in the graph, the set of all 149 CAL500 tags is restricted to those that CAL500 associates with a number of songs that is at least the abscissa value. The number of tags in each restricted subset is indicated next to the corresponding point in the graph. (b) Retrieval performance, averaged over the 10 CAL500 tags that have cardinality of 150 or more, as a function of training set size. Training sets of size 25, 50, ..., 150 are randomly subsampled.

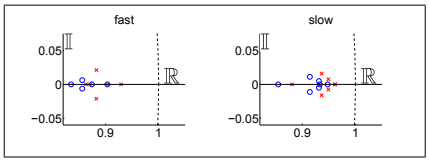
Table 2.8. Annotation and retrieval results on the Swat10k data set, for both tag categories.

Model	P	R	F-score	AROC	MAP	P10
			"Genre"			
HEM-DTM	**0.15**	**0.27**	**0.16**	**0.87**	**0.14**	**0.18**
HEM-GMM	0.11	0.15	0.08	0.82	0.11	0.15
			"Acoustic"			
HEM-DTM	0.10	**0.34**	**0.13**	**0.82**	**0.13**	**0.17**
HEM-GMM	**0.11**	0.24	0.10	0.77	0.10	0.14

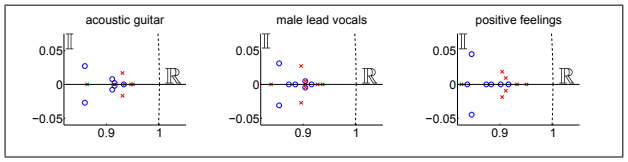


(a) “Genre” tags. (b) “Acoustic” tags.

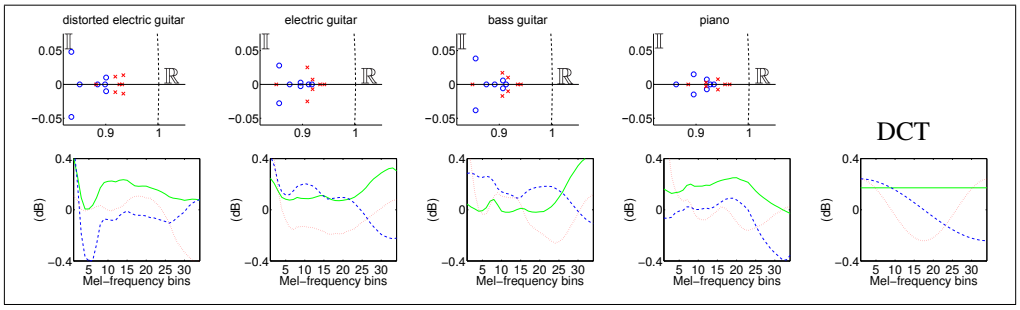
Figure 2.5. Precision-recall curves for annotation experiments on Swat10k, for both tag categories: (a) “genre” tags, and (b) “acoustic” tags.



(a) “Fast” versus “Slow”.



(b) Poles for some other tag models.



(c) Different types of guitar, and piano.

Figure 2.6. The location of the poles of the DTM models for different tags (blue circles and red crosses correspond to different DT components of the DTM). The horizontal and vertical axes of the figures represent the real and imaginary parts of the poles, respectively. The angle between each of the conjugate poles and the positive real axis determines the normalized frequency. (a) “Fast” shows higher normalized frequencies than “Slow”. (b) HEM-DTM captures clear dynamics for tags in the upper portion of Table 6.7, by modeling distinct normalized frequencies. (c) (Top row) Similar instruments are modeled with similar normalized frequencies. (Bottom row) Timbral characteristics are modeled by the observation matrix, C. The first three columns of C are depicted in solid green, dashed blue and dotted red line, for the corresponding tags in the top row. The columns of C define a basis that is optimized to best represent the instantaneous audio content for each tag. For comparison, the standard DCT basis (used to compute MFCCs) is shown on the far right.

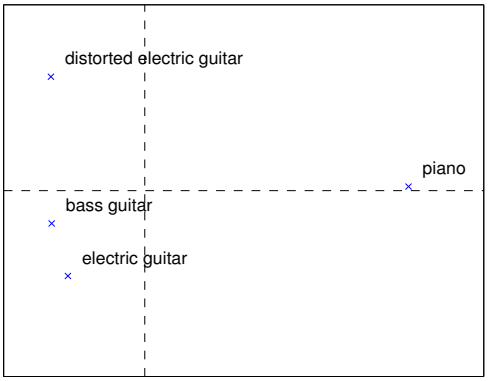
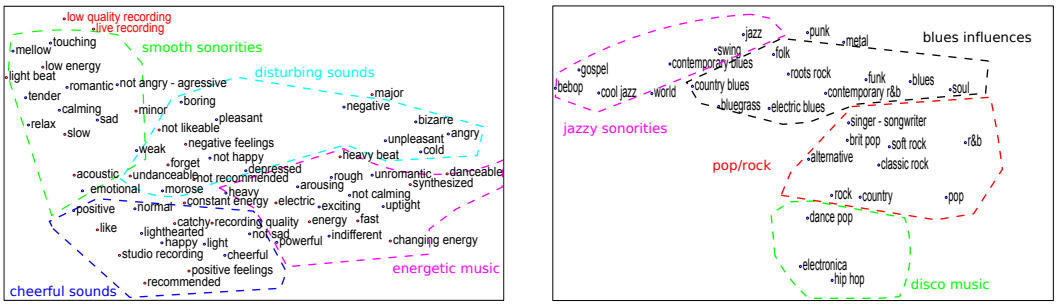


Figure 2.7. Each point represents the audio feature subspace of a different tag. The axes are unlabeled since only the relative positions of the points are relevant. The relative positions reflect similarity between the subspaces based on Martin distance. Guitar-type models are more similar, and clearly separated from the piano model.



(a) (b)

Figure 2.8. Two-dimensional embeddings of DTM-based tag models based on t-SNE and symmetrized KL divergence. Relative positions are qualitatively consistent with the semantic meaning of the tags. (a) “Emotion” and “Acoustic characteristics” tags. The top-left tip hosts smooth acoustic sensations. In the bottom prevails cheerful music and, moving right, energetic sounds. (b) “Genre” tags. The center gathers pop-rock sonorities. Moving towards the top-left tip, this evolves to sophisticated jazzy sounds. Hip hop, electronica and dance music are at the bottom of the plot.

Chapter 3

Multivariate Autoregressive Mixture Models for Music

3.1 Introduction

Browsing and discovery of new music can largely benefit from semantic search engines for music, which represent songs within a vocabulary of semantic tags, i.e., words or short phrases describing songs' attributes. By just typing the desired tags as in a standard text search engines (e.g., Bing or Google), users can find the music they desire.

Historically, attempts to map songs onto a semantic vocabulary have initially relied on available metadata (e.g. artist names, genre annotation, critical reviews), or manual labeling from expert human annotators and social networks. More recently, distributed human computation games, such as TagATune [98] and HerdIt [14], have attempted to scale up manual labeling to larger collections by recruiting non-expert users through engaging or rewarding games. However, these efforts have so far covered only a small portion of the songs available in modern music collections.[1] This motivates the development of content-based auto-tagging systems, i.e., intelligent algorithms that, by analyzing and understanding the acoustic content of songs, can automatically index them

[1] Pandora annotated catalog and TagATune labeled clips represent less than 5% and 0.15% of the iTunes' collection, respectively.

with semantic tags.

3.1.1 Related work

A large number of content-based auto-taggers are trained on a database of songs annotated with respect to a semantic vocabulary following a common scheme. First, a time series of low-level spectral features (e.g., Mel Frequency Cepstral Coefficients (MFCCs)) is extracted from each song in the database. Then, for each tag, a representative statistical model is fine tuned to capture the most predictive patterns common to the songs annotated with that tag. Once a new song is available, the auto-tagger uses the learned tag-models to process its low-level features and produces a vector of tag-affinities. The tag-affinities are then mapped onto a semantic multinomial (SMN), which represents the song within the semantic vocabulary.

A variety of auto-taggers, based on either generative models [160, 159, 140, 81] or discriminative models [106, 55, 60, 167, 150, 29], rely on a Bag-of-Features (BoF) representation of the spectral content of songs, which ignores temporal dynamics by treating all feature vectors as independent. While augmenting the spectral features with their first and second instantaneous derivatives has represented a common choice to enrich the BoF representation with temporal information (e.g. [159, 15]), more principled solutions have been implemented in recently proposed auto-taggers. The dynamic texture mixture (DTM) treats short fragments of audio as the output of a linear dynamical system [42]. The multi-scale learning algorithm in [73] leverages several pooling functions for summarization of the features over time. The Bag-of-Systems (BoS) approach represents songs with a codebook of time-series models [56]. The multivariate autoregressive (AR) model was used in [108] in a semi-parametric approach for genre classification of short musical snippets. In [143] various methods for temporal integration, including the AR model, were examined for musical instrument classification.

3.1.2 Original contribution

In this paper we introduce the autoregressive mixture (ARM) model [4] for automatic music annotation and retrieval. We first model each song as an ARM, estimated from a collection of audio-fragments extracted from the song. Note that this is different from estimating single AR models from *individual* audio clips as done in [108], since each mixture component of a song-level ARM models the music content of *several* (perceptually similar) audio fragments.

In order to model tag-level distributions as ARMs, we propose a novel efficient hierarchical expectation maximization algorithm for ARMs (HEM-ARM). Starting from all the song-ARMs that are relevant for a specific tag, the proposed algorithm summarizes the common music content by clustering similar AR components together, and learning a tag-ARM model with fewer components. We compare our HEM-ARM with previous auto-taggers that used GMMs [159] and DTMs [42], to model tag-level distribution, in tandem with an efficient HEM algorithm for learning. In particular, we obtain that HEM-DTM generally performs better than HEM-ARM (e.g., the annotation F-scores are 0.264, 0.254, respectively). However, relative to HEM-DTM, our HEM-ARM has significantly lower time requirements, both for training (two orders of magnitude) and for annotation (one order of magnitude). These results are explained by the differences in the *graphical* structures of the models. The DT model has an observed layer (which models the spectral content) and a hidden layer (that encodes the temporal dynamics). As a consequence, using DTMs can learn different frequency bases that better adapt to specific tags, but requires marginalization over the hidden variables — and hence delays — at each training iteration and for inference at annotation. On the opposite, the AR is a fully observable model. Hence, training and annotation can be implemented efficiently by computing sufficient statistics for each song a single time.

In addition, once songs are modeled with ARMs, we investigate a kernel-SVM method upon these song-ARMs for semantic retrieval, similar to the work done in [15] over GMMs and in [108] over single ARs. We test several kernel functions, some of which represent each song by the quantized frequency responses (QFR) of its AR components.

The remainder of this paper is organized as follows. In section 3.2 we present the autoregressive (mixture) model, and in section 3.3 we derive the hierarchical EM algorithm for ARMs. In Section 3.4 we present our kernel-SVM approach. In Section 6.6 we report our experiments.

3.2 The autoregressive mixture model

In this section we present the autoregressive (AR) model and the autoregressive mixture (ARM) model for music time series.

3.2.1 The AR model

A multivariate autoregressive (AR) model is a generative time-series model for audio fragments. Given a time series of T $d-$dimensional feature vectors $x_{1:T} \in \mathbb{R}^{d\times T}$, the AR model assumes each audio feature x_t at time t is a linear combination of the previous p audio features. Specifically, the AR model is described by the equation

$$x_t = \sum_{j=1}^{p} A_j x_{t-j} + v_t \tag{3.1}$$

where $\{A_j\}_{j=1}^{p}$ are p transition matrices of dimension $d \times d$. v_t is a driving noise process and is i.i.d. zero-mean Gaussian distributed, i.e., $v_t \sim \mathcal{N}(0,Q)$, where $Q \in \mathbb{R}^{d\times d}$ is a covariance matrix. The initial condition is specified by $x_1 \sim \mathcal{N}(\mu,S)$, where $S \in \mathbb{R}^{d\times d}$

is a covariance matrix. We can express (3.1) in a vectorial form:

$$x_t = \widetilde{A} x_{t-p}^{t-1} + \nu_t \tag{3.2}$$

where $\widetilde{A} = [A_1 \dots A_p] \in \mathbb{R}^{d \times dp}$ and $x_a^b = [x_b' \dots x_a']' \in \mathbb{R}^{dp \times 1}$. Note that, for convenience, we assume $x_t = 0$ for $t \in \{-p+1, \dots, 0\}$, and hence assume that x_1 triggered the generation of the whole time series. An AR model is hence parametrized by $\Theta = \{\mu, S, \widetilde{A}, Q\}$. The likelihood of a sequence $x_{1:T}$ is

$$p(x_{1:T}|\Theta) = \mathcal{N}(x_1|\mu, S) \prod_{t=2}^{T} \mathcal{N}(x_t| \sum_{j=1}^{p} A_j x_{t-j}, Q) \tag{3.3}$$

where $\mathcal{N}(\cdot|\mu, \Sigma)$ is the pdf of a Gaussian distribution with mean μ and covariance matrix Σ.

The parameters of an AR model can be estimated from a time-series $x_{1:T}$ with various optimization criteria [108, 118].

3.2.2 The ARM model

An ARM model treats a group of audio fragments as samples from K AR models. Specifically, for a given sequence, an assignment variable $z \sim \text{categorical}(\pi_1, \cdots \pi_K)$ selects one of the K AR models, where the i^{th} AR model is selected with probability π_i. Each mixture component is specified by the parameters $\Theta_i = \{\mu^i, S^i, \widetilde{A}^i, Q^i\}$, and the ARM model is specified by $\Theta = \{\pi_i, \Theta_i\}_{i=1}^K$. Whereas a single AR model suffices to describe an individual audio fragment, the ARM model is a more appropriate modeling choice for an entire song. This is motivated by the observation that a song usually shows significant structural variations within its duration, and hence multiple AR components are necessary to model the heterogeneous sections.

The likelihood of an audio fragment $x_{1:T}$ under an ARM model is

$$p(x_{1:T}|\Theta) = \sum_{i=1}^{K} \pi_i p(x_{1:T}|z=i,\Theta_i), \tag{3.4}$$

where the likelihood of $x_{1:T}$ under the i^{th} AR component $p(x_{1:T}|z=i,\Theta_i)$ is given by (3.3).

The parameters of an ARM model can be estimated from a collection of audio-fragments using the expectation maximization (EM) algorithm [49], which is an iterative procedure that alternates between estimating the assignment variables given the current estimate of the parameters, and re-estimating the parameters based on the estimated assignment variables.

3.3 The HEM algorithm for ARM models

In this paper we proposed to model tag distributions as ARM models. One way to estimate a tag-level ARM model is to run the EM algorithm directly on all the audio fragments extracted from the relevant songs. However, this approach would require excessive memory and computation time, to store all the input audio-sequences in RAM and to compute their likelihood at each iteration. In order to avoid this computational bottleneck, we propose a novel hierarchical EM algorithm for ARM models (HEM-ARM), which allows to learn ARM models using an efficient hierarchical estimation procedure. In a first stage, intermediate ARM models are estimated in parallel for each song, using the EM algorithm for ARMs on the song's audio fragments. Then, the HEM-ARM algorithm estimates the final model by summarizing the common information represented in the relevant song-ARMs. This is achieved by aggregating together all the relevant song-ARMs into a single big ARM model, and clustering similar AR models together to form the final tag-level ARM model.

At a high level, the HEM algorithm consists in maximum likelihood estimation of the ARM tag model from virtual samples distributed according to the song ARM models. However, since using the virtual samples can be approximated with a marginalization over the song ARM distribution (for the law of large numbers, see (3.8)), the estimation is carried out in an efficient manner that requires only knowledge of the parameters of the song models without the need of generating actual samples. The HEM algorithm was originally proposed by Vasconcelos and Lipmann [164] to reduce a GMM with a large number of mixture components to a compact GMM with fewer components, and extended to DTMs by Chan et al. [33]. The HEM algorithm has been successfully applied to the estimation of GMM tag-distribution [159] and DTM tag-distribution [42]. We now derive the HEM algorithm for ARMs.

3.3.1 Derivation of the HEM for ARMs

Formally, let $\Theta^{(s)} = \{\pi_i^{(s)}, \Theta_i^{(s)}\}_{i=1}^{K^{(s)}}$ be an ARM model with $K^{(s)}$ components, which pools together the ARM models of all the songs relevant for a tag. The goal of the HEM-ARM algorithm is to learn a tag-level ARM model $\Theta^{(t)} = \{\pi_j^{(t)}, \Theta_j^{(t)}\}_{j=1}^{K^{(t)}}$ with fewer components
(i.e., $K^{(t)} < K^{(s)}$), that represents $\Theta^{(s)}$ well. The likelihood of the tag ARM $\Theta^{(t)}$ is given by (3.4).

The HEM algorithm uses a set of N *virtual* samples generated from the base model $\Theta^{(s)}$, where the $N_i = N\pi_i^{(s)}$ samples $X_i = \{x_{1:\tau}^{(i,m)}\}_{m=1}^{N_i}$ are from the i^{th} component, i.e., $x_{1:\tau}^{(i,m)} \sim \Theta_i^{(s)}$. We assume that samples within each X_i are assigned to the same component of the tag model $\Theta^{(t)}$, and we denote the entire set of virtual samples with $X = \{X_i\}_{i=1}^{K^{(s)}}$.

The log likelihood of the incomplete data under $\Theta^{(t)}$ is

$$\begin{aligned}\log p(X|\Theta^{(t)}) &= \log \prod_{i=1}^{K^{(s)}} p(X_i|\Theta^{(t)}) \\ &= \log \prod_{i=1}^{K^{(s)}} \sum_{j=1}^{K^{(t)}} \pi_j^{(t)} p(X_i|\Theta_j^{(t)}).\end{aligned} \tag{3.5}$$

The HEM algorithm consists of the maximum likelihood estimation of the parameters of $\Theta^{(t)}$ from (3.5). Since (3.5) involves marginalizing over the hidden assignment variables $z_i^{(s)} \in \{1,\ldots,K^{(t)}\}$, its maximization can be solved with the EM algorithm. Hence, we introduce an indicator variable $\mathbf{z}_{i,j}$ for when the virtual audio sample set X_i is assigned to the j^{th} component of $\Theta^{(t)}$, i.e., when $z_i^{(s)} = j$. The complete data log-likelihood is then:

$$\begin{aligned}&\log p(X,Z|\Theta^{(t)}) = \\ &= \sum_{i=1}^{K^{(s)}} \sum_{j=1}^{K^{(t)}} \mathbf{z}_{i,j} \log \pi_j^{(t)} + \mathbf{z}_{i,j} \log p(X_i|\Theta_j^{(t)})\end{aligned} \tag{3.6}$$

The $\mathcal{Q}$-function is obtained by taking the conditional expectation of (3.6) with respect to Z, and the dependency on the virtual samples is removed by using the law of large numbers, i.e.,

$$\log p(X_i|\Theta_j^{(t)}) = N_i \frac{1}{N_i} \sum_{m=1}^{N_i} \log p(x_{1:\tau}^{(i,m)}|\Theta_j^{(t)}) \tag{3.7}$$

$$\approx N_i \, \mathbb{E}_{x_{1:\tau}|\Theta_i^{(s)}} \left[\log p(x_{1:\tau}|\Theta_j^{(t)}) \right]. \tag{3.8}$$

Note that (3.8) can be computed using the chain rule of the expected log-likelihood and

(3.1) to break the expectation

$$\mathbb{E}_{x_{1:\tau}|\Theta_i^{(s)}}\left[\log p(x_{1:\tau}|\Theta_j^{(t)})\right] = \tag{3.9}$$

$$= \sum_{t=1}^{\tau} \mathbb{E}_{x_{1:t}|\Theta_i^{(s)}}\left[\log p(x_t|x_{1:t-1},\Theta_j^{(t)})\right] \tag{3.10}$$

$$= \sum_{t=1}^{\tau} \mathbb{E}_{x_{1:t}|\Theta_i^{(s)}}\left[\log p(x_t|x_{t-p:t-1},\Theta_j^{(t)})\right] \tag{3.11}$$

$$= \sum_{t=1}^{\tau} \mathbb{E}_{x_{1:t-1}|\Theta_i^{(s)}}\left[\mathbb{E}_{x_t|x_{t-p:t-1},\Theta_i^{(s)}}\left[\log p(x_t|x_{t-p:t-1},\Theta_j^{(t)})\right]\right] \tag{3.12}$$

The inner expectation in (3.12) is the expected log-likelihood of a Gaussian, and its closed form solution depends on the first and second order statistics of $x_{t-p,t-1} \sim \Theta_i^{(s)}$. The outer expectation involves the computation of the expected first and second order statistics of $\Theta_i^{(s)}$, which can be carried out with the recursion presented in Algorithm 2. Note that, since the AR model $\Theta_j^{(t)}$ has no hidden variables, the computation of the expected sufficient statistics in Algorithm 2 is independent of $\Theta_j^{(t)}$, and hence needs to be executed only once for each input component $\Theta_i^{(s)}$.

If hidden variables are present (which is the case for the DT components of the DTM model, but not for the AR model), computing the expected sufficient statistics of a song component $\Theta_i^{(s)}$ involves marginalizing over the hidden variables of $\Theta_j^{(t)}$, and hence needs to be repeated at every iteration for each $j = 1,\ldots,K^{(t)}$.

The E-step of the HEM consists of computing of the expected sufficient statistics in Algorithm 2, the assignments variables in (3.14) and (3.15), and the cumulative expected sufficient statistics in (3.16). The M-step maximizes the $\mathcal{Q}$-function with respect to $\Theta^{(t)}$, giving the updates in (A.19). The full HEM-ARM scheme is presented in Algorithm 3.

Algorithm 2. Expected sufficient statistics

1: **Input**: song-level AR model $\Theta_i^{(s)} = \{\mu, S, \widetilde{A}, Q\}$, length of virtual samples τ.

2: Compute expected sufficient statistics for $t = 1, \ldots, \tau - 1$:

$$\tilde{E}_1^{(i)} = \mathbb{E}_{x_1 \Theta_i^{(s)}}\left[x_1 x_1'\right] = \mu\mu' + S$$

$$\hat{E}_1^{(i)} = \mathbb{E}_{x_{-p+1:1}\Theta_i^{(s)}}\left[x_{-p+1}^1 {x_{-p+1}^1}'\right] = \begin{bmatrix} \tilde{E}_1^{(i)} & 0_{d\times(d-1)p} \\ 0_{(d-1)p\times d} & 0_{(d-1)p\times(d-1)p} \end{bmatrix}$$

For $t = 1, \ldots, \tau - 1$

$$\hat{E}_t^{(i)} = \mathbb{E}_{x_{1:t}|\Theta_i^{(s)}}\left[x_{t-p+1}^t {x_{t-p+1}^t}'\right] = \begin{bmatrix} \widetilde{A}\hat{E}_{t-1}^{(i)}\widetilde{A}' + Q & A\hat{E}_{t-1}^{(i)} \\ \hat{E}_{t-1}^{(i)}\widetilde{A}' & \hat{E}_{t-1}^{(i)} \end{bmatrix}_{(1:dp,1:dp)}$$

Endfor

3: Compute expected sufficient statistics:

$$\hat{E}^{(i)} = \sum_{t=1}^{\tau-1} \hat{E}_t^{(i)} \tag{3.13}$$

4: **Output**: expected sufficient statistics: $\hat{E}^{(i)}$.

3.4 Kernel-SVM approach

We then used a semi-parametric approach that leverages the ARM model at the song level, and kernel support vector machine (SVM) for retrieval. In particular, we first model each song as an ARM using the EM algorithm. Then, for each tag, we learn a binary SVM classifier over the train set, based on a notion of similarity between ARM models defined in terms on their proximity in parameter space. Finally, following [15], we use the SVMs' decision values as the relevance of a song for a tag, and use it for retrieval of test songs based on one-tag queries.

Since the AR parameters lie on a non-linear manifold, naïvely treating them as Euclidean vectors would not necessary produce a correct similarity score. Hence, in the remainder of this section, we present several kernel functions based on more appropriate similarity scores between autoregressive (mixture) models. In previous work, Meng and

Algorithm 3. HEM algorithm for ARM

Input: combined song-level ARM $\{\pi_i^{(s)}, \Theta_i^{(s)}\}_{i=1}^{K^{(s)}}$, number of virtual samples N.
Compute cumulative expected sufficient statistics $\hat{E}^{(i)}$ for each $\Theta_i^{(s)}$ using Algorithm 2
Initialize tag-level ARM, $\{\pi_j^{(t)}, \Theta_j^{(t)}\}_{j=1}^{K^{(t)}}$.
repeat
{E-step}
Compute expected log-likelihood for each $\Theta_i^{(s)}$ and $\Theta_j^{(t)}$:

$$\begin{aligned}\ell_{i|j} &= \mathbb{E}_{x_{1:\tau}|\Theta_i^{(s)}}[\log p(x_{1:\tau}|\Theta_j^{(t)})] \\ &= -\tfrac{d\tau}{2}\log 2\pi - \tfrac{1}{2}\log|S_j^{(t)}| \\ &\quad -\tfrac{1}{2}\mathrm{trace}{S_j^{(t)}}^{-1}[S_i^{(s)} + (\mu_j^{(t)} - \mu_i^{(s)})'(\mu_j^{(t)} - \mu_i^{(s)})] \\ &\quad -\tfrac{\tau-1}{2}\mathrm{trace}{Q_j^{(t)}}^{-1}Q_i^{(s)} - \tfrac{\tau-1}{2}\log|Q_j^{(t)}| \\ &\quad -\tfrac{1}{2}\mathrm{trace}[{Q_j^{(t)}}^{-1}(\widetilde{A}_j^{(t)} - \widetilde{A}_i^{(s)})\hat{E}^{(i)}(\widetilde{A}_j^{(t)} - \widetilde{A}_i^{(s)})']\end{aligned}$$

Compute assignment probability and weighting:

$$\hat{\mathbf{z}}_{i,j} = \frac{\pi_j^{(t)}\exp\left(N_i\ell_{i|j}\right)}{\sum_{j'=1}^{K^{(t)}}\pi_{j'}^{(t)}\exp\left(N_i\ell_{i|j'}\right)} \tag{3.14}$$

$$\hat{\mathbf{w}}_{i,j} = \hat{\mathbf{z}}_{i,j}N_i = \hat{\mathbf{z}}_{i,j}\pi_i^{(s)}N \tag{3.15}$$

Computed aggregated expectations for each $\hat{\Theta}_j^{(t)}$:

$$\begin{array}{ll}\hat{N}_j = \sum_i \hat{\mathbf{z}}_{i,j}, & \hat{M}_j = \sum_i \hat{\mathbf{w}}_{i,j}, \\ \hat{S}_j = \sum_i \hat{\mathbf{w}}_{i,j}[S_i^{(s)} + \mu_i^{(s)}(\mu_i^{(s)})'] & \hat{m}_j = \sum_i \hat{\mathbf{w}}_{i,j}\mu_i^{(s)} \\ \hat{V}_j = \sum_i \hat{\mathbf{w}}_{i,j}A_i^{(s)}\hat{E}^{(i)}(\widetilde{A}_i^{(s)})' & \hat{P}_j = \sum_i \hat{\mathbf{w}}_{i,j}\hat{E}^{(i)} \\ \hat{R}_j = \sum_i \hat{\mathbf{w}}_{i,j}\hat{E}^{(i)}(\widetilde{A}_i^{(s)})' & \hat{Q}_j = \sum_i \hat{\mathbf{w}}_{i,j}Q_i^{(s)}\end{array} \tag{3.16}$$

{M-step}
Recompute parameters for each component $\hat{\Theta}_j^{(t)}$:

$$\begin{array}{ll}\widetilde{A}_j^* = \hat{R}_j'\hat{P}_j^{-1} & Q_j^* = \frac{1}{(\tau-1)\hat{M}_j}(\hat{V}_i - A_j^*\hat{R}_j + \hat{Q}_j), \\ \mu_j^* = \frac{1}{\hat{M}_j}\hat{m}_j, & S_j^* = \frac{1}{\hat{M}_j}\hat{S}_j - \mu_j^*(\mu_j^*)', \\ \pi_j^* = \frac{\sum_{i=1}^{K^{(s)}}\hat{\mathbf{z}}_{i,j}}{K^{(s)}}. & \end{array} \tag{3.17}$$

until convergence
Output: tag-level ARM $\{\pi_j^{(t)}, \Theta_j^{(t)}\}_{j=1}^{K^{(t)}}$.

Shawe-Taylor [108] specialize the Probability Product Kernel [85] to the AR case, which

depends non-linearly on the AR parameters, and is define as:

$$\mathscr{K}_{\mathrm{AR}}(\Theta_a, \Theta_b) = \int_{x_{1:p}} (p(x_{1:p}|\Theta_a) p(x_{1:p}|\Theta_b))^{\rho}, \tag{3.18}$$

where $\rho = 0.5$ corresponds to the Battaccharyya affinity. Since a song-ARM is associated with several AR components, for retrieval we collect a decision value for each AR component, and then rank the songs according to the average of the corresponding decision values (PPK-AR). Note that we compute PPK between individual AR components of the song-ARMs. This is different from [108], which uses *single* ARs on individual audio snippets.

In addition, we experiment SVM classification in tandem with a probability product kernel between auotoregressive *mixture* models (PPK-ARM). Following an approximation by Jebara et al. [85], the PPK-ARM can be computed from the PPK between individual components as

$$\mathscr{K}_{\mathrm{ARM}}(\Theta^{(1)}, \Theta^{(2)}) = \sum_{a=1}^{K_s} \sum_{b=1}^{K_s} (\pi_a^{(1)} \pi_b^{(2)})^{\rho} \mathscr{K}_{\mathrm{AR}}(\Theta_a^{(1)}, \Theta_b^{(2)}). \tag{3.19}$$

We finally propose a novel descriptor of AR models based on their frequency responses, and compute a kernel between these descriptors. Since an AR is a linear time invariant (LTI) system, its dynamics can be characterized by a transfer function defined as:

$$H(s) = (I_d - \sum_{j=1}^{p} A_j s^{-j})^{-1} \in \mathbb{C}^{d \times d} \tag{3.20}$$

where $s \in \mathbb{C}$ is a complex number and I_d is the d dimensional identity matrix. The transfer function describes the cross influences of each pair of components of the audio feature

vectors. In particular, we sample the transfer function at 200 equally spaced points on the unit circle, and then sum the the absolute values of these matrices over 30 linearly spaced frequency bins, to get a representation of the system's frequency response. By concatenating the AR's μ parameter and the log values of these 30 frequency response matrices, we get a descriptor $\Delta \in \mathbb{R}^{(d+30d^2)\times 1}$, which we call quantized frequency response (QFR). Finally, we use a SVM over QFRs based on cosine-similarity (CS) kernel and radial basis function (RBF) kernel.[2]

3.5 Experiments

3.5.1 Data

We performed automatic music annotation on the CAL500 dataset (details in [159] and references therein), which is a collection of 502 popular Western songs by as many different artists, and provides binary annotations with respect to a vocabulary of semantic tags. In our experiments we consider the 97 tags associated to at least 30 songs in CAL500 (11 genre, 14 instrumentation, 25 acoustic quality, 6 vocal characteristics, 35 mood and 6 usage tags).

The acoustic content of a song (resampled at $22,050Hz$) is represented by computing a time-series of 34-bin Mel-frequency spectral (MFS) features, extracted over half overlapping windows of 92 msec of audio signal, i.e., every ~ 46 msec. Following the insight in recent work of Hamel et al. [73], MFS features where further projected on the first $d = 20$ principal components, which we estimated over the MFSs collected from the $10,870$ songs in the CAL10K dataset [156].

Song level ARMs were learned with $K = 4$ components and memory of $p = 5$ steps, from a dense sampling of audio fragments of length $T = 125$ (i.e., approximately

[2] The CS kernel is defined as $\mathscr{K}(a,b) = a'b/\sqrt{||a||_2||b||_2}$. The RBF kernel is defines as $\mathscr{K}(a,b) = \exp\{-\|a-b\|_2^2/\sigma\}$. We set the bandwidth σ of the RBF kernel to the descriptor dimension $dim(\Delta)$.

Table 3.1. Annotation and retrieval on CAL500, for HEM-ARM, HEM-DTM and HEM-GMM.

	annotation			**retrieval**			**time**	
	P	R	F	AROC	MAP	P@10	train	test
HEM-ARM	0.468	0.203	0.254	0.696	0.421	0.412	198m	41m
HEM-DTM	0.446	0.217	0.264	0.708	0.446	0.460	424h	482m
HEM-GMM	0.474	0.205	0.213	0.686	0.417	0.425	41m	38m

6s), extracted with 80% overlap.

3.5.2 Results with HEM-ARM

For each tag, all the relevant song ARMs were pooled together to form a big ARM, and a tag-level ARM with
$K^{(t)} = 8$ components was learned with the HEM-ARM algorithm (with $N = 1000$ virtual samples of length $\tau = 10$). To reduce the effects of low likelihood in high dimension, for annotation we smooth the likelihood (3.3) by $T \cdot d \cdot p$. We compare our HEM-ARM with hierarchically trained Gaussian mixture models (HEM-GMM) [159] and dynamic texture mixture models (HEM-DTM) [42].

On the test set, a novel test song is annotated with the 10 most likely tags, corresponding to the peaks in its semantic multinomial. Retrieval given a one tag query involves rank ordering all songs with respect to the corresponding entry in their semantic multinomials. Annotation is measured with average per-tag precision (P), recall (R), and f-score (F). Retrieval is measured by per-tag area under the ROC (AROC), mean average precision (MAP), and precision at the first 10 retrieved objects (P@10). Refer to [159] for a detailed definition of the metrics. All reported metrics are the result of 5 fold-cross validation.

In Table 3.1 we report annotation and retrieval results for HEM-ARM, HEM-DTM and HEM-GMM. In addition, we register the total time for the training stage,

which consist in the estimation of the 97 tag models over the 5 folds (and also includes the estimation of the 502 song-level models), as well as for the test stage, i.e., the automatic-annotation of the 502 songs with the 97 tags.

From Table 3.1 we note that the advantages of the proposed HEM-ARM relative to HEM-DTM are in terms of *computation efficiency*. While HEM-DTM performs better than HEM-ARM on each metric (except on annotation precision where HEM-ARM is better), HEM-ARM has significantly lower time requirements. Specifically, the training time for our HEM-ARM is two orders of magnitude lower than that for HEM-DTM. Similarly, our auto-tagger requires approximately 42 minutes for the test-stage, while the auto-tagger based on DTMs requires 482 minutes to accomplish the same task. These results are explained by comparing the *graphical structures* of the AR and DT models. While the AR is a fully observable model, the DT consists of an observed layer, which model the spectral content, and a hidden layer that encodes the temporal dynamics. Hence, DTMs have the advantage of learning different frequency basis to best represent specific tags [42]. However, the computation of the (expected) sufficient statistics with respect to each DT component requires marginalization of the hidden variables (see [33]). Hence, during training, it needs to be executed at each iteration of the learning algorithms for each input datum (i.e., audio-fragments for EM, and DTs for HEM) and for each individual component of the learned model; during annotation, it needs to be repeated for each audio-fragment and each DT component of the tag models. On the opposite, the corresponding statistics for ARMs involve no marginalization of hidden variables. Hence, during training, they need to be computed only a single time for each input datum (i.e., audio-fragments for EM, and ARs for HEM). In addition, during annotation, the sufficient statistics can be collected a single time for each song.

Finally, HEM-ARM performs favorably relative to HEM-GMM (which does not model temporal dynamics), while still requiring limited computation times. Since

Table 3.2. Annotation (F-score) and retrieval (AROC) performance of HEM-ARM, for different memories $p \in [1{:}9]$.

p	1	2	3	4	5	6	7	8	9
F-score	0.234	0.247	0.252	0.255	0.254	0.245	0.244	0.242	0.237
AROC	0.650	0.672	0.686	0.693	0.696	0.695	0.694	0.695	0.694

learning and inference are performed efficiently, HEM-ARM can leverage higher order memories to model temporal dynamics, without incurring in large delays. In particular, in Table 3.2 we plot annotation (F-score) and retrieval (AROC) performance as a a function of the memory p of the AR models. Performance are fairly stable for $p = 4,5,6$. Shorter memories (e.g., $p = 1,2,3$) do not suffices to capture the interesting dynamics, while too large values deteriorate annotation performance.

3.5.3 Results with kernel-SVM.

We implemented the kernel-SVM approach as described in Section 3.4. In particular, we learned song-ARMs with $K = 4$ components and memory $p = 5$, estimated from the $d = 20$ dimensional PCA-MFS features. We then computed the QFR-CS and QFR-RBF kernels based on the QFR descriptors, the PPK kernel between ARM (PPK-ARM), and the PPK kernel between individual AR components (PPK-AR). For comparison, we also considered PPK similarity between song-GMMs estimated on MFCC features [15] (GMM-PPK) and PPK similarity between single AR models estimated on the MFCC features of entire songs as proposed in [108] (MFCC-PPK-AR). We used the LibSVM software package [36] for the SVM, with all parameters selected using validation on the training set.

Retrieval scores are reported in Table 3.4, and are result of 5-fold cross validation. We note that these results are generally superior to those in Table 3.1, since the SVM is a discriminative algorithm and hence tends to be more robust on strongly labeled datasets

method		AROC	MAP	P@10	train	test
ARM based	tag-conditional ARM (generative)	0.696	0.421	0.412	198m	41m
	kernel SVM (PPK)	0.727	0.463	0.484	287m	194m
	kernel SVM (QFR + RBF)	0.723	0.469	0.488	137m	74m
GMM, kernel SVM (PPK)		0.696	0.436	0.454		
AR on snippets, kernel SVM (PPK)		0.706	0.447	0.463		

Table 3.3. Retrieval for the kernel-SVM approach. Including train and test times

Table 3.4. Retrieval for the kernel-SVM approach. Including train and test times

kernel		AROC	MAP	P@10	train	test
ARM based	PPK-ARM	0.717	0.448	0.459	233m	194m
	PPK-AR	0.727	0.463	0.484	287m	194m
	QFR-CS	0.717	0.461	0.479	125m	65m
	QFR-RBF	0.723	0.469	0.488	137m	74m
GMM-PPK [15]		0.696	0.436	0.454		
MFCC-PPK-AR [108]		0.706	0.447	0.463		

such as CAL500. In particular, the best performance was registered with the QFR-RBF and PPK-AR systems (score differences between them are not statistically significant). In addition, PPK similarity on ARMs (PPK-ARM) proves less performant, suggesting that the approximation in (3.19) may be not enough accurate for our task. Finally, PPK similarity on GMM performs the worst, since it does not leverage termporal dynamics, and MFCC-AR-PPK, which doesn't leverage mixtures, is also significantly behind.

3.6 Discussion

In this paper we have proposed the ARM model for music auto-tagging. We have derived a hierarchical EM algorithm for efficiently learning tag ARMs. We have showed that our HEM-ARM can estimate tag models significantly more efficiently than HEM-DTM, at the price of a small reduction in performance. We have also successfully tested a kernel-SVM approach based on several similarity functions based on the ARM

model.

3.7 Acknowledgements

Chapter 3, in full, is a reprint of the material as it appears in the Proceedings of the 13th International Society for Music Information Retrieval Conference, ISMIR 2012, Porto (Portugal), 8-12 October 2012, E. Coviello, Y. Vaizman, A.B. Chan and G. Lanckriet.

Chapter 4

Clustering Hidden Markov Models with Variational HEM

4.1 Introduction

The hidden Markov model (HMM) [135] is a probabilistic model that assumes a signal is generated by a double embedded stochastic process. A discrete-time hidden state process, which evolves as a Markov chain, encodes the dynamics of the signal, while an observation process encodes the appearance of the signal at each time, conditioned on the current state. HMMs have been successfully applied to a variety of fields, including speech recognition [135], music analysis [134] and identification [17], online hand-writing recognition [115], analysis of biological sequences [95], face recognition citepchuk2013understanding and clustering of time series data [86, 151, 6].

This paper is about clustering HMMs. More precisely, we are interested in an algorithm that, given a collection of HMMs, partitions them into K clusters of "similar" HMMs, while also learning a representative HMM "cluster center" that concisely and appropriately represents each cluster. This is similar to standard k-means clustering, except that the data points are HMMs now instead of vectors in $\mathbb{R}^d$.

Various applications motivate the design of HMM clustering algorithms, ranging from hierarchical clustering of sequential data (e.g., speech or motion sequences modeled

by HMMs as by [86]), to hierarchical indexing for fast retrieval, to reducing the computational complexity of estimating mixtures of HMMs from large datasets (e.g., semantic annotation models for music and video)—by clustering HMMs, efficiently estimated from many small subsets of the data, into a more compact mixture model of all data. However, there has been little work on HMM clustering and, therefore, its applications.

Existing approaches to clustering HMMs operate directly on the HMM *parameter* space, by grouping HMMs according to a suitable pairwise distance defined in terms of the HMM parameters. However, as HMM parameters lie on a non-linear manifold, a simple application of the k-means algorithm will not succeed in the task, since it assumes real vectors in a Euclidean space. In addition, such an approach would have the additional complication that HMM parameters for a particular generative model are not unique, i.e., a permutation of the states leads to the same generative model. One solution, proposed by [86], first constructs an appropriate similarity matrix between all HMMs that are to be clustered (e.g., based on the Bhattacharya affinity, which depends non-linearly on the HMM parameters [85, 77]), and then applies spectral clustering. While this approach has proven successful to group HMMs into similar clusters [86], it does not directly address the issue of generating HMM cluster centers. Each cluster can still be represented by choosing one of the given HMMs, e.g., the HMM which the spectral clustering procedure maps the closest to each spectral clustering center. However, this may be suboptimal for some applications of HMM clustering, for example in hierarchical estimation of annotation models. Another distance between HMM distributions suitable for spectral clustering is the KL divergence, which in practice has been approximated by sampling sequences from one model and computing their log-likelihood under the other [89, 178, 175].

Instead, in this paper we propose to cluster HMMs *directly* with respect to the *probability distributions* they represent. The probability distributions of the HMMs are

used throughout the whole clustering algorithm, and not only to construct an initial embedding as [86]. By clustering the output distributions of the HMMs, marginalized over the hidden-state distributions, we avoid the issue of multiple equivalent parameterizations of the hidden states. We derive a hierarchical expectation maximization (HEM) algorithm that, starting from a collection of input HMMs, estimates a smaller mixture model of HMMs that concisely represents and clusters the input HMMs (i.e., the input HMM distributions guide the estimation of the output mixture distribution).

The HEM algorithm is a *generalization* of the EM algorithm—the EM algorithm can be considered as a special case of HEM for a mixture of delta functions as input. The main difference between HEM and EM is in the E-step. While the EM algorithm computes the sufficient statistics given the observed data, the HEM algorithm calculates the expected sufficient statistics averaged over all possible observations generated by the input probability models. Historically, the first HEM algorithm was designed to cluster *Gaussian* probability distributions [164]. This algorithm starts from a Gaussian mixture model (GMM) with $K^{(b)}$ components and reduces it to another GMM with fewer components, where each of the mixture components of the reduced GMM represents, i.e., *clusters*, a group of the original Gaussian mixture components. More recently, [34] derived an HEM algorithm to cluster *dynamic texture* (DT) models (i.e., linear dynamical systems, LDSs) through their probability distributions. HEM has been applied successfully to construct GMM hierarchies for efficient image indexing [163], to cluster video represented by DTs [33], and to estimate GMMs or DT mixtures (DTMs, i.e., LDS mixtures) from large datasets for semantic annotation of images [28], video [33] and music [159, 42]. Note that HMMs cannot be clustered by using the original HEM by [164]. Specifically, the original formulation of HEM was designed for clustering data points represented by *individual* Gaussian models. When clustering HMMs, we are interested in assigning every HMM as a whole to a cluster, and do not want to treat

their individual Gaussian states independently. Even with GMMs (as opposed to single Gaussians) this is not possible in closed form, since it would need the expected log likelihood of a mixture.

To extend the HEM framework from clustering Gaussians to clustering HMMs, additional marginalization over the hidden-state processes is required, as with DTs. However, while Gaussians and DTs allow tractable inference in the E-step of HEM, this is no longer the case for HMMs. Therefore, in this work, we derive a variational formulation of the HEM algorithm (VHEM), and then leverage a variational *approximation* derived by [78] (which has not been used in a learning context so far) to make the inference in the E-step tractable. The resulting algorithm not only clusters HMMs, but also learns novel HMMs that are representative centers of each cluster. The resulting VHEM algorithm can be generalized to handle other classes of graphical models, for which exact computation of the E-step in the standard HEM would be intractable, by leveraging similar variational approximations—e.g., any mixtures of continuous exponential family distributions (e.g., Gaussian) the more general case of HMMs with emission probabilities that are (mixtures of) continuous exponential family distributions.

Compared to the spectral clustering algorithm of [86], the VHEM algorithm has several advantages that make it suitable for a variety of applications. First, the VHEM algorithm is capable of both clustering, as well as learning *novel* cluster centers, in a manner that is consistent with the underlying generative probabilistic framework. In addition, since it does not require sampling steps, it is also scalable with low memory requirements. As a consequence, VHEM for HMMs allows for efficient estimation of HMM mixtures from large datasets using a hierarchical estimation procedure. In particular, intermediate HMM mixtures are first estimated in *parallel* by running the EM algorithm on small independent portions of the dataset. The final model is then estimated from the intermediate models using the VHEM algorithm. Because VHEM is based

on maximum-likelihood principles, it drives model estimation towards similar optimal parameter values as performing maximum-likelihood estimation on the full dataset. In addition, by averaging over all possible observations compatible with the input models in the E-step, VHEM provides an implicit form of regularization that prevents over-fitting and improves robustness of the learned models, compared to a direct application of the EM algorithm on the full dataset. Note that, in contrast to [86], VHEM does not construct a kernel embedding, and is therefore expected to be more efficient, especially for large $K^{(b)}$.

In summary, the contributions of this paper are three-fold: i) we derive a variational formulation of the HEM algorithm for clustering HMMs, which generates novel HMM centers representative of each cluster; ii) we evaluate VHEM on a variety of clustering, annotation, and retrieval problems involving time-series data, showing improvement over current clustering methods; iii) we demonstrate in experiments that VHEM can effectively learn HMMs from large sets of data, more efficiently than standard EM, while improving model robustness through better regularization. With respect to our previous work, the VHEM algorithm for HMMs was originally proposed by [41]

The remainder of the paper is organized as follows. We review the hidden Markov model (HMM) and the hidden Markov mixture model (H3M) in Section 4.2. We present the derivation of the VHEM-H3M algorithm in Section 4.3, followed by a discussion in Section 4.4.boldsymbol Finally, we present experimental results in Sections 4.5 and 4.6.

4.2 The Hidden Markov (Mixture) Model

A hidden Markov model (HMM) $\mathscr{M}$ assumes a sequence of τ observations $\mathbf{y} = \{y_1, \ldots, y_\tau\}$ is generated by a double embedded stochastic process, where each observation (or emission) y_t at time t depends on the state of a discrete hidden variable x_t, and the sequence of hidden states $\mathbf{x} = \{x_1, \ldots, x_\tau\}$ evolves as a first-order Markov chain.

The hidden variables can take one of S values, $\{1,\ldots,S\}$, and the evolution of the hidden process is encoded in a state transition matrix $A = [a_{\beta,\beta'}]_{\beta,\beta'=1,\ldots,S}$, where each entry, $a_{\beta,\beta'} = p(x_{t+1} = \beta' | x_t = \beta, \mathscr{M})$, is the probability of transitioning from state β to state β', and an initial state distribution $\pi = [\pi_1,\ldots,\pi_S]$, where $\pi_\beta = p(x_1 = \beta|\mathscr{M})$.

Each state β generates observations according to an emission probability density function, $p(y_t|x_t = \beta, \mathscr{M})$. Here, we assume the emission density is *time-invariant*, and modeled as a Gaussian mixture model (GMM) with M components:

$$p(y|x = \beta, \mathscr{M}) = \sum_{m=1}^{M} c_{\beta,m} p(y|\zeta = m, x = \beta, \mathscr{M}), \tag{4.1}$$

where $\zeta \sim \text{multinomial}(c_{\beta,1},\ldots,c_{\beta,M})$ is the hidden assignment variable that selects the mixture component, with $c_{\beta,m}$ as the mixture weight of the mth component, and each component is a multivariate Gaussian distribution,

$$p(y|\zeta = m, x = \beta, \mathscr{M}) = \mathscr{N}(y; \mu_{\beta,m}, \Sigma_{\beta,m}),$$

with mean $\mu_{\beta,m}$ and covariance matrix $\Sigma_{\beta,m}$. The HMM is specified by the parameters

$$\mathscr{M} = \{\pi, A, \{\{c_{\beta,m}, \mu_{\beta,m}, \Sigma_{\beta,m}\}_{m=1}^{M}\}_{\beta=1}^{S}\},$$

which can be efficiently learned from an observation sequence $\mathbf{y}$ with the Baum-Welch algorithm [135], which is based on maximum likelihood estimation.

The probability distribution of a state sequence $\mathbf{x}$ generated by an HMM $\mathscr{M}$ is

$$p(\mathbf{x}|\mathscr{M}) = p(x_1|\mathscr{M}) \prod_{t=2}^{\tau} p(x_t|x_{t-1}, \mathscr{M}) = \pi_{x_1} \prod_{t=2}^{\tau} a_{x_{t-1},x_t},$$

while the joint likelihood of an observation sequence $\mathbf{y}$ and a state sequence $\mathbf{x}$ is

$$p(\mathbf{y},\mathbf{x}|\mathscr{M}) = p(\mathbf{y}|\mathbf{x},\mathscr{M})p(\mathbf{x}|\mathscr{M}) = p(x_1|\mathscr{M})\prod_{t=2}^{\tau} p(x_t|x_{t-1},\mathscr{M})\prod_{t=1}^{\tau} p(y_t|x_t,\mathscr{M}).$$

Finally, the observation likelihood of $\mathbf{y}$ is obtained by marginalizing out the state sequence from the joint likelihood,

$$p(\mathbf{y}|\mathscr{M}) = \sum_{\mathbf{x}} p(\mathbf{y},\mathbf{x}|\mathscr{M}) = \sum_{\mathbf{x}} p(\mathbf{y}|\mathbf{x},\mathscr{M})p(\mathbf{x}|\mathscr{M}), \tag{4.2}$$

where the summation is over all state sequences of length τ, and can be performed efficiently using the *forward algorithm* [135].

A hidden Markov mixture model (H3M) [151] models a set of observation sequences as samples from a group of K hidden Markov models, each associated to a specific sub-behavior. For a given sequence, an assignment variable

$$z \sim \text{multinomial}(\omega_1,\cdots,\omega_K)$$

selects the parameters of one of the K HMMs, where the kth HMM is selected with probability ω_k. Each mixture component is parametrized by

$$\mathscr{M}_z = \{\pi^z, A^z, \{\{c^z_{\beta,m}, \mu^z_{\beta,m}, \Sigma^z_{\beta,m}\}_{m=1}^{M}\}_{\beta=1}^{S}\},$$

and the H3M is parametrized by $\mathscr{M} = \{\omega_z, \mathscr{M}_z\}_{z=1}^{K}$, which can be estimated from a collection of observation sequences using the EM algorithm [151, 6].

To reduce clutter, here we assume that all the HMMs have the same number S of hidden states and that all emission probabilities have M mixture components. Our derivation could be easily extended to the more general case though.

4.3 Clustering Hidden Markov Models

Algorithms for clustering HMMs can serve a wide range of applications, from hierarchical clustering of sequential data (e.g., speech or motion sequences modeled by HMMs [86]), to hierarchical indexing for fast retrieval, to reducing the computational complexity of estimating mixtures of HMMs from large weakly-annotated datasets—by clustering HMMs, efficiently estimated from many small subsets of the data, into a more compact mixture model of all data.

In this work we derive a hierarchical EM algorithm for clustering HMMs (HEM-H3M) with respect to their probability distributions. We approach the problem of clustering HMMs as reducing an input HMM mixture with a large number of components to a new mixture with fewer components. Note that different HMMs in the input mixture are allowed to have different weights (i.e., the mixture weights $\{\omega_z\}_{z=1}^{K}$ are not necessarily all equal).

One method for estimating the reduced mixture model is to generate samples from the input mixture, and then perform maximum likelihood estimation, i.e., maximize the log-likelihood of these samples. However, to avoid explicitly generating these samples, we instead maximize the *expectation* of the log-likelihood with respect to the input mixture model, thus averaging over all possible samples from the input mixture model. In this way, the dependency on the samples is replaced by a marginalization with respect to the input mixture model. While such marginalization is tractable for Gaussians and DTs, this is no longer the case for HMMs. Therefore, in this work, we i) derive a variational formulation of the HEM algorithm (VHEM), and ii) specialize it to the HMM case by leveraging a variational approximation proposed by [78]. Note that the work of [78] was proposed as an alternative to MCMC sampling for the computation of the KL divergence between two HMMs, and has not been used in a learning context so far.

We present the problem formulation in Section 4.3.1, and derive the algorithm in Sections 4.3.2, 4.3.3 and 4.3.4.

4.3.1 Formulation

Let $\mathscr{M}^{(b)}$ be a base hidden Markov mixture model with $K^{(b)}$ components. The goal of the VHEM algorithm is to find a reduced hidden Markov mixture model $\mathscr{M}^{(r)}$ with $K^{(r)} < K^{(b)}$ (i.e., fewer) components that represents $\mathscr{M}^{(b)}$ well. The likelihood of a random sequence $\mathbf{y} \sim \mathscr{M}^{(b)}$ is given by

$$p(\mathbf{y}|\mathscr{M}^{(b)}) = \sum_{i=1}^{K^{(b)}} \omega_i^{(b)} p(\mathbf{y}|z^{(b)} = i, \mathscr{M}^{(b)}), \tag{4.3}$$

where $z^{(b)} \sim \text{multinomial}(\omega_1^{(b)}, \cdots \omega_{K^{(b)}}^{(b)})$ is the hidden variable that indexes the mixture components. $p(\mathbf{y}|z = i, \mathscr{M}^{(b)})$ is the likelihood of $\mathbf{y}$ under the ith mixture component, as in (4.2), and $\omega_i^{(b)}$ is the mixture weight for the ith component. Likewise, the likelihood of the random sequence $\mathbf{y} \sim \mathscr{M}^{(r)}$ is

$$p(\mathbf{y}|\mathscr{M}^{(r)}) = \sum_{j=1}^{K^{(r)}} \omega_j^{(r)} p(\mathbf{y}|z^{(r)} = j, \mathscr{M}^{(r)}), \tag{4.4}$$

where $z^{(r)} \sim \text{multinomial}(\omega_1^{(r)}, \cdots, \omega_{K^{(r)}}^{(r)})$ is the hidden variable for indexing components in $\mathscr{M}^{(r)}$.

At a high level, the VHEM-H3M algorithm estimates the reduced H3M model $\mathscr{M}^{(r)}$ in (7.7) from *virtual* sequences distributed according to the base H3M model $\mathscr{M}^{(b)}$ in (7.6). From this estimation procedure, the VHEM algorithm provides:

1. a soft clustering of the original $K^{(b)}$ components into $K^{(r)}$ groups, where cluster membership is encoded in assignment variables that represent the *responsibility* of each reduced mixture component for each base mixture component, i.e., $\hat{z}_{i,j} =$

$p(z^{(r)} = j|z^{(b)} = i)$, for $i = 1,\ldots,K^{(b)}$ and $j = 1,\ldots,K^{(r)}$;

2. novel cluster centers represented by the individual mixture components of the reduced model in (7.7), i.e., $p(\mathbf{y}|z^{(r)} = j, \mathscr{M}^{(r)})$ for $j = 1,\ldots,K^{(r)}$.

Finally, because we take the expectation over the virtual samples, the estimation is carried out in an efficient manner that requires only knowledge of the parameters of the base model, without the need of generating actual virtual samples.

Notation

We will always use i and j to index the components of the base model $\mathscr{M}^{(b)}$ and the reduced model $\mathscr{M}^{(r)}$, respectively. To reduce clutter, we will also use the short-hand notation $\mathscr{M}_i^{(b)}$ and $\mathscr{M}_j^{(r)}$ to denote the ith component of $\mathscr{M}^{(b)}$ and the jth component of $\mathscr{M}^{(r)}$, respectively. Hidden states of the HMMs are denoted with β for the base model $\mathscr{M}_i^{(b)}$, and with ρ for the reduced model $\mathscr{M}_j^{(r)}$.

The GMM emission models for each hidden state are denoted as $\mathscr{M}_{i,\beta}^{(b)}$ and $\mathscr{M}_{j,\rho}^{(r)}$. We will always use m and ℓ for indexing the individual Gaussian components of the GMM emissions of the base and reduced models, respectively. The individual Gaussian components are denoted as $\mathscr{M}_{i,\beta,m}^{(b)}$ for the base model, and $\mathscr{M}_{j,\rho,\ell}^{(r)}$ for the reduced model. Finally, we denote the parameters of ith HMM component of the base mixture model as $\mathscr{M}_i^{(b)} = \{\pi^{(b),i}, A^{(b),i}, \{\{c_{\beta,m}^{(b),i}, \mu_{\beta,m}^{(b),i}, \Sigma_{\beta,m}^{(b),i}\}_{m=1}^{M}\}_{\beta=1}^{S}\}$, and for the jth HMM in the reduced mixture as $\mathscr{M}_j^{(r)} = \{\pi^{(r),j}, A^{(r),j}, \{\{c_{\rho,\ell}^{(r),j}, \mu_{\rho,\ell}^{(r),j}, \Sigma_{\rho,\ell}^{(r),j}\}_{\ell=1}^{M}\}_{\rho=1}^{S}\}$.

When appearing in a probability distribution, the short-hand model notation (e.g., $\mathscr{M}_i^{(b)}$) always implies *conditioning* on the model being active. For example, we will use $p(\mathbf{y}|\mathscr{M}_i^{(b)})$ as short-hand for $p(\mathbf{y}|z^{(b)} = i, \mathscr{M}^{(b)})$, or $p(y_t|\mathscr{M}_{i,\beta}^{(b)})$ as short-hand for $p(y_t|x_t = \beta, z^{(b)} = i, \mathscr{M}^{(b)})$. Furthermore, we will use $\pi_\beta^{(b),i}$ as short-hand for the probability of the state sequence β according to the base HMM component $\mathscr{M}_i^{(b)}$, i.e.,

Table 4.1. Notation used in the derivation of the VHEM-H3M algorithm.

variables	*base model (b)*	*reduced model (r)*
index for HMM components	i	j
number of HMM components	$K^{(b)}$	$K^{(r)}$
HMM states	β	ρ
number of HMM states	S	S
HMM state sequence	$\beta = \{\beta_1, \cdots, \beta_\tau\}$	$\rho = \{\rho_1, \cdots, \rho_\tau\}$
index for component of GMM	m	ℓ
number of Gaussian components	M	M
models		
H3M	$\mathscr{M}^{(b)}$	$\mathscr{M}^{(r)}$
HMM component (of H3M)	$\mathscr{M}_i^{(b)}$	$\mathscr{M}_j^{(r)}$
GMM emission	$\mathscr{M}_{i,\beta}^{(b)}$	$\mathscr{M}_{j,\rho}^{(r)}$
Gaussian component (of GMM)	$\mathscr{M}_{i,\beta,m}^{(b)}$	$\mathscr{M}_{j,\rho,\ell}^{(r)}$
parameters		
H3M component weight	$\omega_i^{(b)}$	$\omega_j^{(r)}$
HMM initial state	$\pi^{(b),i}$	$\pi^{(r),j}$
HMM state transition matrix	$A^{(b),i}$	$A^{(r),j}$
GMM emission	$\{c_{\beta,m}^{(b),i}, \mu_{\beta,m}^{(b),i}, \Sigma_{\beta,m}^{(b),i}\}_{m=1}^{M}$	$\{c_{\rho,\ell}^{(r),j}, \mu_{\rho,\ell}^{(r),j}, \Sigma_{\rho,\ell}^{(r),j}\}_{\ell=1}^{M}$
probability distributions	*notation*	*short-hand*
HMM state sequence (b)	$p(\mathbf{x} = \beta \vert z^{(b)} = i, \mathscr{M}^{(b)})$	$p(\beta \vert \mathscr{M}_i^{(b)}) = \pi_\beta^{(b),i}$
HMM state sequence (r)	$p(\mathbf{x} = \rho \vert z^{(r)} = j, \mathscr{M}^{(r)})$	$p(\rho \vert \mathscr{M}_j^{(r)}) = \pi_\rho^{(r),j}$
HMM observation likelihood (r)	$p(\mathbf{y} \vert z^{(r)} = j, \mathscr{M}^{(r)})$	$p(\mathbf{y} \vert \mathscr{M}_j^{(r)})$
GMM emission likelihood (r)	$p(y_t \vert x_t = \rho, \mathscr{M}_j^{(r)})$	$p(y_t \vert \mathscr{M}_{j,\rho}^{(r)})$
Gaussian component likelihood (r)	$p(y_t \vert \zeta_t = \ell, x_t = \rho, \mathscr{M}_j^{(r)})$	$p(y_t \vert \mathscr{M}_{j,\rho,\ell}^{(r)})$
expectations		
HMM observation sequence (b)	$\mathbb{E}_{\mathbf{y} \vert z^{(b)}=i, \mathscr{M}^{(b)}}[\cdot]$	$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\cdot]$
GMM emission (b)	$\mathbb{E}_{y_t \vert x_t=\beta, \mathscr{M}_i^{(b)}}[\cdot]$	$\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\cdot]$
Gaussian component (b)	$\mathbb{E}_{y_t \vert \zeta_t=m, x_t=\beta, \mathscr{M}_i^{(b)}}[\cdot]$	$\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\cdot]$
expected log-likelihood	*lower bound*	*variational distribution*
$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(Y_i \vert \mathscr{M}^{(r)})]$	$\mathscr{L}_{H3M}^{i}$	$q_i(z_i = j) = z_{ij}$
$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y} \vert \mathscr{M}_j^{(r)})]$	$\mathscr{L}_{HMM}^{i,j}$	$q^{i,j}(\rho \vert \beta) = \phi_{\rho \vert \beta}^{i,j}$ $= \phi_1^{i,j}(\rho_1 \vert \beta_1) \prod_{t=2}^{\tau} \phi_t^{i,j}(\rho_t \vert \rho_{t-1}, \beta_t)$
$\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\log p(y \vert \mathscr{M}_{j,\rho}^{(r)})]$	$\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$	$q_{\beta,\rho}^{i,j}(\zeta = \ell \vert m) = \eta_{\ell \vert m}^{(i,\beta),(j,\rho)}$

$p(\beta \vert \mathscr{M}_i^{(b)})$, and likewise $\mathscr{M}_\rho^{(r),j}$ for the reduced HMM component.

Expectations will also use the short-hand model notation to imply conditioning on the model. In addition, expectations are assumed to be taken with respect to the

output variable ($\mathbf{y}$ or y_t), unless otherwise specified. For example, we will use $\mathbb{E}_{\mathscr{M}_i^{(b)}}[\cdot]$ as short-hand for $\mathbb{E}_{\mathbf{y}|z^{(b)}=i,\mathscr{M}^{(b)}}[\cdot]$.

Table 4.1 summarizes the notation used in the derivation, including the variable names, model parameters, and short-hand notations for probability distributions and expectations. The bottom of Table 4.1 also summarizes the variational lower bound and variational distributions, which will be introduced subsequently.

4.3.2 Variational HEM Algorithm

To learn the reduced model in (7.7), we consider a set of N *virtual* samples, distributed according to the base model $\mathscr{M}^{(b)}$ in (7.6), such that $N_i = N\omega_i^{(b)}$ samples are drawn from the ith component. We denote the set of N_i virtual samples for the ith component as $Y_i = \{\mathbf{y}^{(i,m)}\}_{m=1}^{N_i}$, where $\mathbf{y}^{(i,m)} \sim \mathscr{M}_i^{(b)}$, and the entire set of N samples as $Y = \{Y_i\}_{i=1}^{K^{(b)}}$. Note that, in this formulation, we are not considering virtual samples $\{\mathbf{x}^{(i,m)}, \mathbf{y}^{(i,m)}\}$ for each base component, according to its joint distribution $p(\mathbf{x},\mathbf{y}|\mathscr{M}_i^{(b)})$. The reason is that the hidden-state space of each base mixture component $\mathscr{M}_i^{(b)}$ may have a different representation (e.g.the numbering of the hidden states may be permuted between the components). This mismatch will cause problems when the parameters of $\mathscr{M}_j^{(r)}$ are computed from virtual samples of the hidden states of $\{\mathscr{M}_i^{(b)}\}_{i=1}^{K^{(b)}}$. Instead, we treat $X_i = \{\mathbf{x}^{(i,m)}\}_{m=1}^{N_i}$ as "missing" information, and estimate them in the E-step. The log-likelihood of the virtual samples is

$$\log p(Y|\mathscr{M}^{(r)}) = \sum_{i=1}^{K^{(b)}} \log p(Y_i|\mathscr{M}^{(r)}), \tag{4.5}$$

where, in order to obtain a consistent clustering, we assume the entirety of samples Y_i is assigned to the same component of the reduced model [164].

The original formulation of HEM [164] maximizes (4.5) with respect to $\mathscr{M}^{(r)}$,

and uses the law of large numbers to turn the virtual samples Y_i into an expectation over the base model components $\mathscr{M}_i^{(b)}$. In this paper, we will start with a different objective function to derive the VHEM algorithm. To estimate $\mathscr{M}^{(r)}$, we will maximize the average log-likelihood of all possible virtual samples, weighted by their likelihood of being generated by $\mathscr{M}_i^{(b)}$, i.e., the *expected* log-likelihood of the virtual samples,

$$\mathscr{J}(\mathscr{M}^{(r)}) = \mathbb{E}_{\mathscr{M}^{(b)}}\left[\log p(Y|\mathscr{M}^{(r)})\right] = \sum_{i=1}^{K^{(b)}} \mathbb{E}_{\mathscr{M}_i^{(b)}}\left[\log p(Y_i|\mathscr{M}^{(r)})\right], \qquad (4.6)$$

where the expectation is over the base model components $\mathscr{M}_i^{(b)}$. Maximizing (4.6) will eventually lead to the same estimate as maximizing (4.5), but allows us to strictly preserve the variational lower bound, which would otherwise be ruined when applying the law of large numbers to (4.5).

A general approach to deal with maximum likelihood estimation in the presence of hidden variables (which is the case for H3Ms) is the EM algorithm [49]. In the traditional formulation the EM algorithm is presented as an alternation between an expectation step (E-step) and a maximization step (M-step). In this work, we take a variational perspective [116, 166, 48], which views each step as a maximization step. The variational E-step first obtains a family of lower bounds to the (expected) log-likelihood (i.e., to equation 4.6), indexed by variational parameters, and then optimizes over the variational parameters to find the tightest bound. The corresponding M-step then maximizes the lower bound (with the variational parameters fixed) with respect to the model parameters. One advantage of the variational formulation is that it readily allows for useful extensions to the EM algorithm, such as replacing a difficult inference in the E-step with a variational approximation. In practice, this is achieved by restricting the maximization in the variational E-step to a smaller domain for which the lower bound is tractable.

The EM algorithm with variational E-step is guaranteed to converge [71]. Despite the approximation prevents convergence to local maxima of the data log-likelihood [71], the algorithm still performs well empirically, as shown in Section 4.5 and Section 4.6.

Lower Bound to an Expected Log-likelihood

Before proceeding with the derivation of VHEM for H3Ms, we first need to derive a lower-bound to an expected log-likelihood term, e.g., (4.6). Our derivation starts from a variational lower-bound to a log-likelihood (as opposed to an *expected* log-likelihood), a standard tool in machine learning [88, 84], which we briefly review next. In all generality, let $\{O,H\}$ be the observation and hidden variables of a probabilistic model, respectively, where $p(H)$ is the distribution of the hidden variables, $p(O|H)$ is the conditional likelihood of the observations, and $p(O) = \sum_H p(O|H)p(H)$ is the observation likelihood. We can define a *variational lower bound* to the observation log-likelihood [88, 84]:

$$\begin{aligned}\log p(O) &\geq \log p(O) - D(q(H)||p(H|O)) \\ &= \sum_H q(H) \log \frac{p(H)p(O|H)}{q(H)},\end{aligned}$$

where $p(H|O)$ is the posterior distribution of H given observation O, and $D(p\|q) = \int p(y) \log \frac{p(y)}{q(y)} dy$ is the Kullback-Leibler (KL) divergence between two distributions, p and q. We introduce a variational distribution $q(H)$, which approximates the posterior distribution, where $\sum_H q(H) = 1$ and $q(H) \geq 0$. When the variational distribution equals the true posterior, $q(H) = P(H|O)$, then the KL divergence is zero, and hence the lower-bound reaches $\log p(O)$. When the true posterior cannot be computed, then typically q is restricted to some set of approximate posterior distributions $\mathcal{Q}$ that are tractable, and the

best lower-bound is obtained by maximizing over $q \in \mathscr{Q}$,

$$\log p(O) \geq \max_{q \in \mathscr{Q}} \sum_{H} q(H) \log \frac{p(H)p(O|H)}{q(H)}. \tag{4.7}$$

From the standard lower bound in (4.7), we can now derive a lower bound to an expected log-likelihood expression. Let $\mathbb{E}_b[\cdot]$ be the expectation with respect to O with some distribution $p_b(O)$. Since $p_b(O)$ is non-negative, taking the expectation on both sides of (4.7) yields,

$$\mathbb{E}_b\left[\log p(O)\right] \geq \mathbb{E}_b\left[\max_{q \in Q} \sum_{H} q(H) \log \frac{p(H)p(O|H)}{q(H)}\right] \tag{4.8}$$

$$\geq \max_{q \in Q} \mathbb{E}_b\left[\sum_{H} q(H) \log \frac{p(H)p(O|H)}{q(H)}\right] \tag{4.9}$$

$$= \max_{q \in Q} \sum_{H} q(H) \left\{\log \frac{p(H)}{q(H)} + \mathbb{E}_b\left[\log p(O|H)\right]\right\}, \tag{4.10}$$

where (4.9) follows from Jensen's inequality (i.e., $f(\mathbb{E}[x]) \leq \mathbb{E}[f(x)]$ when f is convex), and the convexity of the max function. Hence, (4.10) is a variational lower bound on the expected log-likelihood, which depends on the family of variational distributions Q.

In (4.8) we are computing the best lower-bound (4.7) to $\log p(O)$ *individually* for each value of the observation variable O, which in general corresponds to different optimal $q^* \in \mathscr{Q}$ for different values of O. Note that the expectation in (4.8) is not analytically tractable when $p(O)$ is a mixture model (i.e, it is the expected log-likelihood of a mixture). Hence, we treat the ensemble of observations $O \sim p_b$ as a whole, and in (4.9) find a single $q^* \in \mathscr{Q}$ for which the lower bound is best on average. Mathematically, this correspond to using Jensen inequality to pass from (4.8) to (4.9), which shows that the additional approximation makes the lower-bound looser.

Variational Lower Bound

We now derive a lower bound to the expected log-likelihood cost function in (4.6). The derivation will proceed by successively applying the lower bound from (4.10) to each expected log-likelihood term that arises. This will result in a set of nested lower bounds.

A variational lower bound to the expected log-likelihood of the virtual samples in (4.6) is obtained by lower bounding each of the expectation terms $\mathbb{E}_{\mathscr{M}_i^{(b)}}$ in the sum,

$$\mathscr{J}\left(\mathscr{M}^{(r)}\right) = \sum_{i=1}^{K^{(b)}} \mathbb{E}_{\mathscr{M}_i^{(b)}}\left[\log p(Y_i|\mathscr{M}^{(r)})\right] \geq \sum_{i=1}^{K^{(b)}} \mathscr{L}_{H3M}^{i}, \tag{4.11}$$

where we define three nested lower bounds, corresponding to different model elements (the H3M, the component HMMs, and the emission GMMs):

$$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(Y_i|\mathscr{M}^{(r)})] \geq \mathscr{L}_{H3M}^{i}, \tag{4.12}$$

$$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})] \geq \mathscr{L}_{HMM}^{i,j}, \tag{4.13}$$

$$\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho}^{(r)})] \geq \mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}. \tag{4.14}$$

In (4.12), the first lower bound, $\mathscr{L}_{H3M}^{i}$, is on the expected log-likelihood of an H3M $\mathscr{M}^{(r)}$ with respect to an HMM $\mathscr{M}_i^{(b)}$. Because $p(Y_i|\mathscr{M}^{(r)})$ is the likelihood under a mixture of HMMs, as in (7.7), where the observation variable is Y_i and the hidden variable is z_i (the assignment of Y_i to a component of $\mathscr{M}^{(r)}$), its expectation cannot be calculated directly. Hence, we introduce the variational distribution $q_i(z_i)$ and apply (4.10) to (4.12), yielding the lower bound (see Appendix A.1),

$$\mathscr{L}_{H3M}^{i} = \max_{q_i} \sum_j q_i(z_i = j)\left\{\log \frac{p(z_i = j|\mathscr{M}^{(r)})}{q_i(z_i = j)} + N_i \mathscr{L}_{HMM}^{i,j}\right\}. \tag{4.15}$$

The lower bound in (4.15) depends on the second lower bound (Eq. 4.13), $\mathscr{L}_{HMM}^{i,j}$, which is on the expected log-likelihood of an HMM $\mathscr{M}_j^{(r)}$, averaged over observation sequences from a *different* HMM $\mathscr{M}_i^{(b)}$. Although the data log-likelihood $\log p(\mathbf{y}|\mathscr{M}_j^{(r)})$ can be computed exactly using the forward algorithm [135], calculating its expectation is not analytically tractable since an observation sequence $\mathbf{y}$ from a HMM $\mathscr{M}_j^{(r)}$ is essentially an observation from a mixture model[1].

To calculate the lower bound $\mathscr{L}_{HMM}^{i,j}$ in (4.13), we first rewrite the expectation $\mathbb{E}_{\mathscr{M}_i^{(b)}}$ in (4.13) to explicitly marginalize over the state sequence β of $\mathscr{M}_i^{(b)}$, and then apply (4.10) where the hidden variable is the state sequence ρ of $\mathscr{M}_j^{(r)}$, yielding (see Appendix A.1)

$$\mathscr{L}_{HMM}^{i,j} = \sum_{\beta} \pi_{\beta}^{(b),i} \max_{q^{i,j}} \sum_{\rho} q^{i,j}(\rho|\beta) \left\{ \log \frac{p(\rho|\mathscr{M}_j^{(r)})}{q^{i,j}(\rho|\beta)} + \sum_t \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \right\}, \qquad (4.16)$$

where we introduce a variational distribution $q^{i,j}(\rho|\beta)$ on the state sequence ρ, which depends on a particular sequence β from $\mathscr{M}_i^{(b)}$. As before, (4.16) depends on another nested lower bound, $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$ in (4.14), which is on the expected log-likelihood of a GMM emission density $\mathscr{M}_{j,\rho}^{(r)}$ with respect to another GMM $\mathscr{M}_{i,\beta}^{(b)}$. This lower bound does not depend on time, as we have assumed that the emission densities are time-invariant.

Finally, we obtain the lower bound $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$ for (4.14), by explicitly marginalizing over the GMM hidden assignment variable in $\mathscr{M}_{i,\beta}^{(b)}$ and then applying (4.10) to the

[1] For an observation sequence of length τ, an HMM with S states can be considered as a mixture model with $O(S^{\tau})$ components.

expectation of the GMM emission distribution $p(y|\mathscr{M}^{(r)}_{j,\rho})$, yielding (see Appendix A.1),

$$\mathscr{L}^{(i,\beta),(j,\rho)}_{GMM} = \sum_{m=1}^{M} c^{(b),i}_{\beta,m} \max_{q^{i,j}_{\beta,\rho}} \sum_{\zeta=1}^{M} q^{i,j}_{\beta,\rho}(\zeta|m) \left\{ \log \frac{p(\zeta|\mathscr{M}^{(r)}_{j,\rho})}{q^{i,j}_{\beta,\rho}(\zeta|m)} + \mathbb{E}_{\mathscr{M}^{(b)}_{i,\beta,m}} [\log p(y|\mathscr{M}^{(r)}_{j,\rho,\zeta})] \right\}, \tag{4.17}$$

where we introduce the variational distribution $q^{i,j}_{\beta,\rho}(\zeta|m)$, which is conditioned on the observation y arising from the mth component in $\mathscr{M}^{(b)}_{i,\beta}$. In (4.17), the term

$$\mathbb{E}_{\mathscr{M}^{(b)}_{i,\beta,m}} [\log p(y|\mathscr{M}^{(r)}_{j,\rho,\ell})]$$

is the expected log-likelihood of the Gaussian distribution $\mathscr{M}^{(r)}_{j,\rho,\ell}$ with respect to the Gaussian $\mathscr{M}^{(b)}_{i,\beta,m}$, which has a closed-form solution (see Section 4.3.3).

In summary, we have derived a variational lower bound to the expected log-likelihood of the virtual samples, which is given by (4.11). This lower bound is composed of three nested lower bounds in (4.15), (4.16), and (4.17), corresponding to different model elements (the H3M, the component HMMs, and the emission GMMs), where $q_i(z_i)$, $q^{i,j}(\rho|\beta)$, and $q^{i,j}_{\beta,\rho}(\zeta|m)$ are the corresponding variational distributions. Finally, the variational HEM algorithm for HMMs consists of two alternating steps:

- (variational E-step) given $\mathscr{M}^{(r)}$, calculate the variational distributions $q^{i,j}_{\beta,\rho}(\zeta|m)$, $q^{i,j}(\rho|\beta)$, and $q_i(z_i)$ for the lower bounds in (4.17), (4.16), and (4.15);
- (M-step) update the model parameters via $\mathscr{M}^{(r)^*} = \text{argmax}_{\mathscr{M}^{(r)}} \sum_{i=1}^{K^{(b)}} \mathscr{L}^{i}_{H3M}$.

In the following subsections, we derive the E- and M-steps of the algorithm. The entire procedure is summarized in Algorithm 4.

4.3.3 Variational E-Step

The variational E-step consists of finding the variational distributions that maximize the lower bounds in (4.17), (4.16), and (4.15). In particular, given the nesting of the lower bounds, we proceed by first maximizing the GMM lower bound $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$ for each pair of emission GMMs in the base and reduced models. Next, the HMM lower bound $\mathscr{L}_{HMM}^{i,j}$ is maximized for each pair of HMMs in the base and reduced models, followed by maximizing the H3M lower bound $\mathscr{L}_{H3M}^{i}$ for each base HMM. Finally, a set of summary statistics are calculated, which will be used in the M-step.

Variational Distributions

We first consider the forms of the three variational distributions, as well as the optimal parameters to maximize the corresponding lower bounds.

GMM: For the GMM lower bound $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$, we assume each variational distribution has the form [78]

$$q_{\beta,\rho}^{i,j}(\zeta = l|m) = \eta_{\ell|m}^{(i,\beta),(j,\rho)},$$

where $\sum_{\ell=1}^{M} \eta_{\ell|m}^{(i,\beta),(j,\rho)} = 1$, and $\eta_{\ell|m}^{(i,\beta),(j,\rho)} \geq 0, \forall \ell$. Intuitively, $\eta^{(i,\beta),(j,\rho)}$ is the responsibility matrix between each pair of Gaussian components in the GMMs $\mathscr{M}_{i,\beta}^{(b)}$ and $\mathscr{M}_{j,\rho}^{(r)}$, where $\eta_{\ell|m}^{(i,\beta),(j,\rho)}$ represents the probability that an observation from component m of $\mathscr{M}_{i,\beta}^{(b)}$ corresponds to component ℓ of $\mathscr{M}_{j,\rho}^{(r)}$. Substituting into (4.17) and maximizing the variational parameters yields (see Appendix A.2)

$$\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)} = \frac{c_{\rho,\ell}^{(r),j} \exp\left\{\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})]\right\}}{\sum_{\ell'} c_{\rho,\ell'}^{(r),j} \exp\left\{\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell'}^{(r)})]\right\}}, \tag{4.18}$$

where the expected log-likelihood of a Gaussian $\mathscr{M}_{j,\rho,\ell}^{(r)}$ with respect to another Gaussian $\mathscr{M}_{i,\beta,m}^{(b)}$ is computable in closed-form [129],

$$\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})] = -\frac{d}{2}\log 2\pi - \frac{1}{2}\log\left|\Sigma_{\rho,\ell}^{(r),j}\right| - \frac{1}{2}\mathrm{tr}\left((\Sigma_{\rho,\ell}^{(r),j})^{-1}\Sigma_{\beta,m}^{(b),i}\right) - \frac{1}{2}(\mu_{\rho,\ell}^{(r),j} - \mu_{\beta,m}^{(b),i})^T(\Sigma_{\rho,\ell}^{(r),j})^{-1}(\mu_{\rho,\ell}^{(r),j} - \mu_{\beta,m}^{(b),i}).$$

HMM: For the HMM lower bound $\mathscr{L}_{HMM}^{i,j}$, we assume each variational distribution takes the form of a Markov chain,

$$q^{i,j}(\rho|\beta) = \phi^{i,j}(\rho|\beta) = \phi_1^{i,j}(\rho_1|\beta_1)\prod_{t=2}^{\tau}\phi_t^{i,j}(\rho_t|\rho_{t-1},\beta_t),$$

where $\sum_{\rho_1=1}^{S}\phi_1^{i,j}(\rho_1|\beta_1) = 1$, and $\sum_{\rho_t=1}^{S}\phi_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) = 1$, and all the factors are non-negative. The variational distribution $q^{i,j}(\rho|\beta)$ represents the probability of the state sequence ρ in HMM $\mathscr{M}_j^{(r)}$, when $\mathscr{M}_j^{(r)}$ is used to explain the *observation* sequence generated by $\mathscr{M}_i^{(b)}$ that evolved through state sequence β.

Substituting $\phi^{i,j}$ into (4.16), the maximization with respect to $\phi_t^{i,j}(\rho_t|\rho_{t-1},\beta_t)$ and $\phi_1^{i,j}(\rho_1|\beta_1)$ is carried out independently for each pair (i,j), and follows [78]. This is further detailed in Appendix A.2. By separating terms and breaking up the summation over β and ρ, the optimal $\hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t)$ and $\hat{\phi}_1^{i,j}(\rho_1|\beta_1)$ can be obtained using an efficient recursive iteration (similar to the forward algorithm).

H3M: For the H3M lower bound $\mathscr{L}_{H3M}^{i}$, we assume variational distributions of the form $q_i(z_i = j) = z_{ij}$, where $\sum_{j=1}^{K^{(r)}} z_{ij} = 1$, and $z_{ij} \geq 0$. Substituting z_{ij} into (4.15), and maximizing variational parameters are obtained as (see Appendix A.2)

$$\hat{z}_{ij} = \frac{\omega_j^{(r)}\exp(N_i\mathscr{L}_{HMM}^{i,j})}{\sum_{j'}\omega_{j'}^{(r)}\exp(N_i\mathscr{L}_{HMM}^{i,j'})}. \tag{4.19}$$

Note that in the standard HEM algorithm [164, 33], the assignment probabilities z_{ij} are based on the expected log-likelihoods of the components, (e.g., $\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]$ for H3Ms). For the variational HEM algorithm, these expectations are now replaced with their lower bounds (in our case, $\mathscr{L}_{HMM}^{i,j}$).

Lower Bound

Substituting the optimal variational distributions into (4.15), (4.16), and (4.17) gives the lower bounds,

$$\mathscr{L}_{H3M}^{i} = \sum_j \hat{z}_{ij} \left\{ \log \frac{\omega_j^{(r)}}{\hat{z}_{ij}} + N_i \mathscr{L}_{HMM}^{i,j} \right\}, \tag{4.20}$$

$$\mathscr{L}_{HMM}^{i,j} = \sum_\beta \pi_\beta^{(b),i} \sum_\rho \hat{\phi}^{i,j}(\rho|\beta) \left\{ \log \frac{\pi_\rho^{(r),j}}{\hat{\phi}^{i,j}(\rho|\beta)} + \sum_t \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \right\}, \tag{4.21}$$

$$\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)} = \sum_{m=1}^{M} c_{\beta,m}^{(b),i} \sum_{\ell=1}^{M} \hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)} \left\{ \log \frac{c_{\rho,\ell}^{(r),j}}{\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}} + \mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})] \right\}. \tag{4.22}$$

The lower bound $\mathscr{L}_{HMM}^{i,j}$ requires summing over all sequences β and ρ. This summation can be computed efficiently along with $\hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t)$ and $\hat{\phi}_1^{i,j}(\rho_1|\beta_1)$ using a recursive algorithm from [78]. This is described in Appendix A.2.

Summary Statistics

After calculating the optimal variational distributions, we calculate the following summary statistics, which are necessary for the M-step:

$$\nu_1^{i,j}(\rho_1,\beta_1) = \pi_{\beta_1}^{(b),i}\,\hat{\phi}_1^{i,j}(\rho_1|\beta_1),$$
$$\xi_t^{i,j}(\rho_{t-1},\rho_t,\beta_t) = \left(\sum_{\beta_{t-1}=1}^{S} \nu_{t-1}^{i,j}(\rho_{t-1},\beta_{t-1})\,a_{\beta_{t-1},\beta_t}^{(b),i}\right)\hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t), \text{ for } t=2,\ldots,\tau,$$
$$\nu_t^{i,j}(\rho_t,\beta_t) = \sum_{\rho_{t-1}=1}^{S} \xi_t^{i,j}(\rho_{t-1},\rho_t,\beta_t), \text{ for } t=2,\ldots,\tau,$$

and the aggregate statistics

$$\hat{\nu}_1^{i,j}(\rho) = \sum_{\beta=1}^{S} \nu_1^{i,j}(\rho,\beta), \tag{4.23}$$
$$\hat{\nu}^{i,j}(\rho,\beta) = \sum_{t=1}^{\tau} \nu_t^{i,j}(\rho,\beta), \tag{4.24}$$
$$\hat{\xi}^{i,j}(\rho,\rho') = \sum_{t=2}^{\tau}\sum_{\beta=1}^{S} \xi_t^{i,j}(\rho,\rho',\beta). \tag{4.25}$$

The statistic $\hat{\nu}_1^{i,j}(\rho)$ is the expected number of times that the HMM $\mathscr{M}_j^{(r)}$ starts from state ρ, when modeling sequences generated by $\mathscr{M}_i^{(b)}$. The quantity $\hat{\nu}^{i,j}(\rho,\beta)$ is the expected number of times that the HMM $\mathscr{M}_j^{(r)}$ is in state ρ when the HMM $\mathscr{M}_i^{(b)}$ is in state β, when both HMMs are modeling sequences generated by $\mathscr{M}_i^{(b)}$. Similarly, the quantity $\hat{\xi}^{i,j}(\rho,\rho')$ is the expected number of transitions from state ρ to state ρ' of the HMM $\mathscr{M}_j^{(r)}$, when modeling sequences generated by $\mathscr{M}_i^{(b)}$.

4.3.4 M-Step

In the M-step, the lower bound in (4.11) is maximized with respect to the parameters $\mathscr{M}^{(r)}$,

$$\mathscr{M}^{(r)^*} = \underset{\mathscr{M}^{(r)}}{\operatorname{argmax}} \sum_{i=1}^{K^{(b)}} \mathscr{L}^i_{H3M}.$$

The derivation of the maximization is presented in Appendix A.3. Each mixture component of $\mathscr{M}^{(r)}$ is updated independently according to

$$\omega_j^{(r)^*} = \frac{\sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}}{K^{(b)}}, \tag{4.26}$$

$$\pi_\rho^{(r),j^*} = \frac{\sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}\omega_i^{(b)}\hat{\nu}_1^{i,j}(\rho)}{\sum_{\rho'=1}^{S}\sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}\omega_i^{(b)}\hat{\nu}_1^{i,j}(\rho'))}, \quad a_{\rho,\rho'}^{(r),j^*} = \frac{\sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}\omega_i^{(b)}\hat{\xi}^{i,j}(\rho,\rho')}{\sum_{\sigma=1}^{S}\sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}\omega_i^{(b)}\hat{\xi}^{i,j}(\rho,\sigma)}, \tag{4.27}$$

$$c_{\rho,\ell}^{(r),j^*} = \frac{\Omega_{j,\rho}\left(\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}\right)}{\sum_{\ell'=1}^{M}\Omega_{j,\rho}\left(\hat{\eta}_{\ell'|m}^{(i,\beta),(j,\rho)}\right)}, \qquad \mu_{\rho,\ell}^{(r),j^*} = \frac{\Omega_{j,\rho}\left(\eta_{\ell|m}^{(i,\beta),(j,\rho)}\ \mu_{\beta,m}^{(b),i}\right)}{\Omega_{j,\rho}\left(\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}\right)}, \tag{4.28}$$

$$\Sigma_{\rho,\ell}^{(r),j^*} = \frac{\Omega_{j,\rho}\left(\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}\left[\Sigma_{\beta,m}^{(b),i} + (\mu_{\beta,m}^{(b),i} - \mu_{\rho,\ell}^{(r),j})(\mu_{\beta,m}^{(b),i} - \mu_{\rho,\ell}^{(r),j})^T\right]\right)}{\Omega_{j,\rho}\left(\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}\right)}, \tag{4.29}$$

where $\Omega_{j,\rho}(\cdot)$ is the weighted sum operator over all base models, HMM states, and GMM components (i.e., over all tuples (i,β,m)),

$$\Omega_{j,\rho}(f(i,\beta,m)) = \sum_{i=1}^{K^{(b)}} \hat{z}_{i,j}\omega_i^{(b)} \sum_{\beta=1}^{S} \hat{\nu}^{i,j}(\rho,\beta) \sum_{m=1}^{M} c_{\beta,m}^{(b),i} f(i,\beta,m). \tag{4.30}$$

The terms $\pi_\rho^{(r),j}$ and $a_{\rho,\rho'}^{(r),j}$ are elements of the initial state prior and transition matrix, $\pi^{(r),j}$ and $A^{(r),j}$. Note that the covariance matrices of the reduced models in (4.29) include an additional outer-product term, which acts to regularize the covariances of the base

models. This regularization effect derives from the E-step, which averages all possible observations from the base model.

4.4 Applications and Related Work

In the previous section, we derived the VHEM-H3M algorithm to cluster HMMs. We now discuss various applications of the algorithm (Section 4.4.1), and then present some literature that is related to HMM clustering (Section 6.2).

4.4.1 Applications of the VHEM-H3M Algorithm

The proposed VHEM-H3M algorithm clusters HMMs *directly* through the distributions they represent, and learns *novel* HMM cluster centers that compactly represent the structure of each cluster.

An application of the VHEM-H3M algorithm is in *hierarchical clustering* of HMMs. In particular, the VHEM-H3M algorithm is used recursively on the HMM cluster centers, to produce a bottom-up hierarchy of the input HMMs. Since the cluster centers condense the structure of the clusters they represent, the VHEM-H3M algorithm can implicitly leverage rich information on the underlying structure of the clusters, which is expected to impact positively the quality of the resulting hierarchical clustering.

Another application of VHEM is for efficient estimation of H3Ms from data, by using a *hierarchical estimation procedure* to break the learning problem into smaller pieces. First, a data set is split into small (non-overlapping) portions and intermediate HMMs are learned for each portion, via standard EM. Then, the final model is estimated from the intermediate models using the VHEM-H3M algorithm. Because VHEM and standard EM are based on similar maximum-likelihood principles, it drives model estimation towards similar optimal parameter values as performing EM estimation directly on the full dataset. However, compared to direct EM estimation, VHEM-H3M is more

memory- and time-efficient. First, it no longer requires storing in memory the entire data set during parameter estimation. Second, it does not need to evaluate the likelihood of all the samples at each iteration, and converges to effective estimates in shorter times. Note that even if a parallel implementation of EM could effectively handle the high memory requirements, a parallel-VHEM will still require fewer resources than a parallel-EM.

In addition, for the hierarchical procedure, the estimation of the intermediate models can be easily parallelized, since they are learned independently of each other. Finally, hierarchical estimation allows for efficient model updating when adding new data. Assuming that the previous intermediate models have been saved, re-estimating the H3M requires learning the intermediate models of only the new data, followed by running VHEM again. Since estimation of the intermediate models is typically as computationally intensive as the VHEM stage, reusing the previous intermediate models will lead to considerable computational savings when re-estimating the H3M.

In hierarchical estimation (EM on each time-series, VHEM on intermediate models), VHEM implicitly averages over all possible observations (virtual variations of each time-series) compatible with the intermediate models. We expect this to regularize estimation, which may result in models that generalize better (compared to estimating models with direct EM). Lastly, the "virtual" samples (i.e., sequences), which VHEM implicitly generates for maximum-likelihood estimation, need not be of the same length as the actual input data for estimating the intermediate models. Making the virtual sequences relatively short will positively impact the run time of each VHEM iteration. This may be achieved without loss of modeling accuracy, as show in Section 4.6.3.

4.4.2 Related Work

[86]'s approach to clustering HMMs consists of applying spectral clustering to a probability product kernel (PPK) matrix between HMMs—we will refer to it as PPK-SC.

In particular, the PPK similarity between two HMMs, $\mathscr{M}^{(a)}$ and $\mathscr{M}^{(b)}$, is defined as

$$k(a,b) = \int p(\mathbf{y}|\mathscr{M}^{(a)})^{\lambda}\, p(\mathbf{y}|\mathscr{M}^{(b)})^{\lambda} d\mathbf{y}, \tag{4.31}$$

where λ is a scalar, and τ is the length of "virtual" sequences. The case $\lambda = \frac{1}{2}$ corresponds to the Bhattacharyya affinity. While this approach indirectly leverages the probability distributions represented by the HMMs (i.e., the PPK affinity is computed from the probability distributions of the HMMs) and has proven successful in grouping HMMs into similar clusters [86], it has several limitations. First, the spectral clustering algorithm cannot produce novel HMM cluster centers to represent the clusters, which is suboptimal for several applications of HMM clustering. For example, when implementing hierarchical clustering in the spectral embedding space (e.g., using hierarchical k-means clustering), clusters are represented by *single* points in the embedding space. This may fail to capture information on the local structure of the clusters that, when using VHEM-H3M, would be encoded by the novel HMM cluster centers. Hence, we expect VHEM-H3M to produce better hierarchical clustering than the spectral clustering algorithm, especially at higher levels of the hierarchy. This is because, when building a new level, VHEM can leverage more information from the lower levels, as encoded in the HMM cluster centers.

One simple extension of PPK-SC to obtain a HMM cluster center is to select the input HMM that the spectral clustering algorithm maps closest to the spectral clustering center. However, with this method, the HMM cluster centers are limited to be one of the *existing* input HMMs (i.e., similar to the k-medoids algorithm by [90]), instead of the HMMs that optimally condense the structure of the clusters. Therefore, we expect the *novel* HMM cluster centers learned by VHEM-H3M to better represent the clusters. A more involved, "hybrid" solution is to learn the HMM cluster centers with VHEM-H3M

after obtaining clusters with PPK-SC—using the VHEM-H3M algorithm to summarize all the HMMs within each PPK-SC cluster into a single HMM. However, we expect our VHEM-H3M algorithm to learn more accurate clustering models, since it jointly learns the clustering and the HMM centers, by optimizing a single objective function (i.e., the lower bound to the expected log-likelihood in equation 4.11).

A second drawback of the spectral clustering algorithm is that the construction and the inversion of the similarity matrix between the input HMMs is a costly operation when their number is large[2] (e.g., see the experiment on H3M density estimation on the music data in Section 4.6.1). Therefore, we expect VHEM-H3M to be computationally more efficient than the spectral clustering algorithm since, by *directly* operating on the probability distributions of the HMMs, it does not require the construction of an initial embedding or any costly matrix operation on large kernel matrices.

Finally, as [85] note, the exact computation of (4.31) cannot be carried out efficiently, unless $\lambda = 1$. For different values of λ,[3] [85] propose to *approximate* $k(a,b)$ with an alternative kernel function that can be efficiently computed; this alternative kernel function, however, is not guaranteed to be invariant to different but equivalent representations of the hidden state process [85].[4] Alternative approximations for the Bhattacharyya setting (i.e., $\lambda = \frac{1}{2}$) have been proposed by [77].

Note that spectral clustering algorithms (similar to the one by [86]) can be applied to kernel (similarity) matrices that are based on other affinity scores between

[2] The computational complexity of large spectral clustering problems can be alleviated by means of numerical techniques for the solutions of eigenfunction problems such as the Nyström method [123, 62], or by sampling only part of the similarity matrix and using a sparse eigensonver [3].

[3] The experimental results in [85] and [86] suggest to use $\lambda < 1$.

[4] The kernel in (4.31) is computed by marginalizing out the hidden state variables, i.e., $\int \left(\sum_{\mathbf{x}} p(\mathbf{y},\mathbf{x}|\mathscr{M}^{(a)})\right)^{\lambda} \left(\sum_{\mathbf{x}} p(\mathbf{y},\mathbf{x}|\mathscr{M}^{(b)})\right)^{\lambda} d\mathbf{y}$. This can be efficiently solved with the junction tree algorithm only when $\lambda = 1$. For $\lambda \neq 1$, [85] propose to use an alternative kernel $\tilde{k}$ that applies the power operation to the terms of the sum rather than the entire sum, where the terms are joint probabilities $p(\mathbf{y},\mathbf{x})$. I.e., $\tilde{k}(a,b) = \int \sum_{\mathbf{x}} \left(p(\mathbf{y},\mathbf{x}|\mathscr{M}^{(a)})\right)^{\lambda} \sum_{\mathbf{x}} \left(p(\mathbf{y},\mathbf{x}|\mathscr{M}^{(b)})\right)^{\lambda} d\mathbf{y}$.

HMM distributions than the PPK similarity of [85]. Examples can be found in earlier work on HMM-based clustering of time-series, such as by [89], [104], [11], [128]. In particular, [89] propose to approximate the (symmetrised) log-likelihood between two HMM distributions by computing the log-likelihood of real samples generated by one model under the other.[5] Extensions of the work of [89] have been proposed by [178] and [175]. In this work we do not pursue a comparison of the various similarity functions, but implement spectral clustering only based on PPK similarity (which [86] showed to be superior).

HMMs can also be clustered by sampling a number of time-series from each of the HMMs in the base mixture, and then applying the EM algorithm for H3Ms [151], to cluster the time-series. Despite its simplicity, this approach would suffer from high memory and time requirements, especially when dealing with a large number of input HMMs. First, all generated samples need to be stored in memory. Second, evaluating the likelihood of the generated samples at each iteration is computationally intensive, and prevents the EM algorithm from converging to effective estimates in acceptable times.[6] On the contrary, VHEM-H3M is more efficient in computation and memory usage, as it replaces a costly sampling step (along with the associated likelihood computations at each iteration) with an expectation. An additional problem of EM with sampling is that, with a simple application of the EM algorithm, time-series generated from the same input HMM can be assigned to different clusters of the output H3M. As a consequence, the resulting clustering is not necessary *consistent*, since in this case the corresponding input HMM may not be clearly assigned to any single cluster. In our experiments, we circumvent this problem by defining appropriate constrains on the assignment variables.

[5] For two HMM distributions, $\mathscr{M}^{(a)}$ and $\mathscr{M}^{(b)}$, [89] consider the affinity $L(a,b) = \frac{1}{2}\left[\log p(Y_b|\mathscr{M}^{(a)}) + p(Y_a|\mathscr{M}^{(b)})\right]$, where Y_a and Y_b are sets of observation sequences generated from $\mathscr{M}^{(a)}$ and $\mathscr{M}^{(b)}$, respectively.

[6] In our experiments, EM on generated samples took two orders of magnitude more time than VHEM.

The VHEM algorithm is similar in spirit to Bregman-clustering by [12]. Both algorithms base clustering on KL-divergence—the KL divergence and the expected log-likelihood differ only for an entropy term that does not affect the clustering.[7] The main differences are: 1) in our setting, the expected log-likelihood (and KL divergence) is not computable in closed form, and hence VHEM uses an approximation; 2) VHEM-H3M clusters random *processes* (i.e., time series models), whereas Bregman-clustering[12] is limited to single random variables. Note that the number of virtual observations N allows to control the *peakiness* of the assignments $\hat{z}_{i,j}$. Limiting cases for $N \to \infty$ and $N = 1$ are similar to Bregman *hard* and *soft* cluttering, respectively [68, 12, 51].

In the next two sections, we validate the points raised in this discussion through experimental evaluation using the VHEM-H3M algorithm. In particular, we consider clustering experiments in Section 4.5, and H3M density estimation for automatic annotation and retrieval in Section 4.6. Each application exploits some of the benefits of VHEM. First, we show that VHEM-H3M is more accurate in clustering than PPK-SC, in particular at higher levels of a hierarchical clustering (Section 4.5.2), and in an experiment with synthetic data (Section 4.5.3). Similarly, the annotation and retrieval results in Section 4.6 favor VHEM-H3M over PPK-SC and over standard EM, suggesting that VHEM-H3M is more robust and effective for H3M density estimation. Finally, in all the experiments, the running time of VHEM-H3M compares favorably with the other HMM clustering algorithms; PPK-SC suffers long delays when the number of input HMMs is large and the standard EM algorithm is considerably slower. This demonstrates that VHEM-H3M is most efficient for clustering HMMs.

[7] We can show that the VHEM algorithm performs clustering based on KL divergence. Letting $\mathscr{D}^{i,j}_{HMM} = \mathscr{L}^{i,i}_{HMM} - \mathscr{L}^{i,j}_{HMM} \approx D(\mathscr{M}_i^{(b)} || \mathscr{M}_j^{(r)})$ be an approximation to the KL using (4.21), we can rewrite the E-step as $\hat{z}_{ij} \propto \omega_j^{(r)} e^{-N\omega_i^{(b)} \mathscr{D}^{i,j}_{HMM}}$. Similarly, the M-step is $\hat{\mathscr{M}}_j^{(r)} = \arg\min_{\mathscr{M}_j^{(r)}} \sum_{i=1}^{K^{(b)}} \omega_i^{(b)} \hat{z}_{ij} \mathscr{D}^{i,j}_{HMM}$.

4.5 Clustering Experiments

In this section, we present an empirical study of the VHEM-H3M algorithm for clustering and hierarchical clustering of HMMs. Clustering HMMs consists in partitioning K_1 input HMMs into $K_2 < K_1$ groups of similar HMMs. Hierarchical clustering involves organizing the input HMMs in a multi-level hierarchy with h levels, by applying clustering in a recursive manner. Each level ℓ of the hierarchy has K_ℓ groups (with $K_1 > K_2 > \cdots > K_{h-1} > K_h$), and the first level consists of the K_1 input HMMs.

We begin with an experiment on hierarchical clustering, where each of the input HMMs to be clustered is estimated on a sequence of motion capture data (Section 4.5.2). Then, we present a simulation study on clustering synthetic HMMs (Section 4.5.3). First, we provide an overview of the different algorithms used in this study.

4.5.1 Clustering Methods

In the clustering experiments, we will compare our VHEM-H3M algorithm with several other clustering algorithms. The various algorithms are summarized here.

- **VHEM-H3M**: We cluster K_1 input HMMs into K_2 clusters by using the VHEM-H3M algorithm (on the input HMMs) to learn a H3M with K_2 components (as explained in Section 4.3.1). To build a multi-level hierarchy of HMMs with h levels, we start from the first level of K_1 input HMMs, and recursively use the VHEM-H3M algorithm $h-1$ times. Each new level ℓ is formed by clustering the $K_{\ell-1}$ HMMs at the previous level into $K_\ell < K_{\ell-1}$ groups with the VHEM-H3M algorithm, and using the learned HMMs as cluster centers at the new level. In our experiments, we set the number of virtual samples to $N = 10^4 K^{(\ell-1)}$, a large value that favors "hard" clustering (where each HMM is univocally assigned to a single cluster), and the length of the virtual sequences to $\tau = 10$.

- **PPK-SC**: [86] cluster HMMs by calculating a PPK similarity matrix between all HMMs, and then applying spectral clustering. The work in [86] only considered HMMs with single Gaussian emissions, which did not always give satisfactory results in our experiments. Hence, we extended the method of [86] by allowing GMM emissions, and derived the PPK similarity for this more general case [85]. From preliminary experiments, we found the best performance for PPK with $\lambda = \frac{1}{2}$ (i.e., Bhattacharyya affinity), and when integrating over sequences of length $\tau = 10$. Finally, we also extend [86] to construct multi-level hierarchies, by using hierarchical k-means in the spectral clustering embedding.

- **SHEM-H3M**: This is a version of HEM-H3M that maximizes the likelihood of *actual* samples generated from the input HMMs, as in (4.5), rather than the expectation of virtual samples, as in (4.6). In particular, from each input HMM $\mathscr{M}_i^{(b)}$ we sample a set Y_i of $N_i = \pi_i^{(b)} N$ observation sequences (for a large value of N). We then estimate the reduced H3M from the N samples $Y = \{Y_i\}_{i=1}^{K^{(b)}}$, with the EM-H3M algorithm of [151], which was modified to use a single assignment variable for each sample set Y_i, to obtain a consistent clustering.

In many real-life applications, the goal is to cluster a collection of time series, i.e., observed sequences. Although the input data is not a collection of HMMs in that case, it can still be clustered with the VHEM-H3M algorithm by first modeling each sequence as an HMM, and then using the HMMs as input for the VHEM-H3M algorithm. With time-series data as input, it is also possible to use clustering approaches that do not model each sequence as a HMM. Hence, in one of the hierarchical motion clustering experiments, we also compare to the following two algorithms, one that clusters time-series data directly [151], and a second one that clusters the time series after modeling each sequence with a dynamic texture (DT) model [33].

- **EM-H3M**: The EM algorithm for H3Ms [151] is applied directly on a collection of time series to learn the clustering and HMM cluster centers, thus bypassing the intermediate HMM modeling stage. To obtain a hierarchical clustering (with $h \geq 3$ levels), we proceed in a bottom up fashion and build each new level by simply re-clustering the given time series in a smaller number of clusters using the EM algorithm by [151]. We extend the algorithm to use a single assignment variable for each set of sequences Y_i that are within the same cluster in the immediately lower level of the hierarchy. This modification preserves the hierarchical clustering property that sequences in a cluster will remain together at the higher levels.

- **HEM-DTM**: Rather than use HMMs, we consider a clustering model based on linear dynamical systems, i.e., dynamic textures (DTs) [54]. Hierarchical clustering is performed using the hierarchical EM algorithm for DT mixtures (HEM-DTM) [33], in an analogous way to VHEM-H3M. The main difference is that, with HEM-DTM, time-series are modeled as DTs, which have a *continuous* state space (a Gauss-Markov model) and *unimodal* observation model, whereas VHEM-H3M uses a *discrete* state space and *multimodal* observations (GMMs).

We will use several metrics to quantitatively compare the results of different clustering algorithms. First, we will calculate the *Rand-index* [83], which measures the correctness of a proposed clustering against a given ground truth clustering. Intuitively, this index measures how consistent cluster assignments are with the ground truth (i.e., whether pairs of items are correctly or incorrectly assigned to the same cluster, or different clusters). Second, we will consider the *log-likelihood*, as used by [151] to evaluate a clustering. This measures how well the clustering fits the input data. When time series are given as input data, we compute the log-likelihood of a clustering as the sum of the log-likelihoods of each input sequence under the HMM cluster center to which it has been

assigned. When the input data consists of HMMs, we will evaluate the log-likelihood of a clustering by using the expected log-likelihood of observations generated from an input HMM under the HMM cluster center to which it is assigned. For PPK-SC, the cluster center is estimated by running the VHEM-H3M algorithm (with $K^{(r)} = 1$) on the HMMs assigned to the cluster.[8] Note that the log-likelihood will be particularly appropriate to compare VHEM-H3M, SHEM-H3M, EM-H3M and HEM-DTM, since they explicitly optimize for it. However, it may be unfair for PPK-SC, since this method optimizes the PPK similarity and not the log-likelihood. As a consequence, we also measure the *PPK cluster-compactness*, which is more directly related to what PPK-SC optimizes for. The PPK cluster-compactness is the sum (over all clusters) of the average intra-cluster PPK pair-wise similarity. This performance metric favors methods that produce clusters with high intra-cluster similarity.

Note that, *time series* can also be clustered with recourse to similarity measures based on dynamic time warping [124, 92, 91] or methods that rely on non-parametric sequence kernels [100, 26, 96, 39], which have shown good performance in practice. In this work we focus on the problem of clustering *hidden Markov models*, so we do not pursue an empirical evaluation of these methods.

4.5.2 Hierarchical Motion Clustering

In this experiment we test the VHEM algorithm on hierarchical motion clustering from motion capture data, i.e., time series representing human locomotions and actions. To hierarchically cluster a collection of time series, we first model each time series with an HMM and then cluster the HMMs hierarchically. Since each HMM summarizes the appearance and dynamics of the particular motion sequence it represents, the structure encoded in the hierarchy of HMMs directly applies to the original motion sequences.

[8] Alternatively, we could use as cluster center the HMM mapped the closest to the spectral embedding cluster center, but this always resulted in lower log-likelihood.

(a) “Walk” sequence.

(b) “Run” sequence.

Figure 4.1. Examples of motion capture sequences from the MoCap dataset, shown with stick figures.

[86] uses a similar approach to cluster motion sequences, applying PPK-SC to cluster HMMs. However, they did not extend their study to hierarchies with multiple levels.

Datasets and Setup

We experiment on two motion capture datasets, the MoCap dataset (http://mocap.cs.cmu.edu/) and the Vicon Physical Action dataset [155, 9]. For the MoCap dataset, we use 56 motion examples spanning 8 different classes (“jump”, “run”, “jog”, “walk 1”, “walk 2”, “basket”, “soccer”, and “sit”). Each example is a sequence of 123-dimensional vectors representing the (x,y,z)-coordinates of 41 body markers tracked spatially through time. Figure 4.1 illustrates some typical examples. We built a hierarchy of $h = 4$ levels. The first level (Level 1) was formed by the $K_1 = 56$ HMMs learned from each individual motion example (with $S = 4$ hidden states, and $M = 2$ components for each GMM emission). The next three levels contain $K_2 = 8$, $K_3 = 4$ and $K_4 = 2$ HMMs. We perform the hierarchical clustering with VHEM-H3M, PPK-SC, EM-H3M, SHEM-H3M ($N \in \{560, 2800\}$ and $\tau = 10$), and HEM-DTM (state dimension of 7). The experiments were repeated 10 times for each clustering method, using different random initializations of the algorithms.

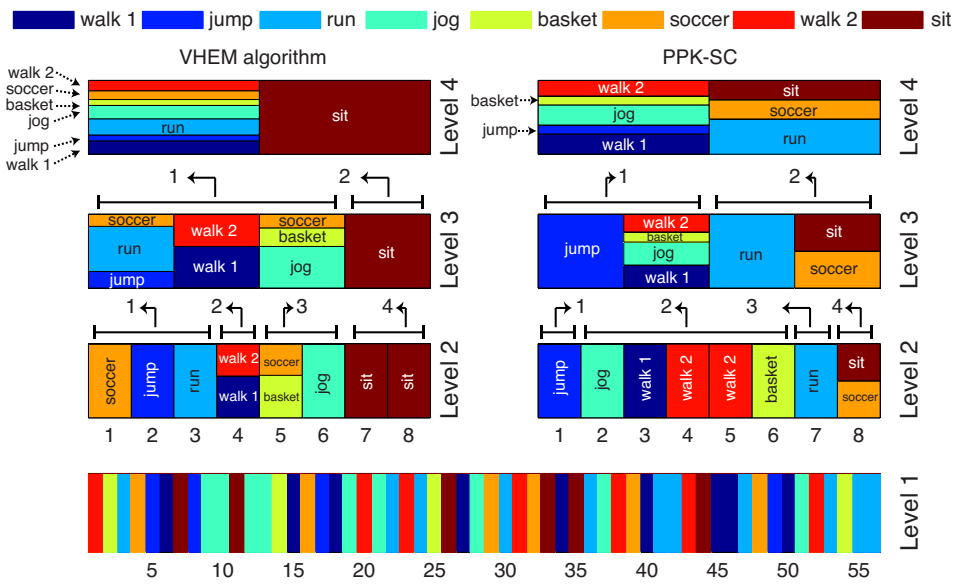


Figure 4.2. An example of hierarchical clustering of the MoCap dataset, with VHEM-H3M and PPK-SC. Different colors represent different motion classes. Vertical bars represent clusters, with the colors indicating the proportions of the motion classes in a cluster, and the numbers on the x-axes representing the clusters' indexes. At Level 1 there are 56 clusters, one for each motion sequence. At Levels 2, 3 and 4 there are 8, 4 and 2 HMM clusters, respectively. For VHEM almost all clusters at Level 2 are populated by examples from a single motion class. The error of VHEM in clustering a portion of "soccer" with "basket" is probably because both actions involve a sequence of movement, shot, and pause. Moving up the hierarchy, the VHEM algorithm clusters similar motions classes together, and at Level 4 creates a dichotomy between "sit" and the other (more dynamic) motion classes. PPK-SC also clusters motion sequences well at Level 2, but incorrectly aggregates "sit" and "soccer", which have quite different dynamics. At Level 4, the clustering obtained by PPK-SC is harder to interpret than that by VHEM.

The Vicon Physical Action dataset is a collection of 200 motion sequences. Each sequence consists of a time series of 27-dimensional vectors representing the (x,y,z)-coordinates of 9 body markers captured using the Vicon 3D tracker. The dataset includes 10 normal and 10 aggressive activities, performed by each of 10 human subjects a single time. We build a hierarchy of $h = 5$ levels, starting with $K_1 = 200$ HMMs (with $S = 4$ hidden states and $M = 2$ components for each GMM emission) at the first level (i.e., one for each motion sequence), and using $K_2 = 20$, $K_3 = 8$, $K_4 = 4$, and $K_5 = 2$ for the next four levels. The experiment was repeated 5 times with VHEM-H3M and PPK-SC, using

different random initializations of the algorithms.

In similar experiments where we varied the number of levels h of the hierarchy and the number of clusters at each level, we noted similar relative performances of the various clustering algorithms, on both datasets.

Results on the MoCap Dataset

An example of hierarchical clustering of the MoCap dataset with VHEM-H3M is illustrated in Figure 4.2 (left). In the first level, each vertical bar represents a motion sequence, with different colors indicating different ground-truth classes. In the second level, the $K_2 = 8$ HMM clusters are shown with vertical bars, with the colors indicating the proportions of the motion classes in the cluster. Almost all clusters are populated by examples from a single motion class (e.g., "run", "jog", "jump"), which demonstrates that VHEM can group similar motions together. We note an error of VHEM in clustering a portion of the "soccer" examples with "basket". This is probably caused by the fact that both types of actions begin with a stationary phase (e.g., subject focusing on the execution) followed with a forward movement (note that our "basket" examples correspond to forward dribbles). Moving up the hierarchy, the VHEM algorithm clusters similar motion classes together (as indicated by the arrows), for example "walk 1" and "walk 2" are clustered together at Level 2, and at the highest level (Level 4) it creates a dichotomy between "sit" and the rest of the motion classes. This is a desirable behavior as the kinetics of the "sit" sequences (which in the MoCap dataset correspond to starting in a standing position, sitting on a stool, and returning to a standing position) are considerably different from the rest. On the right of Figure 4.2, the same experiment is repeated with PPK-SC. PPK-SC clusters motion sequences properly, but incorrectly aggregates "sit" and "soccer" at Level 2, even though they have quite different dynamics. Furthermore, the highest level (Level 4) of the hierarchical clustering produced by PPK-SC is harder

Table 4.2. Hierarchical clustering of the MoCap dataset using VHEM-H3M, PPK-SC, SHEM-H3M, EM-H3M and HEM-DTM. The number in brackets after SHEM-H3M represents the number of real samples used. We computed Rand-index, data log-likelihood and cluster compactness at each level of the hierarchy, and registered the time (in seconds) to learn the hierarchical structure. Differences in Rand-index at Levels 2, 3, and 4 are statistically significant based on a paired t-test with confidence 95%.

	Rand-index			log-likelihood ($\times 10^6$)			PPK cluster-compactness			time (s)
Level	2	3	4	2	3	4	2	3	4	
VHEM-H3M	0.937	**0.811**	**0.518**	**-5.361**	**-5.682**	**-5.866**	0.0075	**0.0068**	**0.0061**	**30.97**
PPK-SC	**0.956**	0.740	0.393	-5.399	-5.845	-6.068	0.0082	0.0021	0.0008	37.69
SHEM-H3M (560)	0.714	0.359	0.234	-13.632	-69.746	-275.650	0.0062	0.0034	0.0031	843.89
SHEM-H3M (2800)	0.782	0.685	0.480	-14.645	-30.086	-52.227	0.0050	0.0036	0.0030	3849.72
EM-H3M	0.831	0.430	0.340	-5.713	-202.55	-168.90	**0.0099**	0.0060	0.0056	667.97
HEM-DTM	0.897	0.661	0.412	-7.125	-8.163	-8.532	-	-	-	121.32

to interpret than that of VHEM.

Table 4.2 presents a quantitative comparison between PPK-SC and VHEM-H3M at each level of the hierarchy. While VHEM-H3M has lower Rand-index than PPK-SC at Level 2 (0.937 vs. 0.956), VHEM-H3M has higher Rand-index at Level 3 (0.811 vs. 0.740) and Level 4 (0.518 vs. 0.393). In terms of PPK cluster-compactness, we observe similar results. In particular, VHEM-H3M has higher PPK cluster-compactness than PPK-SC at Level 3 and 4. Overall, keeping in mind that PPK-SC is explicitly driven by PPK-similarity, while the VHEM-H3M algorithm is not, these results can be considered as strongly in favor of VHEM-H3M (over PPK-SC). In addition, the data log-likelihood for VHEM-H3M is higher than that for PPK-SC at each level of the hierarchy. This suggests that the novel HMM cluster centers learned by VHEM-H3M fit the motion capture data better than the spectral cluster centers, since they condense information of the entire underlying clusters. This conclusion is further supported by the results of the density estimation experiments in Sections 4.6.1 and 4.6.2. Note that the higher up in the hierarchy, the more clearly this effect is manifested.

Comparing to other methods (also in Table 4.2), EM-H3M generally has lower

Rand-index than VHEM-H3M and PPK-SC (consistent with the results from [86]). While EM-H3M directly clusters the original motion sequences, both VHEM-H3M and PPK-SC implicitly integrate over all possible virtual variations of the original motion sequences (according to the intermediate HMM models), which results in more robust clustering procedures. In addition, EM-H3M has considerably longer running times than VHEM-H3M and PPK-SC (i.e., roughly 20 times longer) since it needs to evaluate the likelihood of all training sequences at each iteration, at all levels.

The results in Table 4.2 favor VHEM-H3M over SHEM-H3M, and empirically validate the variational approximation that VHEM uses for learning. For example, when using $N = 2800$ samples, running SHEM-H3M takes over two orders of magnitude more time than VHEM-H3M, but still does not achieve performance competitive with VHEM-H3M. With an efficient closed-form expression for averaging over all possible virtual samples, VHEM approximates the sufficient statistics of a virtually unlimited number of observation sequences, without the need of using real samples. This has an additional regularization effect that improves the robustness of the learned HMM cluster centers. In contrast, SHEM-H3M uses real samples, and requires a large number of them to learn accurate models, which results in significantly longer running times.

In Table 4.2, we also report hierarchical clustering performance for HEM-DTM. VHEM-H3M consistently outperforms HEM-DTM, both in terms of Rand-index and data log-likelihood.[9] Since both VHEM-H3M and HEM-DTM are based on the hierarchical EM algorithm for learning the clustering, this indicates that HMM-based clustering models are more appropriate than DT-based models for the human MoCap data. Note that, while PPK-SC is also HMM-based, it has a lower Rand-index than HEM-DTM at Level 4. This further suggests that PPK-SC does not optimally cluster the HMMs.

[9] We did not report PPK cluster-compactness for HEM-DTM, since it would not be directly comparable with the same metric based on HMMs.

Finally, to assess stability of the clustering results, starting from the 56 HMMs learned on the motion examples, in turn we exclude one of them and build a hierarchical clustering of the remaining ones ($h = 4$ levels, $K_1 = 55, K_2 = 8, K_3 = 4, K_4 = 2$), using in turn VHEM-H3M and PPK-SC. We then compute cluster stability as the mean Rand-index between all possible pairs of clusterings. The experiment is repeated 10 times for both VHEM-H3M and PPK-SC, using a different random initialization for each trial. The stability of VHEM-H3M is 0.817, 0.808 and 0.950, at Levels 2, 3 and 4. The stability of PPK-SC is 0.786, 0.837 and 0.924, at Levels 2, 3 and 4. Both methods are fairly stable in terms of Rand-index, with a slight advantage for VHEM-H3M over PPK-SC (with average across levels of 0.858 versus 0.849).

The experiments were repeated 10 times for each clustering method, using different random initializations of the algorithms.

Alternatively, we evaluate the out-of-sample generalization of the clusterings discovered by VHEM-H3M and PPK-SC, by computing the fraction of times the held out HMMs is assigned[10] to the cluster that contains the majority of HMMs from its same ground truth class. Out of sample generalization of VHEM-H3M and PPK-SC are comparable, registering an averages of 0.875, respectively, 0.851 across the different levels.

Results on the Vicon Physical Action Dataset

Table 4.3 presents results using VHEM-H3M and PPK-SC to cluster the Vicon Physical Action dataset. While the two algorithms performs similarly in terms of Rand-index at lower levels of the hierarchy (i.e., Level 2 and Level 3), at higher levels (i.e., Level 4 and Level 5) VHEM-H3M outperforms PPK-SC. In addition, VHEM-H3M registers higher data log-likelihood than PPK-SC at each level of the hierarchy. This, again,

[10]For VHEM-H3M we use the expected log-likelihood, for PPK-SC we use the out-of-sample extension for spectral clustering of [19]

Table 4.3. Hierarchical clustering of the Vicon Physical Action dataset using VHEM-H3M and PPK-SC. Performance is measured in terms of Rand-index, data log-likelihood and PPK cluster-compactness at each level. Differences in Rand-index at Levels 2, 4 and 5 are statistically significant based on a paired t-test with confidence 95%. The test failed at Level 3.

	Rand-index				log-likelihood ($\times 10^6$)				PPK cluster-compactness			
Level	2	3	4	5	2	3	4	5	2	3	4	5
VHEM-H3M	0.909	0.805	0.610	0.244	-1.494	-3.820	-5.087	-6.172	0.224	0.059	0.020	0.005
PPK-SC	0.900	0.807	0.452	0.092	-3.857	-5.594	-6.163	-6.643	0.324	0.081	0.026	0.008

suggests that by learning new cluster centers, the VHEM-H3M algorithm retains more information on the clusters' structure than PPK-SC. Finally, compared to VHEM-H3M, PPK-SC produces clusters that are more compact in terms of PPK similarity. However, this does not necessarily imply a better agreement with the ground truth clustering, as evinced by the Rand-index metrics.

4.5.3 Clustering Synthetic Data

In this experiment, we compare VHEM-H3M and PPK-SC on clustering a synthetic dataset of HMMs.

Dataset and Setup

The synthetic dataset of HMMs is generated as follows. Given a set of C HMMs $\{\mathscr{M}^{(c)}\}_{c=1}^{C}$, for each HMM we synthesize a set of K "noisy" versions of the original HMM. Each "noisy" HMM $\tilde{\mathscr{M}}_k^{(c)}$ $(k = 1, \ldots K)$ is synthesized by generating a random sequence $y_{1:T}$ of length T from $\mathscr{M}^{(c)}$, corrupting it with Gaussian noise $\sim \mathscr{N}(0, \sigma_n^2 \mathrm{I}_d)$, and estimating the parameters of $\tilde{\mathscr{M}}_k^{(c)}$ on the corrupted version of $y_{1:T}$. Note that this procedure adds noise in the *observation* space. The number of noisy versions (of each given HMM), K, and the noise variance, σ_n^2, will be varied during the experiments.

The collection of original HMMs was created as follows. Their number was

always set to $C = 4$, the number of hidden states of the HMMs to $S = 3$, the emission distributions to be single, one-dimensional Gaussians (i.e., GMMs with $M = 1$ component), and the length of the sequences to $T = 100$. For all original HMMs $\mathscr{M}^{(c)}$, *if not otherwise specified*, the initial state probability, the state transition matrix, and the means and variances of the emission distributions were fixed as

$$\pi^{(c)} = \begin{bmatrix} 1/3 \\ 1/3 \\ 1/3 \end{bmatrix}, \quad A^{(c)} = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.8 & 0 \\ 0 & 0.2 & 0.8 \end{bmatrix}, \quad \begin{cases} \mu_1^{(c)} = 1 \\ \mu_2^{(c)} = 2 \\ \mu_3^{(c)} = 3 \end{cases} \quad {\sigma_\rho^{(c)}}^2 = 0.5 \; \forall \rho, \quad \forall c.$$

We consider three different experimental settings. In the first setting, experiment (a), the HMMs $\mathscr{M}^{(c)}$ only differ in the means of the emission distributions,

$$\begin{cases} \mu_1^{(1)} = 1 \\ \mu_2^{(1)} = 2 \\ \mu_3^{(1)} = 3 \end{cases}, \begin{cases} \mu_1^{(2)} = 3 \\ \mu_2^{(2)} = 2 \\ \mu_3^{(2)} = 1 \end{cases}, \begin{cases} \mu_1^{(3)} = 1 \\ \mu_2^{(3)} = 2 \\ \mu_3^{(3)} = 2 \end{cases}, \begin{cases} \mu_1^{(4)} = 1 \\ \mu_2^{(4)} = 3 \\ \mu_3^{(4)} = 3 \end{cases} \tag{4.32}$$

In the second setting, experiment (b), the HMMs differ in the variances of the emission distributions,

$${\sigma_\rho^{(1)}}^2 = 0.5, \quad {\sigma_\rho^{(2)}}^2 = 0.1, \quad {\sigma_\rho^{(3)}}^2 = 1, \quad {\sigma_\rho^{(4)}}^2 = 0.05, \quad \forall \rho. \tag{4.33}$$

In the last setting, experiment (c), the HMMs differ in the transition matrices,

$$A^{(1)} = \begin{bmatrix} 0.8 & 0.1 & 0.1 \\ 0.2 & 0.8 & 0 \\ 0 & 0.2 & 0.8 \end{bmatrix} \quad A^{(2)} = \begin{bmatrix} 0.2 & 0.2 & 0.2 \\ 0.4 & 0.6 & 0 \\ 0 & 0.4 & 0.6 \end{bmatrix} \tag{4.34}$$

$$A^{(3)} = \begin{bmatrix} 0.9 & 0.05 & 0.05 \\ 0.1 & 0.9 & 0 \\ 0 & 0.1 & 0.9 \end{bmatrix} \quad A^{(4)} = \begin{bmatrix} 0.4 & 0.3 & 0.4 \\ 0.6 & 0.4 & 0 \\ 0 & 0.6 & 0.4 \end{bmatrix}.$$

The VHEM-H3M and PPK-SC algorithms are used to cluster the synthesized HMMs, $\{\{\tilde{\mathscr{M}}_k^{(c)}\}_{k=1}^K\}_{c=1}^C$, into C groups, and the quality of the resulting clusterings is measured with the Rand-index, PPK cluster-compactness, and the expected log-likelihood of the discovered cluster centers with respect to the original HMMs. The expected log-likelihood was computed using the lower bound, as in (4.13), with each of the original HMMs assigned to the most likely HMM cluster center. The results are averages over 10 trials.

The reader is referred to Appendix A.4.2 for experiments where the order of the model used for clustering does not match the order of the true model used for generating the data.

Results

Figure 4.3 reports the performance metrics when varying the number $K \in \{2,4,8,16,32\}$ of noisy versions of each of the original HMMs, and the noise variance $\sigma_n^2 \in \{0.1, 0.5, 1\}$, for the three experimental settings. (Note that, in each trial, for each class, we first generated 32 noisy HMMs per class, and then, varying $K \in \{4,8,16,32\}$, we subsampled only K of them.) For the majority of settings of K and σ_n^2, the clustering produced by VHEM-H3M is superior to the one produced by PPK-SC, for each of the

considered metrics (i.e., in the plots, solid lines are usually above dashed lines of the same color). The only exception is in experiment (b) where, for low noise variance (i.e., $\sigma_n^2 = 0.1$) PPK-SC is the best in terms of Rand-index and cluster compactness. It is interesting to note that the gap in performance between VHEM-H3M and PPK-SC is generally larger at low values of K. We believe this is because, when only a limited number of input HMMs is available, PPK-SC produces an embedding of lower quality. This does not affect VHEM-H3M, since it clusters in HMM distribution space and does not use an embedding.

Note that the Rand-Index values for experiment (c) (i.e., different state transition matrices) are superior to the corresponding ones in experiments (a) and (b) (i.e., different emission distributions) for both VHEM-H3M and PPK-SC. This shows that VHEM-H3M (and slightly less robustly PPK-SC as well) can cluster dynamics characterized by different hidden states processes more easily than dynamics that differ only in the emission distributions. In addition, VHEM-H3M is more robust to noise than PPK-SC, as demonstrated by the experiments with $\sigma_n^2 = 1$ (blue lines).

These results suggest that, by clustering HMMs *directly* in distribution space, VHEM-H3M is generally more robust than PPK-SC, the performance of which instead depends on the quality of the underlying embedding.

Finally, the results in Figure 4.3 show only small fluctuations across different values of K at each noise value, suggesting stability of the clustering results (*relative to the ground truth clustering*) for both VHEM-H3M and PPK-SC, to both subsampling and jittering (i.e., addition of noise) [76]. In particular, from the expected log-likelihood values (central column in Figure 4.3), we evince that the similarity of the discovered clustering to the true distribution (i.e., the original HMMs) is not largely affected by the amount of subsampling, except when only as little as the 12.5% of the data is used (when $K = 4$).

4.6 Density Estimation Experiments

In this section, we present an empirical study of VHEM-H3M for density estimation, in automatic annotation and retrieval of music (Section 4.6.1) and hand-written digits (Section 4.6.2).

4.6.1 Music Annotation and Retrieval

In this experiment, we evaluate VHEM-H3M for estimating annotation models in content-based music auto-tagging. As a generative time-series model, H3Ms allow to account for timbre (i.e., through the GMM emission process) as well as longer term temporal dynamics (i.e., through the HMM hidden state process), when modeling musical signals. Therefore, in music annotation and retrieval applications, H3Ms are expected to prove more effective than existing models that do not explicitly account for temporal information [159, 106, 55, 81].

Music Dataset

We consider the CAL500 collection from [159], which consists of 502 songs and provides binary annotations with respect to a vocabulary $\mathscr{V}$ of 149 tags, ranging from genre and instrumentation, to mood and usage. To represent the acoustic content of a song we extract a time series of audio features $\mathscr{Y} = \{y_1, \ldots, y_{|\mathscr{Y}|}\}$, by computing the first 13 Mel frequency cepstral coefficients (MFCCs) [135] over half-overlapping windows of 46ms of audio signal, augmented with first and second instantaneous derivatives. The song is then represented as a collection of *audio fragments*, which are sequences of $T = 125$ audio features (approximately 6 seconds of audio), using a dense sampling with 80% overlap.

Music Annotation Models

Automatic music tagging is formulated as a supervised multi-label problem [28], where each class is a tag from $\mathcal{V}$. We approach this problem by modeling the audio content for each tag with a H3M probability distribution. I.e., for each tag, we estimate an H3M over the audio fragments of the songs in the database that have been associated with that tag, using the hierarchical estimation procedure based on VHEM-H3M. More specifically, the database is first processed at the song level, using the EM algorithm to learn a H3M with $K^{(s)} = 6$ components for each song[11] from its audio fragments. Then, for each tag, the song-level H3Ms labeled with that tag are pooled together to form a large H3M, and the VHEM-H3M algorithm is used to reduce this to a final H3M tag-model with $K = 3$ components ($\tau = 10$ and $N = N_v N_t K^{(s)}$, where $N_v = 1000$ and N_t is the number of training songs for the particular tag).

Given the tag-level models, a song can be represented as a vector of posterior probabilities of each tag (a semantic multinomial, SMN), by extracting features from the song, computing the likelihood of the features under each tag-level model, and applying Bayes' rule.[12] A test song is annotated with the top-ranking tags in its SMN. To retrieve songs given a tag query, a collection of songs is ranked by the tag's probability in their SMNs.

We compare VHEM-H3M with three alternative algorithms for estimating the H3M tag models: PPK-SC, PPK-SC-hybrid, and EM-H3M.[13] For all three alternatives,

[11] Most pop songs have 6 structural parts: intro, verse, chorus, solo, bridge and outro.

[12] We compute the likelihood of a song under a tag model as the geometric average of the likelihood of the individual segments, and further normalized it by the length of the segments to prevent the posteriors from being too "peaked" [42].

[13] For this experiment, we were not able to successfully estimate accurate H3M tag models with SHEM-H3M. In particular, SHEM-H3M requires generating an appropriately large number of real samples to produce accurate estimates. However, due to the computational limits of our cluster, we were able to test SHEM-H3M only using a small number of samples. In preliminary experiments we registered performance only slightly above chance level and training times still twice longer than for VHEM-H3M. For a comparison between VHEM-H3M and SHEM-H3M on density estimation, the reader can refer to

we use the same number of mixture components in the tag models ($K = 3$). For the two PPK-SC methods, we leverage the work of [86] to learn H3M tag models, and use it in place of the VHEM-H3M algorithm in the second stage of the hierarchical estimation procedure. We found that it was necessary to implement the PPK-SC approaches with song-level H3Ms with only $K^{(s)} = 1$ component (i.e., a single HMM), since the computational cost for constructing the initial embedding scales poorly with the number of input HMMs.[14] PPK-SC first applies spectral clustering to the song-level HMMs and then selects as the cluster centers the HMMs that map closest to the spectral cluster centers in the spectral embedding. PPK-SC-hybrid is a hybrid method combining PPK-SC for clustering, and VHEM-H3M for estimating the cluster centers. Specifically, after spectral clustering, HMM cluster centers are estimated by applying VHEM-H3M (with $K^{(r)} = 1$) to the HMMs assigned to each of the resulting clusters. In other words, PPK-SC and PPK-SC-hybrid use spectral clustering to summarize a collection of song-level HMMs with a few HMM centers, forming a H3M tag model. The mixture weight of each HMM component (in the H3M tag model) is set proportional to the number of HMMs assigned to that cluster.

For EM-H3M, the H3M tag models were estimated directly from the audio fragments from the relevant songs using the EM-H3M algorithm.[15] Empirically, we found that, due to its runtime and RAM requirements, for EM-H3M we must use non-overlapping audio-fragments and evenly subsample by 73% on average, resulting in 14.5% of the sequences used by VHEM-H3M. Note that, however, EM-H3M is still using 73% of the actual song data (just with non-overlapping sequences). We believe this to be a reasonable comparison between EM and VHEM, as both methods use roughly

the experiment in Section 4.6.2 on online hand-writing classification and retrieval.

[14] Running PPK-SC with $K^{(s)} = 2$ took 3958 hours in total (about 4 times more than when setting $K^{(s)} = 1$), with no improvement in annotation and retrieval performance. A larger $K^{(s)}$ would yield impractically long learning times.

[15] The EM algorithm has been used to estimate HMMs from music data in previous work [144, 140].

similar resources (the sub-sampled EM is still 3 times slower, as reported in Table 6.4). Based on our projections, running EM over densely sampled song data would require roughly 9000 hours of CPU time (e.g., more than 5 weeks when parallelizing the algorithm over 10 processors), as opposed to 630 hours for VHEM-H3M. This would be extremely cpu-intensive given the computational limits of our cluster. The VHEM algorithm, on the other hand, can learn from considerable amounts of data while still maintaining low runtime and memory requirements.[16]

Finally, we also compare against two state-of-the-art models for music tagging, HEM-DTM [42], which is based on a different time-series model (mixture of dynamic textures), and HEM-GMM [159], which is a bag-of-features model using GMMs. Both methods use efficient hierarchical estimation based on a HEM algorithm [33, 164] to obtain the tag-level models.[17]

Performance Metrics

A test song is annotated with the 10 most likely tags, and annotation performance is measured with the per-tag precision (P), recall (R), and F-score (F), averaged over all tags. If $|w_H|$ is the number of test songs that have the tag w in their ground truth annotations, $|w_A|$ is the number of times an annotation system uses w when automatically tagging a song, and $|w_C|$ is the number of times w is correctly used, then precision, recall and F-score for the tag w are defined as:

$$\mathrm{P} = \frac{|w_C|}{|w_A|},\ \mathrm{R} = \frac{|w_C|}{|w_H|},\ \mathrm{F} = 2\left((\mathrm{P})^{-1} + (\mathrm{R})^{-1}\right)^{-1}. \tag{4.35}$$

[16]For example, consider learning a tag-level H3M from 200 songs, which corresponds to over 3GB of audio fragments. Using the hierarchical estimation procedure, we first model each song (in average, 15MB of audio fragments) individually as a song-level H3M, and we save the song models (150 KB of memory each). Then, we pool the 200 song models into a large H3M (in total 30MB of memory), and reduce it to a smaller tag-level H3M using the VHEM-H3M algorithm.

[17]Both auto-taggers operate on audio features extracted over half-overlapping windows of 46ms. HEM-GMM uses MFCCs with first and second instantaneous derivatives [159]. HEM-DTM uses 34-bins of Mel-spectral features [42], which are further grouped in audio fragments of 125 consecutive features.

Table 4.4. Annotation and retrieval performance on CAL500, for VHEM-H3M, PPK-SC, PPK-SC-*hybrid*, EM-H3M, HEM-DTM [42] and HEM-GMM [159].

	annotation			**retrieval**				
	P	R	F	MAP	P@5	P@10	P@15	time (h)
VHEM-H3M	0.446	0.211	0.260	0.440	0.474	0.451	0.435	629.5
PPK-SC	0.299	0.159	0.151	0.347	0.358	0.340	0.329	974.0
PPK-SC-*hybrid*	0.407	0.200	0.221	0.415	0.439	0.421	0.407	991.7
EM-H3M	0.415	0.214	0.248	0.423	0.440	0.422	0.407	1860.4
HEM-DTM	0.431	0.202	0.252	0.439	0.479	0.454	0.428	-
HEM-GMM	0.374	0.205	0.213	0.417	0.441	0.425	0.416	-

Retrieval is measured by computing per-tag mean average precision (MAP) and precision at the first k retrieved songs (P@k), for $k \in \{5, 10, 15\}$. The P@k is the fraction of true positives in the top-k of the ranking. MAP averages the precision at each point in the ranking where a song is correctly retrieved. All reported metrics are averages over the 97 tags that have at least 30 examples in CAL500 (11 genre, 14 instrument, 25 acoustic quality, 6 vocal characteristics, 35 emotion and 6 usage tags), and are the result of 5-fold cross-validation.

Finally, we also record the total time (in hours) to learn the 97 tag-level H3Ms on the 5 splits of the data. For hierarchical estimation methods (VHEM-H3M and the PPK-SC approaches), this also includes the time to learn the song-level H3Ms.

Results

In Table 6.4 we report the performance of the various algorithms for both annotation and retrieval on the CAL500 dataset. Looking at the overall runtime, VHEM-H3M is the most efficient algorithm for estimating H3M distributions from music data, as it requires only 34% of the runtime of EM-H3M, and 65% of the runtime of PPK-SC. The VHEM-H3M algorithm capitalizes on the first stage of song-level H3M estimation

(about one third of the total time) by efficiently and effectively using the song-level H3Ms to learn the final tag models. Note that the runtime of PPK-SC corresponds to setting $K^{(s)} = 1$. When we set $K^{(s)} = 2$, we registered a running time four times longer, with no significant improvement in performance.

The gain in computational efficiency does not negatively affect the quality of the corresponding models. On the contrary, VHEM-H3M achieves better performance than EM-H3M,[18] strongly improving the top of the ranked lists, as evinced by the higher P@k scores. Relative to EM-H3M, VHEM-H3M has the benefit of regularization, and during learning can efficiently leverage all the music data condensed in the song H3Ms. VHEM-H3M also outperforms both PPK-SC approaches on all metrics. PPK-SC discards considerable information on the clusters' structure by selecting one of the original HMMs to approximate each cluster. This significantly affects the accuracy of the resulting annotation models. VHEM-H3M, on the other hand, generates novel HMM cluster centers to summarize the clusters. This allows to retain more accurate information in the final annotation models.

PPK-SC-hybrid achieves considerable improvements relative to standard PPK-SC, at relatively low additional computational costs.[19] This further demonstrates that the VHEM-H3M algorithm can effectively summarize in a smaller model the information contained in *several* HMMs. In addition, we observe that VHEM-H3M still outperforms PPK-SC-hybrid, suggesting that the former produces more accurate cluster centers and density estimates. In fact, VHEM-H3M *couples* clustering and learning HMM cluster centers, and is entirely based on maximum likelihood for estimating the H3M annotation models. PPK-SC-hybrid, on the contrary, separates clustering and parameter estimation,

[18] The differences in performance are statistically significant based on a paired t-test with 95% confidence.

[19] In PPK-SC-hybrid, each run of the VHEM-H3M algorithm converges quickly since there is only one HMM component to be learned, and can benefit from clever initialization (i.e., to the HMM mapped the closest to the spectral clustering center).

and optimizes them against two different metrics (i.e., PPK similarity and expected log-likelihood, respectively). As a consequence, the two phases may be mismatched, and the centers learned with VHEM may not be the best representatives of the clusters according to PPK affinity.

Finally, VHEM-H3M compares favorably to the auto-taggers based on other generative models. First, VHEM-H3M outperforms HEM-GMM, which does not model temporal information in the audio signal, on all metrics. Second, the performances of VHEM-H3M and HEM-DTM (a continuous-state temporal model) are not statistically different based on a paired t-test with 95% confidence, except for annotation precision where VHEM-H3M scores significantly higher. Since HEM-DTM is based on linear dynamic systems (a continuous-state model), it can model stationary time-series in a linear subspace. In contrast, VHEM-H3M uses HMMs with discrete states and GMM emissions, and can hence better adapt to non-stationary time-series on a non-linear manifold. This difference is illustrated in the experiments: VHEM-H3M outperforms HEM-DTM on the human MoCap data (see Table 4.2), which has non-linear dynamics, while the two perform similarly on the music data (see Table 6.4), where audio features are often stationary over short time frames.

4.6.2 On-Line Hand-Writing Data Classification and Retrieval

In this experiment, we investigate the performance of the VHEM-H3M algorithm in estimating class-conditional H3M distributions for automatic classification and retrieval of on-line hand-writing data.

Dataset

We consider the Character Trajectories Data Set [9], which is a collection of 2858 examples of characters from the same writer, originally compiled to study handwriting

motion primitives [169].[20] Each example is the trajectory of one of the 20 different characters that correspond to a single pen-down segment. The data was captured from a WACOM tablet at 200 Hz, and consists of (x, y)-coordinates and pen tip force. The data has been numerically differentiated and Gaussian smoothed [169]. Half of the data is used for training, with the other half held out for testing.

Classification Models and Setup

From the hand-writing examples in the training set, we estimate a series of class-conditional H3M distributions, one for each character, using hierarchical estimation with the VHEM-H3M algorithm. First, for each character, we partition all the relevant training data into groups of 3 sequences, and learn a HMM (with $S = 4$ states and $M = 1$ component for the GMM emissions) from each group using the Baum-Welch algorithm. Next, we estimate the class-conditional distribution (classification model) for each character by aggregating all the relevant HMMs and summarizing them into a H3M with $K = 4$ components using the VHEM-H3M algorithm ($\tau = 10$ and $N = N_v N_c$, where $N_v = 10$,[21] and N_c is the number of intermediate models for that character). Using the character-level H3Ms and Bayes' rule, for each hand-writing example in the test set we compute the posterior probabilities of all of the 20 characters. Each example is classified as the character with largest posterior probability. For retrieval given a query character, examples in the test set are ranked by the character's posterior probability.

We repeated the same experiment using PPK-SC or SHEM-H3M ($\tau = 10$, $N =$

[20] Even if we limit this experiment to the dataset of [169], we would like to refer the interest readers to the dataset of [102], which is a collection of *word* trajectories (as opposed to *character* trajectories).

[21] Note that choosing a lower value of N_v (compared to the music experiments) plays a role in making the clustering algorithm more *reliable*. Using fewer virtual samples equates to attaching smaller "virtual probability masses" to the input HMMs, and leads to *less certain* assignments of the input HMMs to the clusters (cf. equation 4.19). This determines more mixing in the initial iterations of the algorithm (e.g., similar to higher annealing temperature), and reduces the risk of prematurely specializing any cluster to one of the original HMMs. This effect is desirable, since the input HMMs are estimated over a smaller number of sequences (compared to the music experiments) and can therefore be noisier and less reliable.

1000) to estimate the classification models from the intermediate HMMs. Finally, we considered the EM-H3M algorithm, which directly uses the training sequences to learn the class-conditional H3M ($K = 4$).

Since VHEM-H3M, SHEM-H3M and EM-H3M are *iterative* algorithms, we studied them when varying the stopping criterion. In particular, the algorithms were terminated when the relative variation in the value of the objective function between two consecutive iterations was lower than a threshold ΔLL, which we varied in the set $\{10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}\}$.[22]

Finally, we measure classification and retrieval performance on the test set using the classification accuracy, and the average per-tag P@3, P@5 and MAP. We also report the total training time, which includes the time used to learn the intermediate HMMs. The experiments consisted of 5 trials with different random initializations of the algorithms.

Results

Table 4.5 lists the classification and retrieval performance on the test set for the various methods. Consistent with the experiments on music annotation and retrieval (Section 4.6.1), VHEM-H3M performs better than PPK-SC on all metrics. By learning novel HMM cluster centers, VHEM-H3M estimates H3M distributions that are representative of all the relevant intermediate HMMs, and hence of all the relevant training sequences. While EM-H3M is the best in classification (at the price of longer training times), VHEM-H3M performs better in retrieval, as evinced by the P@3 and P@5 scores. In terms of training time, VHEM-H3M and PPK-SC are about 5 times faster than EM-H3M. In particular, PPK-SC is the fastest algorithm, since the small number of input HMMs (i.e., on average 23 per character) allows to build the spectral clustering embedding efficiently.

The version of HEM based on actual sampling (SHEM-H3M) performs better than

[22] In a similar experiment where we used the number of iterations as the stopping criterion, we registered similar results.

Table 4.5. Classification and retrieval performance, and training time on the Character Trajectories Data Set, for VHEM-H3M, PPK-SC, SHEM-H3M, and EM-H3M.

	stop	**retrieval**			**classification**	
	ΔLL	P@3	P@5	MAP	Accuracy	total time (s)
VHEM-H3M	10^{-2}	0.750	0.750	0.738	0.618	1838.34
	10^{-3}	0.717	0.750	0.741	0.619	1967.55
	10^{-4}	0.733	0.790	0.773	0.648	2210.77
	10^{-5}	0.750	0.820	0.775	0.651	2310.93
SHEM-H3M	10^{-2}	0.417	0.440	0.517	0.530	13089.32
	10^{-3}	0.683	0.680	0.728	0.664	23203.44
	10^{-4}	0.700	0.750	0.753	0.689	35752.20
	10^{-5}	0.700	0.750	0.754	0.690	50094.36
EM-H3M	10^{-2}	0.583	0.610	0.667	0.646	6118.53
	10^{-3}	0.617	0.650	0.731	0.674	7318.56
	10^{-4}	0.650	0.710	0.756	0.707	9655.08
	10^{-5}	0.517	0.560	0.665	0.635	10957.38
PPK-SC	-	0.600	0.700	0.698	0.646	1463.54

VHEM-H3M in classification, but VHEM-H3M has higher retrieval scores. However, the training time for SHEM-H3M is approximately 15 times longer than for VHEM-H3M. These differences in performance can be understood in terms of the different types of *approximation* used by SHEM-H3M and VHEM-H3M. The sampling operation of SHEM-H3M provides an *unbiased* estimator, whose *variance* depends on the number of samples used. Hence, in order to reliably estimate the reduced model (i.e., making the variance small), the SHEM-H3M algorithm requires generating and handling a sufficiently large number of samples (relative to the model order/complexity, [75]). In contrast, the variational approximation of VHEM-H3M does not require generating samples (and hence avoids the problem of large variances associate to relatively small samples), but introduces a bias in the estimator [71]. Hence, it is not surprising that, on a relatively less complex problem where a relatively smaller sample size suffices, SHEM-H3M can perform more robustly than VHEM-H3M (even if still at a larger

computational cost).

It is also interesting to note that EM-H3M appears to suffer from overfitting of the training set, as suggested by the *overall drop* in performance when the stopping criterion changes from $\Delta LL = 10^{-4}$ to $\Delta LL = 10^{-5}$. In contrast, both VHEM-H3M and SHEM-H3M consistently improve on all metrics as the algorithm converges (again looking at $\Delta LL \in \{10^{-4}, 10^{-5}\}$). These results suggest that the regularization effect of hierarchical estimation, which is based on averaging over *more* samples (either virtual or actual), can positively impact the generalization of the learned models.[23]

Finally, we elaborate on how these results compare to the experiments on music annotation and retrieval (in Section 4.6.1). First, in the Character Trajectory Data Set the number of training sequences associated with each class (i.e, each character) is small compared to the CAL500 dataset.[24] As a result, the EM-H3M algorithm is able to process all the data, and achieve good classification performance. However, EM-H3M still needs to evaluate the likelihood of all the original sequences at each iteration. This leads to slower iterations, and results in a total training time about 5 times longer than that of VHEM-H3M (see Table 4.5). Second, the Character Trajectory data is more "controlled" than the CAL500 data, since each class corresponds to a single character, and all the examples are from the same writer. As a consequence, there is less variation in the intermediate HMMs (i.e., they are clustered more closely), and several of them may summarize the cluster well, providing good candidate cluster centers for PPK-SC. In conclusion, PPK-SC faces only a limited loss of information when selecting one of the initial HMMs to represent each cluster, and achieves reasonable performances.

[23] For smaller values of ΔLL (e.g., $\Delta LL < 10^{-5}$), the performance of EM-H3M did not improve.

[24] In the Character Trajectory dataset there are on average 71 training sequences per character. In CAL500, each tag is associated with *thousands* of training sequences at the song level (e.g., an average of about 8000 audio fragments per tag).

Table 4.6. Annotation and retrieval performance on CAL500 for VHEM-H3M when varying the virtual sample parameters N_v and τ.

	$N_v = 1000$				$\tau = 10$			
	$\tau = 2$	$\tau = 5$	$\tau = 10$	$\tau = 20$	$N_v = 1$	$N_v = 10$	$N_v = 100$	$N_v = 1000$
P@5	0.4656	0.4718	0.4738	0.4734	0.4775	0.4689	0.4689	0.4738
P@10	0.4437	0.4487	0.4507	0.4478	0.4534	0.4491	0.4466	0.4507
P@15	0.4236	0.4309	0.4346	0.4327	0.4313	0.4307	0.4242	0.4346

4.6.3 Robustness of VHEM-H3M to Number and Length of Virtual Samples

The generation of virtual samples in VHEM-H3M is controlled by two parameters: the number of virtual sequences (N), and their length (τ). In this section, we investigate the impact of these parameters on annotation and retrieval performance on CAL500. For a given tag t, we set $N = N_v N_t K^{(s)}$, where N_v is a constant, N_t the number of training songs for the tag, and $K^{(s)}$ the number of mixture components for each song-level H3M. Starting with $(N_v, \tau) = (1000, 10)$, each parameter is varied while keeping the other one fixed, and annotation and retrieval performance on the CAL500 dataset are calculated, as described in Section 4.6.1.

Table 4.6 presents the results, for $\tau \in \{2, 5, 10, 20\}$ and $N_v \in \{1, 10, 100, 1000\}$. The performances when varying τ are close on all metrics. For example, average P@5, P@10 and P@15 vary in small ranges (0.0082, 0.0070 and 0.0110, respectively). Similarly, varying the number of virtual sequences N_v has a limited impact on performance as well.[25] This demonstrates that VHEM-H3M is fairly robust to the choice of these parameters.

Finally, we tested VHEM-H3M for music annotation and retrieval on CAL500,

[25] Note that the E-step of the VHEM-H3M algorithm averages over all possible observations compatible with the input models, also when we choose a low value of N_v (e.g., $N_v = 1$). The number of virtual samples controls the "virtual mass" of each input HMMs and thus the certainty of cluster assignments.

using virtual sequences of the same length as the audio fragments used at the song level, i.e., $\tau = T = 125$. Compared to $\tau = 10$ (the setting used in earlier experiments), we registered an 84% increase in total running time, with no corresponding improvement in performance. Thus, in our experimental setting, making the virtual sequences relatively short positively impacts the running time, without reducing the quality of the learned models.

4.7 Conclusions

In this paper, we presented a variational HEM (VHEM) algorithm for clustering HMMs with respect to their probability distributions, while generating a novel HMM center to represent each cluster. Experimental results demonstrate the efficacy of the algorithm on various clustering, annotation, and retrieval problems involving time-series data, showing improvement over current methods. In particular, using a two-stage, *hierarchical estimation* procedure—learn H3Ms on many smaller subsets of the data, and summarize them in a more compact H3M model of the data—the VHEM-H3M algorithm estimates annotation models from data more efficiently than standard EM and also improves model robustness through better regularization. Specifically, averaging over all possible virtual samples prevents over-fitting, which can improve the generalization of the learned models. Moreover, using relatively short virtual sequences positively impacts the running time of the algorithm, without negatively affecting its performance on practical applications. In addition, we have noted that the VHEM-H3M algorithm is robust to the choice of the number and length of virtual samples.

In our experiments, we have implemented the first stage of the hierarchical estimation procedure by partitioning data in *non-overlapping* subsets (and learning an intermediate H3M on each subset). In particular, partitioning the CAL500 data at the song level had a practical advantage. Since individual songs in CAL500 are relevant

to several tags, the estimation of the song H3Ms can be executed one single time for each song in the database, and *re-used* in the VHEM estimation of all the associated tag models. This has a positive impact on computational efficiency. Depending on the particular application, however, a slightly different implementation of this first stage (of the hierarchical estimation procedure) may be better suited. For example, when estimating a H3M from a very large amount of training data, one could use a procedure that does not necessarily cover all data, inspired by [94]. If n is the size of the training data, first estimate $B > 1$ intermediate H3Ms on as many (possibly overlapping) "little" bootstrap subsamples of the data,[26] each of size $b < n$. Then summarize all the intermediate H3Ms into a final H3M using the VHEM-H3M algorithm.

In future work we plan to extend VHEM-H3M to the case where all HMMs share a large GMM universal background model for the emission distributions (with each HMM state having a different set of weights for the Gaussian components), which is commonly used in speech [82, 18, 135] or in hand-writing recognition [142]. This would allow for faster training (moving the complexity to estimating the noise background model) and would require a faster implementation of the inference (e.g., using a strategy inspired by [45]). In addition, we plan to derive a HEM algorithm for HMMs with discrete emission distributions, and compare its performance to the work presented here and to the extension with the large GMM background model.

Finally, in this work we have not addressed the model selection problem, i.e., selecting the number of reduced mixture components. Since VHEM is based on maximum likelihood principles, it is possible to apply standard statistical model selection techniques, such as Bayesian information criterion (BIC) and Akaike information criterion (AIC) [105]. Alternatively, inspired by Bayesian non-parametric statistics, the

[26] Several techniques have been proposed to bootstrap from sequences of samples, for example refer to [72].

VHEM formulation could be extended to include a Dirichlet process prior [23], with the number of components adapting to the data.

4.8 Acknowledgements

Chapter 4, in full, is a reprint of the material as it appears in the Journal of Machine Learning Research, 15, pp. 697-747, 2014, E. Coviello, A.B. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Algorithm 4. VHEM algorithm for H3Ms

Input: base H3M $\mathscr{M}^{(b)} = \{\omega_i^{(b)}, \mathscr{M}_i^{(b)}\}_{i=1}^{K^{(b)}}$, number of virtual samples N.
Initialize reduced H3M $\mathscr{M}^{(r)} = \{\omega_j^{(r)}, \mathscr{M}_j^{(r)}\}_{j=1}^{K^{(r)}}$.
repeat
  {Variational E-step}
  Compute optimal variational distributions and variational lower bounds:
  for each pair of HMMs $\mathscr{M}_i^{(s)}$ and $\mathscr{M}_j^{(r)}$
    for each pair of emission GMMs for state β of $\mathscr{M}_i^{(s)}$ and ρ of $\mathscr{M}_j^{(r)}$:
      Compute optimal variational distributions $\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)}$ as in (4.18)
      Compute optimal lower bound $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$ to expected log-likelihood as in (4.22)
    Compute optimal variational distributions for HMMs as in Appendin B
      $\hat{\phi}_1^{i,j}(\rho_1|\beta_1), \quad \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t)$ for $t = \tau,\ldots,2$
    Compute optimal lower bound $\mathscr{L}_{HMM}^{i,j}$ to expected log-likelihood as in (4.21)
  Compute optimal assignment probabilities:

$$\hat{z}_{ij} = \frac{\omega_j^{(r)} \exp(N\omega_i^{(b)} \mathscr{L}_{HMM}^{i,j})}{\sum_{j'} \omega_{j'}^{(r)} \exp(N\omega_i^{(b)} \mathscr{L}_{HMM}^{i,j'})}$$

  Compute aggregate summary statistics for each pair of HMMs $\mathscr{M}_i^{(s)}$ and $\mathscr{M}_j^{(r)}$ as in Section 4.3.3:

$$\hat{v}_1^{i,j}(\rho) = \sum_{\beta=1}^{S} v_1^{i,j}(\rho,\beta), \hat{v}^{i,j}(\rho,\beta) = \sum_{t=1}^{\tau} v_t^{i,j}(\rho,\beta),$$

$$\hat{\xi}^{i,j}(\rho,\rho') = \sum_{t=2}^{\tau}\sum_{\beta=1}^{S} \xi_t^{i,j}(\rho,\rho',\beta)$$

  {M-step}
  For each component $\mathscr{M}_j^{(r)}$, recompute parameters using (4.26)-(4.30).
until convergence
Output: reduced H3M $\{\omega_j^{(r)}, \mathscr{M}_j^{(a)}\}_{j=1}^{K^{(r)}}$.

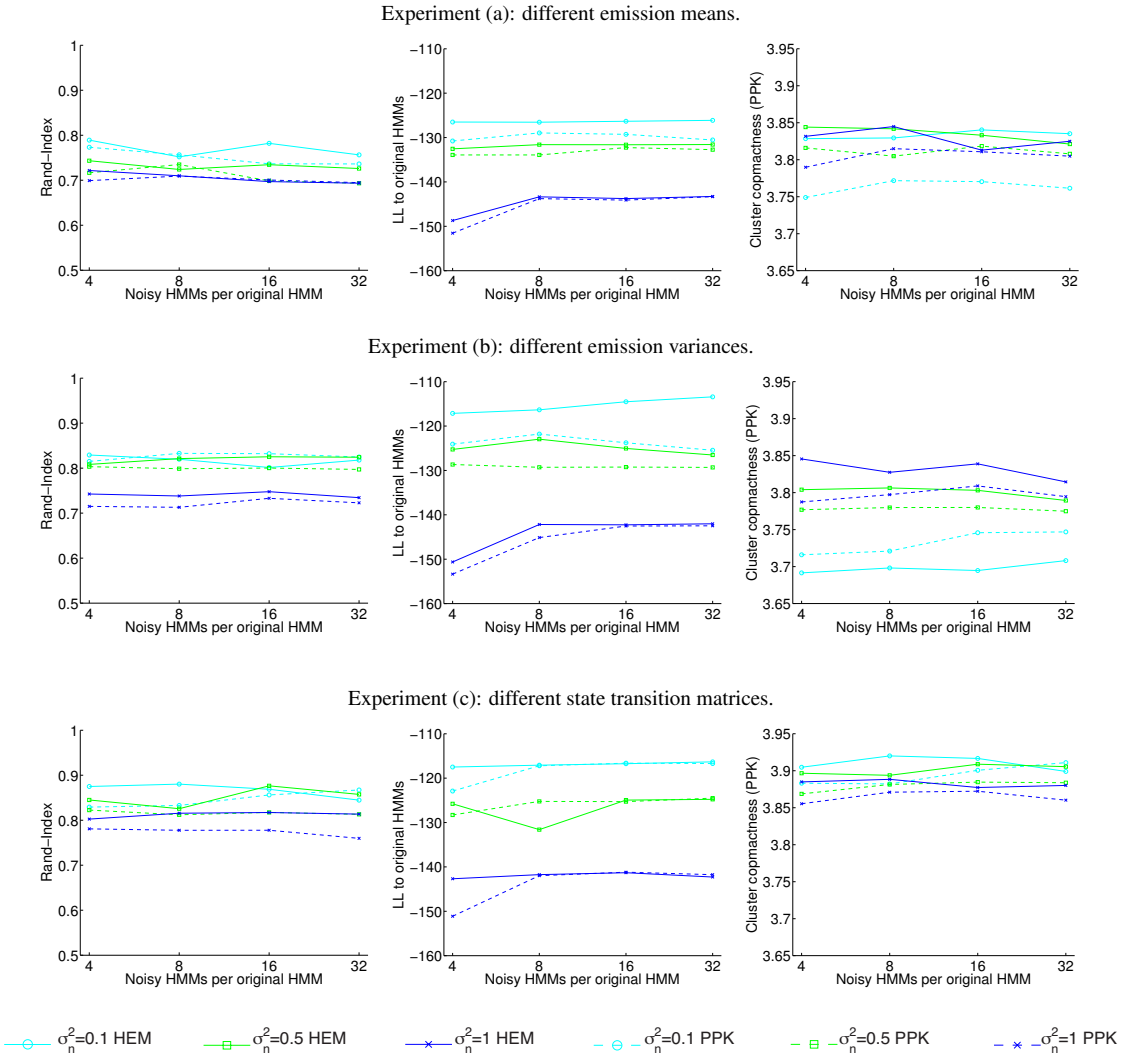


Figure 4.3. Results on clustering synthetic data with VHEM-H3M and PPK-SC. Performance is measured in terms of Rand-index, expected log-likelihood and PPK cluster-compactness.

Chapter 5

Combining Content-Based AutoTaggers with Decision-Fusion

5.1 Introduction

The recent age of music proliferation has raised the need for automatic algorithms to efficiently search and discover music. Many successful recommendation systems rely on textual metadata provided by expert musicologists or social services in the form of semantic tags – keywords or short phrases that capture relevant characteristics of music pieces, ranging from genre and instrumentation, to mood and usage. By bridging the gap between music and human semantics, tags allow semantic retrieval based on transparent textual descriptions, or query-by-example recommendation based on semantic similarity (as opposed to acoustic similarity) to a query song.

Meta-data-based methods work well in practice, provided that enough annotations are available. However, the cold start problem and the prohibitive cost of manual labour limit their applicability to large-scale applications. Therefore, the deployment of modern music recommendation systems can benefit from the development of auto-taggers, i.e., machine-learning algorithms that automatically analyze and index music with semantic tags, which can then be used to improve the search experience and speed up the discovery of desired content.

5.1.1 Previous work

Most auto-taggers are based on music content analysis and are trained from a database of annotated songs (e.g., see [159, 81, 106, 55]). After extracting a set of acoustic features from each training song, a series of statistical models are estimated, each of which capturing the characteristic acoustic patterns in the songs that are associated with one of the tags from a given vocabulary. When analyzing a new song, the auto-tagger processes the time series of acoustic features of the song and outputs a vector of tag-affinities. The affinity-vector can then be transformed into a semantic multinomial (SMN), i.e., a probability distribution characterizing the relevance of each tag to a song. A song is then annotated by selecting the top-ranking tags in its SMN, or the SMN itself can be used as a high-level descriptor, e.g., for retrieving songs based on semantic similarity. A number of discriminative (e.g., see [106, 55, 60, 167, 150, 29]) and generative (e.g., see [160, 159, 140, 81]) machine learning algorithms have been proposed to model predictive acoustic patterns in audio content based on a bag-of-features (BoF) representation, which treats audio features independently and ignores their temporal order. Recently, Coviello et al. [40] proposed to leverage dynamic texture mixture (DTM) models for auto-tagging purposes. More precisely, DTM-based auto-taggers model audio fragments (i.e., time series of audio features extracted from a few seconds of musical signal) as the output of linear dynamical systems. This approach explicitly captures temporal structures in the musical signal, whereas a BoF representation discards such dynamics.

At a higher level of abstraction, contextual approaches have focused on modeling the semantic context that drives the correlation between different tags (e.g., a song tagged with “drums” is more likely to also be tagged with “electric guitar” than “violin”). While content-based models operate on low-level acoustic features to predict semantic

multinomials, contextual models are designed to capture meaningful tag correlations in these SMNs, to reinforce accurate tag predictions while suppressing spurious ones. So, a contextual model naturally complements a content-based model, which usually treats tags independently. Combining them has been shown to improve performance. State-of-the-art solutions are based on discriminative approaches (e.g., support vector machines [117], boosting [22], ordinal regression [174]) as well as generative models (e.g., Dirichlet mixture models (DMM) [110]).

5.1.2 Original contribution

The main contribution of this paper is to propose *decision-fusion*, which uses semantic context modeling to simultaneously leverage the benefits of different content-based auto-taggers. Using two or more content-based auto-taggers that emphasize diverse aspects of the musical signal (e.g., only timbre vs. temporal dynamics), we collect alternative opinions on each song-tag association. We expect that, besides modeling the context between tags predicted from the same auto-tagger, context modeling can capture the correlations that arise between tag predictions based on different auto-taggers, leading to a more sophisticated system.

This offers a solution to the problem of selecting or combining alternative annotation models that previous work has pointed out. Coviello et al. [40], for example, noted that even though their DTM-based auto-tagger generally outperformed a BoF approach based on Gaussian mixture models (GMM), the improvements were most significant on tags with clear temporal characteristics; for some tags, in fact, the GMM-based model was still favorable (i.e., tags where "timbre says it all").

Experimental results show that decision-fusion leads to improved annotation and retrieval performance compared to i) each individual auto-tagger, ii) each individual auto-tagger in tandem with a contextual model (the "traditional" context-based approach)

and iii) various other approaches to combining multiple content-based auto-taggers, such as fixed-combination rules and the regression-based combination algorithms proposed by Tomasik et al. [157]. We note that the focus of the latter was slightly different from our work, since it investigates the combination of tags predicted from different information sources (i.e., content-based auto-tags, social tags, collaborative-filtering-based tags), rather than from different content-based auto-taggers only. In addition, as semantic context modeling is naturally complementary to any content-based auto-tagger, we corroborate the intuition that there is a benefit in combining DTM-based temporal modeling and semantic context modeling, which has not been shown before.

The remainder of this paper is organized as follows. A brief review of the automatic music tagging problem and the models used in this work are presented in Section 5.2. Section 5.3 discusses decision-fusion. Lastly, the experimental setup and results are reported in Sections 6.6 and 5.5, respectively.

5.2 Automatic music tagging

The automatic task of music tagging is widely tackled as a supervised multi-class labeling problem [28], where each class corresponds to a tag w_i of a semantic vocabulary $\mathscr{V}$ (e.g., "rock", "drum", "tender", "mellow"). The music content of a song is represented as a time series of low-level acoustic features $\mathscr{Y} = \{\mathbf{y}_1, \ldots, \mathbf{y}_T\}$, where each feature is extracted from a short snippet of the audio signal and T depends on the length of the song. The semantic content with respect to $\mathscr{V}$ is represented as an annotation vector $\mathbf{c} = (c_1, \ldots, c_{|\mathscr{V}|})$, where $c_i > 0$ only if there is a positive association between a song and the tag w_i. The goal of an auto-tagging system is to infer the relevant semantic annotations of unseen songs.

At this aim, a set of statistical models is trained to capture the patterns in the audio feature space associated with each tag in $\mathscr{V}$, from a database $\mathscr{D} = \{(\mathscr{Y}_d, \mathbf{c}_d)\}_{d=1}^{|\mathscr{D}|}$

of annotated songs. Based on the learned tag models, the auto-tagger can process the acoustic features extracted from a novel song $\mathscr{Y}$ and produce a vector of tag-affinities, which is mapped into a semantic multinomial $\pi = (\pi_1, ..., \pi_{|V|})$ lying on a semantic space (i.e., $\sum_i \pi_i = 1$ with $\pi_i \geq 0$), where $\pi_i = P(w_i|\mathscr{Y})$ represents the probability that the i^{th} tag applies to song $\mathscr{Y}$.

In order to leverage high level relationships that arise in the tag predictions of content-based auto-taggers, contextual approaches additionally introduce a second modeling layer to capture meaningful tag correlations in the SMNs. In particular, a content-based auto-tagger is used to produce a SMN π_d for each song $\mathscr{Y}_d$ in $\mathscr{D}$, while a second layer of statistical models is trained onto $\{(\pi_d, \mathbf{c}_d)\}_{d=1}^{|\mathscr{D}|}$, to capture which patterns in the SMNs are predictive for each tag. For a novel song $\mathscr{Y}$, the contextual tag models can therefore be used to refine the semantic multinomial π produced by the content-based auto-tagger.

Music annotation involves finding the tags that best describe a song; this is achieved by selecting the subset of tags that peak in its semantic multinomial. Retrieval given a one-tag query, requires ranking all songs in a database based on their relevance to the query, e.g., the corresponding entry in the semantic multinomials [159].

In the following we review a variety of content-based auto-tagging strategies, where low-level acoustic content is represented either as a bag-of-features (Sections 5.2.1 and 5.2.1) or as a time series of features (Section 5.2.1). Additionally, Section 5.2.2 introduces a contextual approach for modeling tag correlations as well.

5.2.1 Content modeling

Content-based auto-taggers have been designed to model the acoustic content associated with tags and represented as a bag-of-features using both generative and discriminative models, as in Sections 5.2.1 and 5.2.1, respectively; conversely, the use

of time series of audio features for music tagging has been considered in the generative approach of Section 5.2.1 only.

The Gaussian mixture model (GMM)

Turnbull et al. [159], proposed to capture the most prominent acoustic textures associated to each tag w_i in $\mathcal{V}$ with a probability distribution $p(\mathbf{y}|w_i)$ over the space of audio features $\mathbf{y}$, which is a Gaussian mixture model (GMM):

$$p(\mathbf{y}|w_i) = \sum_{r=1}^{R} a_r^{w_i} \mathcal{N}(\mathbf{y}|\mu_r^{w_i}, \Sigma_r^{w_i}) \,, \tag{5.1}$$

where R is the number of mixture components, $\mathcal{N}(\cdot|\mu,\Sigma)$ a multivariate Gaussian distribution with mean μ and covariance matrix Σ, and $a_r^{w_i}$ the mixing weights. The parameters $\{a_r^{w_i}, \mu_r^{w_i}, \Sigma_r^{w_i}\}_{r=1}^{R}$ of each tag model $p(\mathbf{y}|w_i)$ are *estimated* from the bag-of-features extracted from the songs in $\mathcal{D}$ that are positively associated with w_i, using the hierarchical expectation-maximization (EM) algorithm [164].

Given the audio content of a new song $\mathcal{Y} = \{\mathbf{y}_1, ..., \mathbf{y}_T\}$, the relevance of each tag w_i is computed using the Bayes rule:

$$\pi_i = P(w_i|\mathcal{Y}) = \frac{p(\mathcal{Y}|w_i)\,P(w_i)}{p(\mathcal{Y})}, \tag{5.2}$$

where $P(w_i)$ is the tag prior (assumed to be uniform) and $p(\mathcal{Y})$ the song prior, i.e., $p(\mathcal{Y}) = \sum_{j=1}^{|\mathcal{V}|} p(\mathcal{Y}|w_j)P(w_j)$. The likelihood term in (5.2) is computed as the geometric average of the individual sequence likelihoods, i.e., $p(\mathcal{Y}|w_i) = \prod_{t=1}^{T} p(\mathbf{y}_t|w_i)^{\frac{1}{T}}$.

Boosting (BST)

The boosting approach proposed by Eck et al. [55] is a supervised discriminative algorithm that learns a binary classifier for each tag w_i in the vocabulary $\mathcal{V}$, from both the

positive and the negative training examples for that tag. More specifically, it constructs a *strong classifier* which combines a set of simpler classifiers, called *weak learners*, in an iterative way. As weak learners, according to [22], we use single stumps (i.e., binary thresholding on one low-level acoustic feature).

A novel song $\mathscr{Y}$ is classified by each of the binary classifiers and Platt scaling is applied to produce a probability estimate $\pi_i = P(w_i|\mathscr{Y})$ for each tag w_i. We will refer to this approach as BST.

Temporal modeling (DTM)

Coviello et al. [40] proposed a novel auto-tagger built upon the DTM model, which explicitly captures both the timbral and the temporal structures of music that are most predictive for each tag. Specifically, the dynamic texture (DT) model [54] treats an audio fragment $\mathbf{y}_{1:\tau}$ as output of a linear dynamical system. The model consists of a double embedded stochastic process, in which a lower dimensional Gauss-Markov process x_t encodes the dynamics (evolution) of the acoustic component $\mathbf{y}_t$ over time

Each tag distribution is modeled with a dynamic texture mixture (DTM) [32] probability density over sequences of audio feature vectors:

$$p(\mathbf{y}_{1:\tau}|w_i) = \sum_{r=1}^{R} a_r^{(w_i)} p(\mathbf{y}_{1:\tau}|\Theta_r^{(w_i)}) \,, \tag{5.3}$$

where R is the number of mixtures and $\Theta_r^{(w_i)}$ is the r^{th} DT component. The parameters $\{a_r^{(w_i)}, \Theta_r^{(w_i)}\}_{r=1}^{R}$ are estimated based on the audio fragments extracted from the songs in $\mathscr{D}$ positively associated with the tag w_i, using an efficient hierarchical EM algorithm for DTM (HEM-DTM) [33].

Given the audio fragments extracted from a new song $\mathscr{Y} = \{\mathbf{y}_{1:\tau}^1, \ldots, \mathbf{y}_{1:\tau}^F\}$, where F depends on the length of the song, the relevance of tag w_i is computed using Bayes' rule

(5.2), with the likelihood computed as the geometric average of the individual sequence likelihoods smoothed by the sequence length τ, i.e., $p(\mathscr{Y}|w_i) = \prod_{t=1}^{F} p(\mathbf{y}_{1:\tau}^t|w_i)^{\frac{1}{F\tau}}$.

5.2.2 Context modeling (DMM)

As mentioned in Section 5.1.1, different approaches have been proposed to model contextual relationships in SMNs; in this work, we use the DMM [110]. The DMM is a generative model that assumes the SMNs π of the songs positively associated to a tag w_i are distributed accordingly to a mixture of Dirichlet distributions over the semantic space defined by $\mathscr{V}$:

$$p(\pi|w_i;\Omega^w) = \sum_{r=1}^{R} \beta^{w_i} \mathrm{Dir}(\pi|\alpha_r^{w_i}) \,, \tag{5.4}$$

where R is the number of mixtures, $\beta_k^{w_i}$ are the mixing weights, and $\mathrm{Dir}(\cdot|\alpha)$ is a Dirichelet distribution of parameters $\alpha = (\alpha_1, ..., \alpha_{|\mathscr{V}|})$. The parameters of the DMM for each tag w_i in $\mathscr{V}$ are estimated from the semantic multinomials extracted from the songs in $\mathscr{D}$ positively associated with the tag, via the generalized EM algorithm [137].

Hence, given a new song described by the SMN $\pi = (\pi_1, ..., \pi_{|\mathscr{V}|})$, the relevance of a tag w_i is computed using Bayes' rule to get the tag posterior probabilities in the context space:

$$\theta_i = P(w_i|\pi) = \frac{p(\pi|w_i)P(w_i)}{p(\pi)} \,. \tag{5.5}$$

All the tag posterior probabilities form the *contextual multinomial* distribution of the song, i.e., $\theta = (\theta_1, \ldots, \theta_{|\mathscr{V}|})$, which can then be used for semantic annotation and retrieval.

5.3 Decision-fusion

Each content-based auto-tagger generally emphasizes particular aspects of the musical signal. Despite some auto-taggers could be preferred over others based on average performances (Table 5.1, part (a)), the spread in performances registered on specific tags (e.g., see Figure 5.1) makes unclear if any auto-tagger may be the best. This leaves open the problem of choosing the most appropriate method for each tag, or, indeed, the one of combining different auto-taggers.

In this paper we argue that semantic context modeling can also be used as a strategy to combine different content-based auto-taggers, which we name *decision-fusion*. Indeed, by modeling the patterns that arise from the tag predictions generated by different content-based auto-taggers, decision-fusion combines all the different opinions into a single prediction and leverages the benefits of each of the acoustic characteristics emphasized by the original auto-taggers.

Formally, let us assume a group $\mathscr{A}$ of different content-based auto-tagging algorithms is available. For each song d in the database $\mathscr{D}$, semantic multinomials π_d^a for $a = 1, \ldots, |\mathscr{A}|$ are computed (i.e., one for each auto-tagger in $\mathscr{A}$) and pooled together into the aggregated semantic multinomial:

$$\pi_d^{\mathscr{A}} = (\pi_d^1, \ldots, \pi_d^{|\mathscr{A}|}) \,, \tag{5.6}$$

which is intended to be normalized to sum to 1. In practice, it is as we are now working with a new semantic vocabulary $\mathscr{V}^{\mathscr{A}} = \mathscr{V}^1 \times \cdots \times \mathscr{V}^{|\mathscr{A}|}$ of size $|\mathscr{A}| \cdot |\mathscr{V}|$, where each tag is replicated $|\mathscr{A}|$ times, one for each auto-tagger. Decision-fusion consists in training a set of semantic context models, i.e., $p(\pi^{\mathscr{A}}|w_i)$ for $w_i = 1, \ldots, |\mathscr{V}|$, over the aggregated semantic multinomials $\{(\pi_d^{\mathscr{A}}, \mathbf{c}_d)\}_{d=1}^{|\mathscr{D}|}$ to capture *both* intra- and inter-auto-taggers tag

correlations. Note that traditional context modeling acts on the SMNs of a *single* auto-tagger, thus capturing *only* intra-auto-tagger correlations.

Decision-fusion can be implemented through a variety of context-modeling algorithms. In particular, in this work we tested the DMM presented in Section 5.2.2. Therefore, the aggregated SMNs $\pi^{\mathscr{A}}$ of songs positively associated with tag w_i are assumed to be distributed accordingly to a mixture of Dirichlet distributions over the semantic space $\mathscr{V}^{\mathscr{A}}$:

$$p(\pi^{\mathscr{A}}|w_i) = \sum_{r=1}^{R} \beta^{w_i} \mathrm{Dir}(\pi^{\mathscr{A}}|\alpha_r^{w_i}) \,, \tag{5.7}$$

where $\alpha = (\alpha_1, ..., \alpha_{|\mathscr{A}|\cdot|\mathscr{V}|})$.

An unseen song $\mathscr{Y}$ is first processed by each of the content-based auto-taggers available to produce the semantic multinomials π^a for $a = 1, \ldots, |\mathscr{A}|$, which are then aggregated in $\pi^{\mathscr{A}}$. Finally, Bayes' rule as in Equation 5.5 is applied to compute the posteriors $\theta_i^{\mathscr{A}} = p(w_i|\pi^{\mathscr{A}})$ for each tag w_i, and to form a *decision-fusing multinomial* $\theta^{\mathscr{A}} = (\theta_1^{\mathscr{A}}, \ldots, \theta_{|\mathscr{V}|}^{\mathscr{A}})$.

5.4 Experimental setup

5.4.1 Dataset

In our experiments, we used the CAL500 dataset [159], which consists of 502 popular Western songs by as many different artists. The CAL500 dataset provides binary annotations, which are 1 when a tag applies to the song and 0 otherwise, based on the opinions of human annotators. To accurately fit the experimental models, we restrict ourselves to the subset of 97 tags that have at least 30 songs positively associated with them (11 genre, 14 instrument, 25 acoustic quality, 6 vocal characteristics, 35 emotion and 6 usage tags).

5.4.2 Audio features

The acoustic content of each song in the collection is represented by computing a time series of 34-bin Mel-frequency spectral features [135], extracted over half-overlapping windows of 92 ms of audio signal. For the auto-tagger based on the DTM, Mel-frequency spectral features are grouped into fragments of approximately 6 s. (with 80% overlap), which corresponds to $\tau = 125$ consecutive feature vectors. For the auto-tagger based on the GMM, the Mel-frequency spectral features are decorrelated using the DCT, and the resulting first 13 Mel-frequency cepstral coefficients are augmented with first and second derivatives (MFCC-deltas). Lastly, for the auto-tagger based on boosting, first and second order statistics of the MFCC deltas are computed every 5 s., in order to reduce the computational burden [55] .

5.4.3 Evaluation

In our experiments, we consider the models reviewed in Section 5.2.1, which are the content-based auto-taggers referred as GMM, BST, and DTM, and the semantic context modeling based on the DMM. We obtained the authors' code to run each algorithm. We study model combination via decision-fusion using the DMM and investigate all the possible combinations among the content-based auto-taggers considered. For instance, when combining all the three auto-taggers (i.e., when $\mathscr{A} = \{\text{GMM}, \text{BST}, \text{DTM}\}$) Equation 5.7 acts on the aggregated semantic multinomials defined as:

$$\pi_d^{\mathscr{A}} \;=\; (\pi_d^{\text{GMM}},\; \pi_d^{\text{BST}},\; \pi_d^{\text{DTM}})\;. \tag{5.8}$$

To investigate the advantages of model combination via decision-fusion, we compared its performances to a variety of combination techniques, such as fixed-combination rules [93] and trained-combiners based on regression[157], all of which are applied on the

outputs of the different content-based auto-taggers (i.e., GMM, BST, DTM). We tested different fixed-combination rules (i.e., sum, product, arithmetic average, minimum and maximum rule) in preliminary experiments, with the sum rule ($\sum$ rule) being the best. So, for example, when $\sum$ rule combines GMM, BST and DTM summing the corresponding SMNs, the final semantic multinomial of each song s is:

$$\pi_s^{\text{SUM}} = \pi_s^{\text{GMM}} + \pi_s^{\text{BST}} + \pi_s^{\text{DTM}} , \tag{5.9}$$

which is intended to be normalized to 1.

Additionally, we implemented the trained-combiner based on linear regression (LinReg), which Tomasik et. al [157] showed to outperform alternative regression techniques. In particular, we use LinReg to learn, on a tag-by-tag bases, the optimal coefficients to combining different auto-taggers to predict a ground truth of annotated songs. We refer the reader to Section 3.3 of [157] for more details on this strategy.

Annotation and retrieval performances are measured following [159]. Test set songs are annotated with the 10 most likely tags in their SMNs, and annotation accuracy is reported by computing precision, recall and F-score for each tag. Retrieval performance are evaluated with respect to each one-tag query in our vocabulary; we report mean average precision (MAP), area under the receiver operating characteristic curve (AROC) and top-10 precision (P10). All metrics are averaged over all tags and are intended to be result of 5 fold cross validation, where each song appeared in the test set exactly once.

5.5 Results

Annotation and retrieval results are presented in Table 5.1. Results for (a) individual auto-taggers are in the first block of the table, results for (b) standard contextual approaches are in the second block, and results for (c) content-based auto-tagger combi-

Table 5.1. Annotation and retrieval for the different models on the CAL500 dataset. The best results for each scenario are indicated in bold.

	retrieval			**annotation**		
Model	MAP	AROC	P10	P	R	F-score
GMM	0.417	0.686	0.425	0.374	0.205	0.213
BST	0.432	0.701	0.453	0.334	0.144	0.170
DTM	**0.446**	**0.708**	**0.460**	**0.446**	**0.217**	**0.264**
***(a)** content-based auto-taggers*						
GMM	0.447	0.711	0.465	0.436	**0.238**	0.253
BST	0.457	0.711	0.476	0.424	0.201	0.241
DTM	**0.464**	**0.723**	**0.480**	**0.461**	0.236	**0.275**
***(b)** context-modeling with DMM*						
two BoF models $\mathscr{A}$ = (GMM, BST)						
$\sum$ rule	0.440	0.709	0.463	0.369	0.153	0.185
LinReg [157]	0.444	0.708	0.459	0.371	**0.239**	0.226
context fusion	**0.460**	**0.719**	**0.475**	**0.425**	0.224	**0.255**
a BoF and a time-series model $\mathscr{A}$ = (BST, DTM)						
$\sum$ rule	0.454	0.721	0.475	0.385	0.156	0.189
LinReg [157]	0.445	0.711	0.457	0.388	**0.237**	0.228
context fusion	**0.475**	**0.729**	**0.495**	**0.434**	0.221	**0.265**
a BoF and a time-series model $\mathscr{A}$ = (GMM, DTM)						
$\sum$ rule	0.461	0.726	0.474	0.445	0.229	0.267
LinReg[157]	0.456	0.722	0.460	0.360	**0.248**	0.222
context fusion	**0.470**	**0.730**	**0.487**	**0.484**	0.230	**0.291**
two BoF and a time-series model $\mathscr{A}$ = (GMM, BST, DTM)						
$\sum$ rule	0.457	0.725	0.478	0.39	0.163	0.202
LinReg[157]	0.452	0.715	0.465	0.384	**0.242**	0.232
context fusion	**0.475**	**0.731**	**0.496**	**0.456**	0.217	**0.270**
***(c)** auto-tagger combination*						

nation are in the last four blocks.

First, we notice that for each combination of the content-based auto-taggers considered, decision-fusion outperforms all the other combination techniques, except in recall, where LinReg is generally the best one. Second, differently from $\sum$ rule and LinReg, decision-fusion always improves with respect to the original content-based

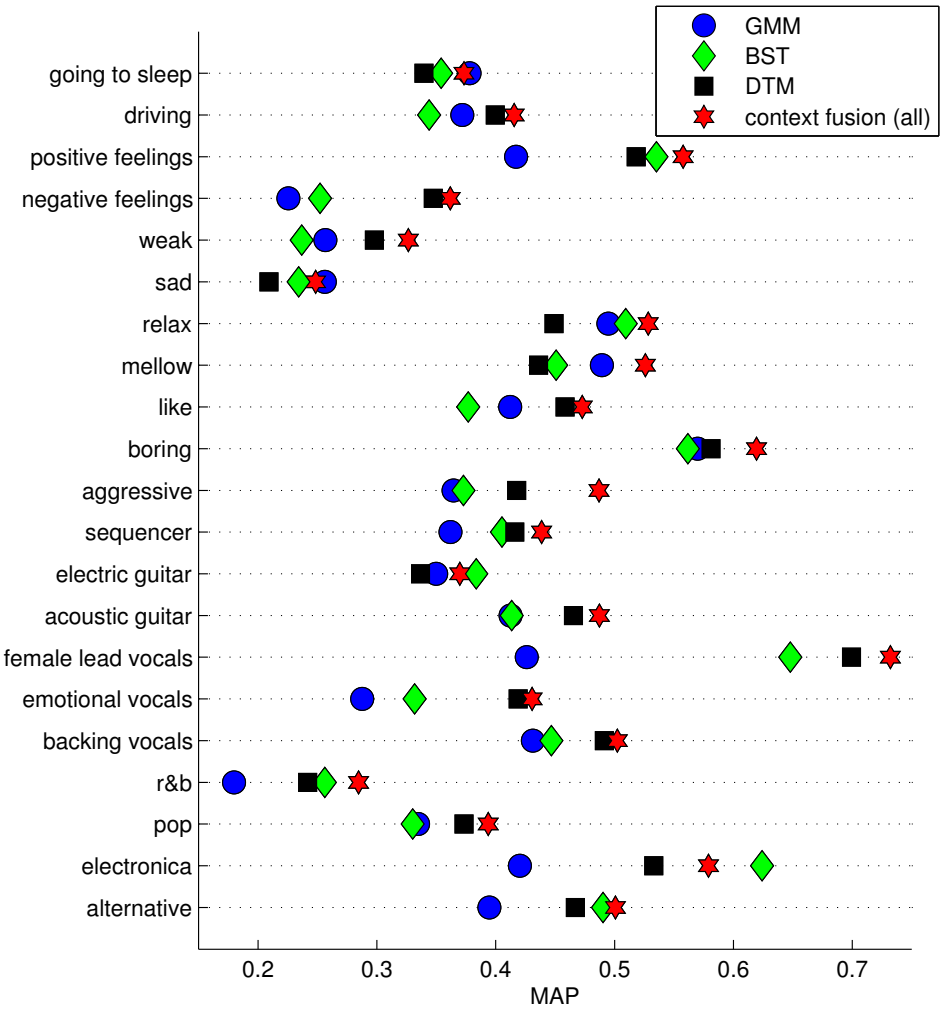


Figure 5.1. Retrieval performance (MAP) for a subset of the CAL500 vocabulary for GMM, BST, DTM, and decision-fusion of GMM, BST and DTM. Among the content-based auto-tagger, each one appears to be best on a subset of tags. However, decision-fusion is superior on the majority of tags.

auto-taggers combined.

Decision-fusion performs better by capturing the correlations that arise between tag predictions based on different auto-taggers and, consequently, by indirectly leveraging various aspects of the musical signal emphasized by each of those auto-taggers.

Indeed, decision-fusion of BoF auto-taggers with the DTM has major benefits, as it takes advantage of predictions that are based on different fundamentals, i.e., timbre and temporal dynamics vs. only timbre. On the other hand, decision-fusion of GMM and BST, which both model only the timbre, does not achieve comparable improvements over the corresponding standard context-models. In addition, the combination of all three auto-taggers with decision-fusion leads to the best retrieval performance; yet the modest improvements over the combination of BST and DTM in retrieval are compensated by improvements in precision and F-score over the same method.

Figure 5.1 depicts the MAP score achieved by a subset of tags, for the content-based auto-taggers (i.e., GMM, BST, DTM) and for decision-fusion using GMM, BST and DTM. Even if DTM could be preferred over both GMM and BST based on the *average* performances reported in Table 5.1, the *fluctuation* in performance on specific tags shown in Figure 5.1 suggests that each content-based auto-tagger may be better suited for a subset of the tags than the others. However, leveraging a rich contextual information that benefits from various acoustic characteristics of the musical signal, decision-fusion using GMM, BST and DTM performs best on the majority of all the tags reported.

Finally, part (b) of Table 5.1 also reports that standard context modeling always improves over the individual performance of the original content-based auto-taggers. While Miotto et al. [110] already showed this for the BoF models (i.e., GMM and BST), we have demonstrated that it holds true for the DTM as well.

5.6 Conclusions

In this paper we have proposed *decision-fusion* as a strategy for combining different content-based auto-taggers. It uses semantic context modeling to simultaneously leverage the benefits of different content-based auto-taggers. Experimental results

demonstrate especially that it achieves better annotation and retrieval performance than individual auto-taggers and various other techniques to combining multiple content-based auto-taggers.

5.7 Acknowledgements

Chapter 5, in full, is a reprint of the material as it appears in the Proceedings of the 12th International Society for Music Information Retrieval Conference, ISMIR 2011, Miami (USA). 24-28 October 2011, E. Coviello, R. Miotto, and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 6

A Bag of Systems Representation for Music Auto-tagging

6.1 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 [165, 37] or social services [101], 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))[97]. 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.* [159] 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. [42] 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 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 [42], Coviello *et al.* reported that while the DTM models they propose outperformed GMM models (as proposed by Turnbull *et. al.* in [159]) 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 [5] 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 [56], 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 6.2 we overview related work. Then in Section 6.3 we delve into the details of the BoS representation of music, including the generative models used as codewords (6.3.1), and algorithms for learning a BoS codebook (6.3.2) and for representing songs using the resulting BoS codebook (6.3.3). Music annotation and retrieval with BoS histograms is discussed in Section 6.4. After introducing the music datasets used in our experiments in Section 6.5,

we present experiments that demonstrate the effectiveness of the BoS representation for music auto-tagging in Section 6.6.

6.2 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) [160, 159], hidden Markov models (HMMs) [126, 140, 41, 43], multi variate autoregressive models [47], hierarchical Dirichlet processes (HDPs) [80], a codeword Bernoulli average model (CBA) [81] and dynamic texture mixture models (DTMs) [42]. Previously proposed auto-taggers that rely on a discriminative framework employ algorithms such as multiple-instance learning [106], stacked SVMs [117], boosting [55], nearest-neighbor [126], logistic regression [172], locally-sensitive hashing [29] and temporal pooling with multilayer perceptrons [73]. In [127], 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 [52] and a multimodal music recommendation algorithm that combines information from music content, music context, and user context [145].

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 [61]. Another approach integrates features computed from shorter frames over the duration of a longer analysis window, before applying a classifier [87]. Additionally, Sturm *et. al.* [153] found that combining MFCCs over multiple time

scales to create a unified feature vector improved performance in automatic musical instrument identification. Multi-scale spectrotemporal features are used to discriminate speech from non-speech audio in [109]. In [99], the authors use an unsupervised deep convolutional representation for genre recognition and artist identification, demonstrating an improvement over raw MFCC features. In [8], scattering representations of MFCCs were used for genre classification. An approach to combine several auto-taggers via semantic context modeling is proposed in [44].

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 [81, 172]. 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 [64]. 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) [85].

A BoS approach has been used for the classification of videos [139, 33, 45], and a similar idea has inspired anchor modeling for speaker identification [152].

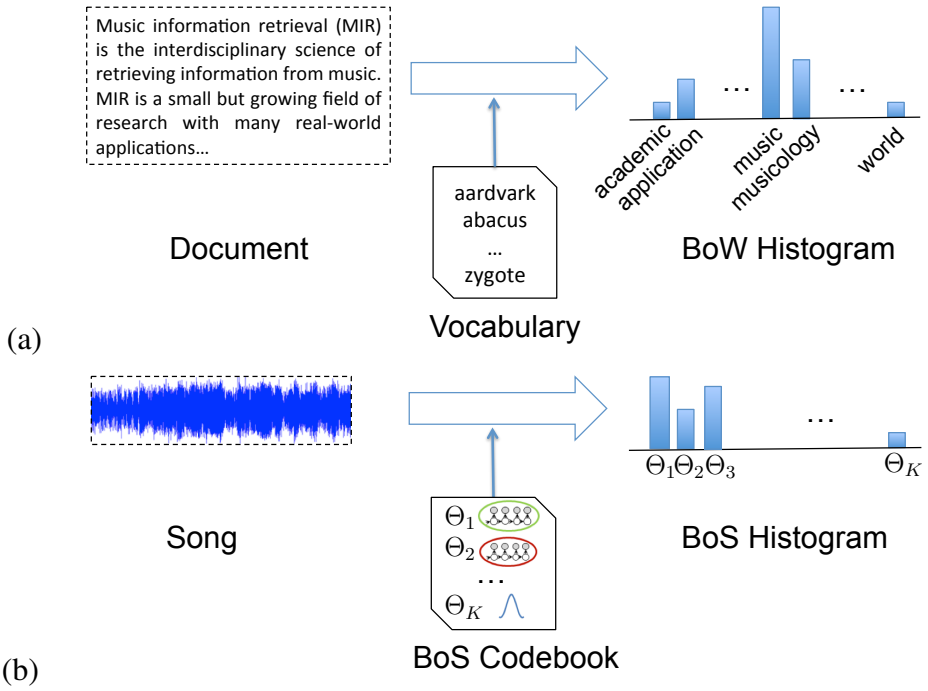


Figure 6.1. (a) Bag of Words modeling process: a document is represented as a histogram over word counts. (b) Bag of Systems modeling process: a song is represented as a histogram over musical codewords, where each codeword is a generative model that captures timbral and temporal characteristics of music.

6.3 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 Figure 6.1).

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 6.3.1).

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 6.3.2). Finally, we describe how to map songs to the BoS codebook using probabilistic inference (Section 6.3.3).

6.3.1 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 [54]. 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) $\mathscr{Y} = \{y_1, \ldots, y_T\}$, 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 $\mathscr{Y}$; DT codewords will model characteristic temporal dynamics in *ordered* subsequences of $\mathscr{Y}$. 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.

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 $\mathscr{N}(\mu, \Sigma)$, where μ and Σ are the

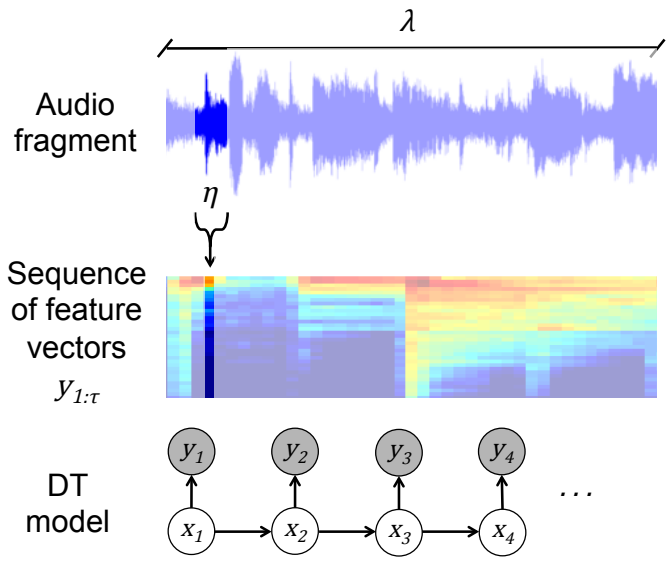


Figure 6.2. A dynamic texture model represents a fragment of audio of length λ. We first compute a sequence of τ feature vectors, each computed from a window of length η. The DT models each feature vector as an observed variable y_t in a linear dynamical system.

mean and the covariance matrix, respectively. Hence the codeword is parameterized by $\Theta = \{\mu, \Sigma\}$.

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.

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

$$x_t = Ax_{t-1} + v_t, \tag{6.1}$$

$$y_t = Cx_t + w_t + \bar{y}, \tag{6.2}$$

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.

Figure 6.2 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 $\lambda = \frac{\eta}{2}(\tau + 1)$, since we compute feature vectors from half-overlapping windows.

6.3.2 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

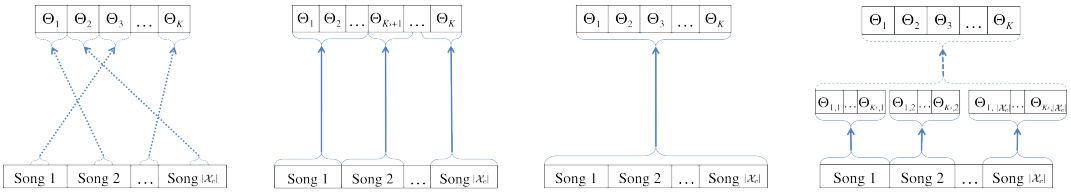


Figure 6.3. Learning a BoS Codebook $\mathscr{V} = \{\Theta_i\}_{i=1}^{K}$ by (a) modeling random audio fragments with a single instance of the model, (b) modeling each song with a small mixture model, (c) modeling all available audio data with a large mixture model using standard EM, or (d) using a 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 6.3.2).

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 $\mathscr{V}_m = \{\Theta_i\}_{i=1}^{K_m}$, where K_m is the number of codewords derived from base model m. Hence for a given song $\mathscr{Y}$, the m^{th} subset of codewords is associated with a particular feature vector representation $\mathscr{Y}_m$ of the song at the specified time scale. For codeword subset m with a Gaussian base model, each codeword $\Theta \in \mathscr{V}_m$ takes the form $\Theta = \{\mu, \Sigma\}$, and song $\mathscr{Y}$ is represented as a bag of audio feature vectors $\mathscr{Y}_m = \{y_1, \ldots, y_{T_m}\}$, extracted from windows of length η_m. For codeword subset n with a DT base model, each codeword $\Theta \in \mathscr{V}_n$ takes the form $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, and song $\mathscr{Y}$ is represented as a bag of audio fragments $\mathscr{Y}_n = \{y_{1:\tau_n}^1, \ldots, y_{1:\tau_n}^{T_n}\}$. 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., $K_1 = K_2 = \cdots = K_M = \tilde{K}$. The union of all these subsets of codewords is the BoS codebook, $\mathscr{V} = \bigcup_{m=1}^{M} \mathscr{V}_m = \{\Theta_i\}_{i=1}^{K}$, 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 $\mathscr{X}_c$. 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 $\mathscr{Y}_m$ for each song $\mathscr{Y} \in \mathscr{X}_c$, 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 $\mathscr{X}_c$. 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 $\mathscr{X}_c$.

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 $\mathscr{X}_c$), (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 $\mathscr{X}_c$), (3) learn *all* $\tilde{K}$ codewords simultaneously from *all* available audio data in $\mathscr{X}_c$, 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 Figure 6.3. In Section 6.6.1 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 repeat the procedure for each base model and pool the resulting codeword subsets.

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 [54]. 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 Figure 6.3(a)). Additionally, since each codeword is learned independently, the learning processes can easily be parallelized, reducing the complexity to $O(\frac{\tilde{K}}{N_p})$, where N_p is the number of processors available.

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 [49] or EM-DTM [32] 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 $\mathscr{V}_m = \{\Theta_i\}_{i=1}^{\tilde{K}}$, where $\tilde{K} = K_s|\mathscr{X}_c|$.

This approach has a complexity of $O(dFK_s|\mathscr{X}_c|) = O(dF\tilde{K})$, 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 Figure 6.3(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 $O(\frac{dFK_s|\mathscr{X}_c|}{N_p})$, where again N_p is the number of processors available.

Collection-based Procedure

All the codewords in $\mathscr{V}_m$ are learned *jointly*, as one large mixture model, from all available audio data in the codebook song set $\mathscr{X}_c$. In particular, for Gaussian codewords, we pool the bag-of-features representations of all the songs in $\mathscr{X}_c$, 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

[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.

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 $\mathscr{X}_c$, 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 $\mathscr{V}_m$ (mixture weights are discarded). Therefore, the number of mixture components corresponds to $\tilde{K}$ (the desired number of codewords in $\mathscr{V}_m$). The computational complexity of this procedure is $O(dN\tilde{K}) = O(dF|\mathscr{X}_c|\tilde{K})$, where $N = F|\mathscr{X}_c|$ is the number of samples used as input to the EM algorithm. This approach has a factor of $|\mathscr{X}_c|$ 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 $N < F|\mathscr{X}_c|$. 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. Figure 6.3(c) shows the collection based procedure.

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 $\mathscr{X}_c$, 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) [164, 33]. Specifically, a large collection of codewords $\mathscr{V}_m^{(b)}$ of size $\tilde{K}^{(b)}$ is learned from $\mathscr{X}_c$ with the song-based procedure, using K_s' mixture components per song. Then, the $\tilde{K}^{(b)} = K_s'|\mathscr{X}_c|$ codewords in $\mathscr{V}_m^{(b)}$ are clustered to form a codeword

set $\mathscr{V}_m$ of size $\tilde{K} < \tilde{K}^{(b)}$ using the HEM algorithm for GMMs [164] or the HEM algorithm for DTMs [33]. In this case, the HEM algorithm takes as input a large mixture model $\{\frac{1}{\tilde{K}^{(b)}}, \Theta_i^{(b)}\}_{i=1}^{\tilde{K}^{(b)}}$ (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 $O(dFK'_s|\mathscr{X}_c| + d'\tilde{K}^{(b)}\tilde{K})$, where d' is the average number of iterations of the HEM algorithm. Assuming $F \approx \tilde{K}$ and $d' \approx d$, this becomes $O(d'|\mathscr{X}_c|(F+\tilde{K})K'_s)$, 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 $O(\frac{dFK'_s|\mathscr{X}_c|}{N_p} + d'\tilde{K}^{(b)}\tilde{K})$. As before, N_p is the number of processors available. Figure 6.3(d) shows the two stages of this method of codebook generation.

6.3.3 Representing Songs with the Codebook

Once a codebook is available, a song $\mathscr{Y}$ 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 $\Theta_i \in \mathscr{V}_m$ in the song, we start from the appropriate song representation, $\mathscr{Y}_m$, 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 $\Theta \in \mathscr{V}_m$ (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 $\mathscr{V}_m$ given the audio data $\mathbf{y}^t$, i.e., $\Theta_i = \text{argmax}_{\Theta \in \mathscr{V}_m} P(\mathbf{y}^t|\Theta)$. 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 $\frac{\eta_m}{2}$.

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

$$b[i] = \frac{1}{M|\mathscr{Y}_m|} \sum_{\mathbf{y}^t \in \mathscr{Y}_m} \mathbb{1}[\Theta_i = \operatorname*{argmax}_{\Theta \in \mathscr{V}_m} P(\mathbf{y}^t|\Theta)]. \tag{6.3}$$

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

Equation 6.3 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 6.3 to support the assignment of multiple codewords at each point in the song. We introduce a smoothing parameter $k \in \{1, 2, \ldots, |\mathscr{V}_m|\}$. In particular, we assign the k most likely codewords (within one subset) to each portion of audio. The softened histogram is then constructed as:

$$b[i] = \frac{1}{M|\mathscr{Y}_m|} \sum_{\mathbf{y}^t \in \mathscr{Y}_m} \frac{1}{k} \mathbb{1}[\Theta_i \in \operatorname*{argmax}^k_{\Theta \in \mathscr{V}_m} P(\mathbf{y}^t|\Theta)], \tag{6.4}$$

where the additional normalization factor of $1/k$ ensures that b is still a valid multinomial for $k > 1$. The argmax^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 $\forall z \in Y, f(z) \geq f(x)\}$.

6.4 Music Annotation and Retrieval using the Bag-of-Systems representation

Once a BoS codebook $\mathscr{V}$ 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.[2]

6.4.1 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

[2] A BoS representation could also be used in multi-modal problems [138, 131].

consists of rank ordering a set of songs $\mathscr{S} = \{s_1, s_2 \ldots s_R\}$ according to their relevance to a query. When the query is a single tag w_i from $\mathscr{T}$, 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.

6.4.2 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).

Codeword Bernoulli Average

The CBA model proposed by Hoffman et. al. [81] 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.,

$$p(z_{ji} = l|\mathbf{b}_j) = b_j[l]. \tag{6.5}$$

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:

$$\begin{aligned} p(y_{ji} = 1|z_{ji},\beta) &= \beta_{z_{ji}i}, \\ p(y_{ji} = 0|z_{ji},\beta) &= 1 - \beta_{z_{ji}i}. \end{aligned} \tag{6.6}$$

We use the author's code [81] to fit the CBA model, i.e., learn the parameters β. 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.

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:

$$P(w_i|\mathbf{b}, \beta_i, \alpha_i) = \frac{1}{1 + \exp(\beta_i^T \mathbf{b} + \alpha_i)}. \tag{6.7}$$

Kernel logistic regression [74] 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 [154], which is defined by the kernel function:

$$\kappa(\mathbf{a}, \mathbf{b}) = \sum_j \min(a[j], b[j]), \tag{6.8}$$

where $\mathbf{a}$ and $\mathbf{b}$ are BoS histograms. In our implementation we use the software package Liblinear [58] 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.

6.5 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.

6.5.1 CAL500 Dataset

The CAL500 [159] 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.

6.5.2 CAL10K Dataset

The CAL10K dataset [156] 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.

6.5.3 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 [159, 42], 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 miliseconds to while codewords

Table 6.1. 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

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 [135]. For the DT codewords, we model 34-bin Mel-frequency spectral features.[3] 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 6.3.1 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 different basis functions (matrices) can be learned to best represent the particular codeword, and different codewords can focus

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

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 6.1. 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.

6.6 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.* [159] for a detailed definition of these metrics.

6.6.1 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 6.1), and (3) logistic regression to annotate

Table 6.2. 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

BoS histograms. Appendix B.1 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.

EXP-1: Codebook Generation Methods

We first compare the four methods of codebook generation presented in section 6.3.2. 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 B.1.

Annotation and retrieval results for these experiments are reported in Table 6.2. 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 6.2 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 B.1 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.

EXP-2: Codebook Size

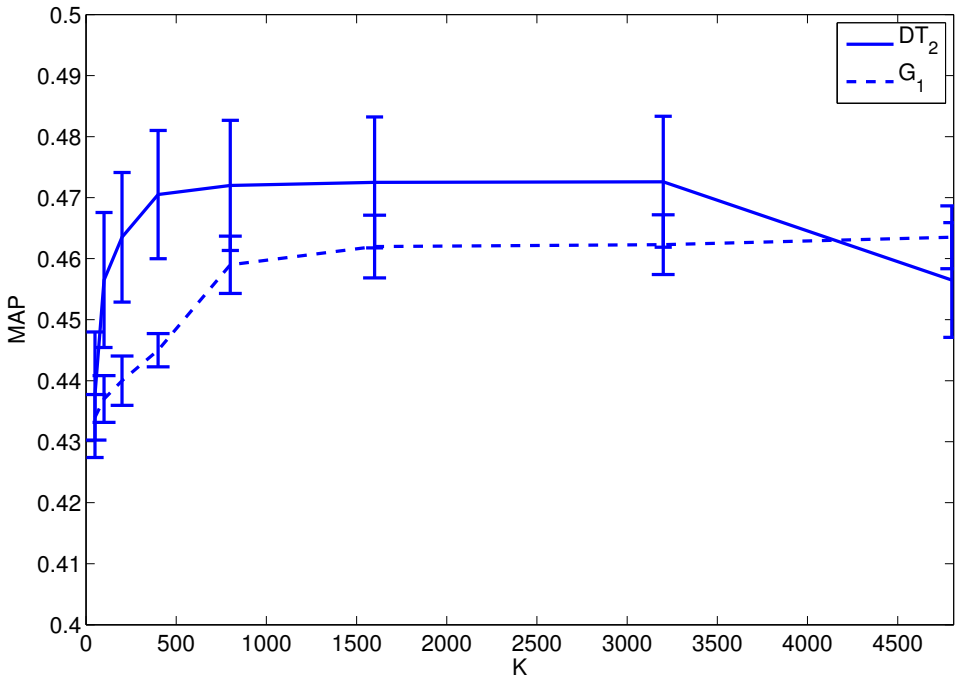


Figure 6.4. EXP-2: Retrieval performance (MAP) as a function of BoS codebook size K. Each codebook is generated with a song-based approach. For a fixed codebook song set $\mathscr{X}_c$, the codebook size is varied by varying the number of codewords learned per song, K_s. The error bars indicate the standard error over the five cross-validation folds of CAL500.

In this experiment we investigate the effect of the codebook size, K, on annotation and retrieval performance. Figure 6.4 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 B.1. 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 B.1). We also note that these results are based on a fixed codebook song set size $|\mathscr{X}_c| = 400$. 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 6.6.2).

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 6.1, 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

Table 6.3. 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

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 B.1.

Results are presented in Table B.2. 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[4] (i.e., G_1) or DT codewords on a shorter time scale[5] (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.[6] While Figure 6.4 shows that adding more codewords of the *same* base model to a codebook of size 1600 saturates and eventually deteriorates performance, Table B.2 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

[4]Paired t-test of MAP score on 97 tags: $t(96) = 6.43, p < .001$
[5]Paired t-test of MAP score on 97 tags: $t(96) = 9.08, p < .001$
[6]Paired t-test of MAP scores on 97 tags: $t(96) = 5.38, p < .001$

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. [7]

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.

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. Figure 6.5 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

[7] See Chan and Vasconcelos [32] for more details.

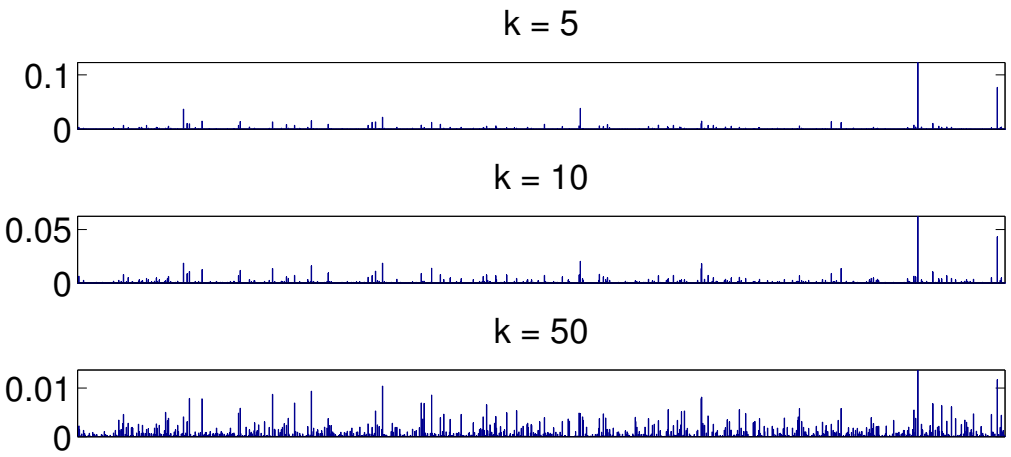


Figure 6.5. BoS histograms with smoothing parameter $k = 5, 10$ and 50 for Guns N' Roses' *November Rain*. The smoothing parameter k determines the number of codewords to be assigned at each point in the song, so using a smaller value of k will assign only a few different codewords, leading to more peaked histograms, while a larger value of k creates smoother histograms. Note the different y-axis scales.

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 B.1. Results are plotted in Figure 6.6 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.

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.

6.6.2 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

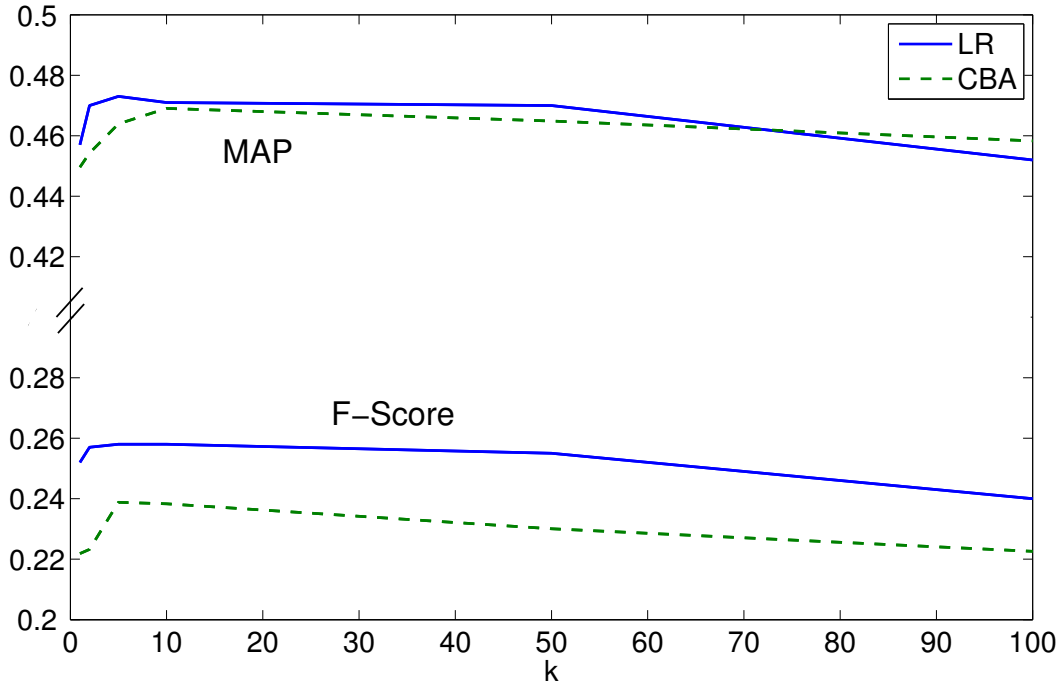


Figure 6.6. EXP-4: Annotation (F-score) and retrieval (MAP) performance of the BoS approach, using LR on the CAL500 dataset, as a function of the smoothing parameter k.

the CAL500 and CAL10K datasets.

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 ($K_s = 4, |\mathscr{X}_c| = 400$) 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., $\mathscr{X}_c = \mathscr{X}_t$).

We compare our BoS-CBA and BoS-LR auto-taggers with three state-of-the-

Table 6.4. 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**

art baselines for automatic music tagging. The first two are based on hierarchically trained GMMs (HEM-GMM) [159] and hierarchically trained DTMs (HEM-DTM) [42], 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 [159, 42].

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 [172]. 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 6.4. 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 Figure 6.7. 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.

In order to provide a qualititative sense of how the BoS autotagger performs for retrieval, Table 6.5 shows the top-10 retrieval results for the query “female lead vocals”, for the three autotaggers HEM-GMM, HEM-DTM and BoS-LR.

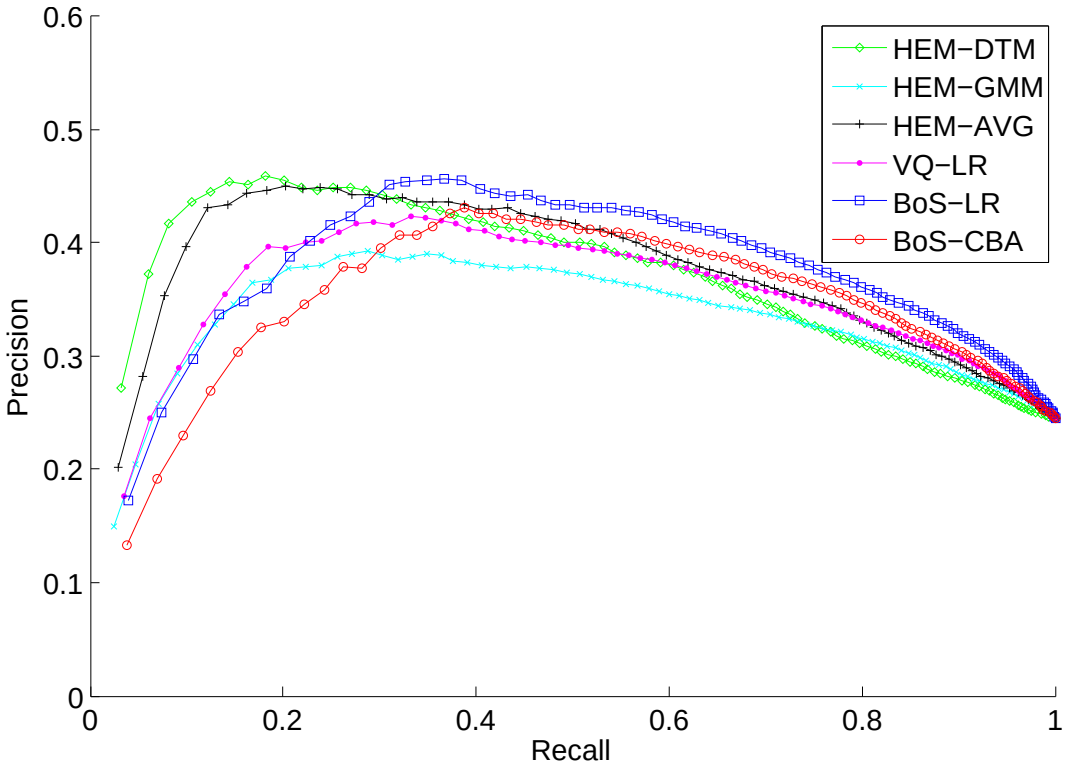


Figure 6.7. Precision-recall curves for various auto-taggers on CAL500. HEM-DTM dominates at low recall, but BoS outperforms HEM-DTM at higher recall.

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 $\mathscr{X}_c$ consisting of one song from each artist (i.e., $|\mathscr{X}_c| = 4,597$), 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 6.6.2), 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.

Annotation and retrieval results reported in Table 6.6 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).

6.7 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 6.7.1), that combining multiple types and time scales of generative models improves performance (6.7.2) and that incorporating unlabeled songs when building a codebook is advantageous (6.7.3).

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

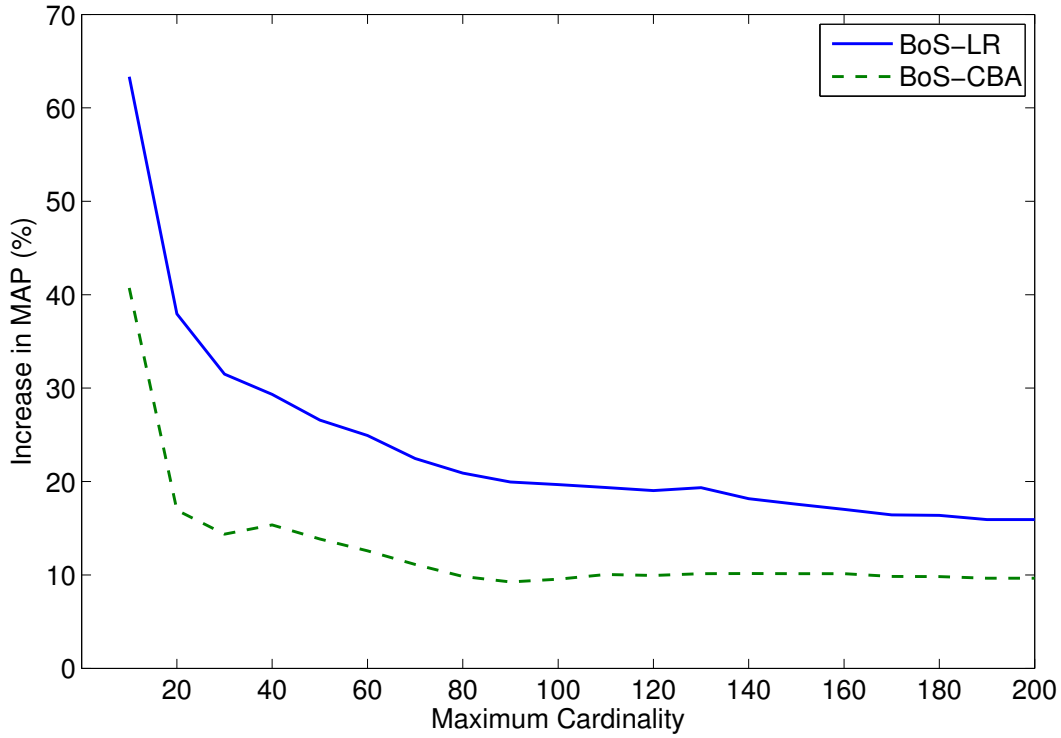


Figure 6.8. Retrieval performance (MAP) of the BoS approach, relative to HEM-DTM, as a function of maximum tag cardinality. For each point in the graph, the set of all CAL500 tags is restricted to those associated with a number of songs that is at most the abscissa value. Experiments are run on the resulting (reduced) data set as described in Section 6.6.2, and the performance metrics are averaged over each tag subset.

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 6.6.2. In Figure 6.8 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.

6.7.2 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 [42], 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 6.7 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 signicant temporal structure, such as vocal characteristics and emotions such as “angry” and “agressive”. 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 726ms, at a resolution of 12ms per feature vector, while DT_2 models fragments of 5.8s, at a resolution of 93ms 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 engery" 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 indvidual base model codebook.

6.7.3 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 $\mathscr{X}_c$.

We start by forming three different codebook song sets by aggregating subsets of CAL10K and CAL500: (1) $\mathscr{X}_c =$ C400, the codebook song set corresponding to the training set for each cross-validation split of CAL500 (as in Section 6.6.2), (2) $\mathscr{X}_c =$ R400, a codebook song set that is a subset of 400 randomly selected songs from the CAL10K collection and (3) $\mathscr{X}_c =$ 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 6.6.2). Note that, since we are using only $\mathscr{X}_t =$ C400 as the training set in these experiments, the CAL10K 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 B.3. Annotation and retrieval results are reported in Table 6.8. 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 C400 $\cup$ A5K in Table 6.8. Adding only a small set of unlabeled music to the codebook song set (i.e., C400 $\cup$ R400) does not lead to substantial improvements over using only the training set.

We have shown before (in Section B.1) 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 6.6.2, 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.

6.8 Conclusions

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.

6.9 Acknowledgements

Chapter 6, in full, is a reprint of the material as it appears in the IEEE Transactions on Audio, Speech and Language Processing, 21(9):2554-2569, December 2013, K. Ellis, E. Coviello, A. Chan and G. Lanckriet. The dissertation author was a co-author of this paper.

Table 6.5. 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"**

Table 6.6. 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**

Table 6.7. 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	$DT_{1,2}$,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

Table 6.8. 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 ($\mathscr{X}_c$)	Codebook Size (K)	**Annotation** P	R	F-Score	**Retrieval** 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

Chapter 7

Growing a Bag of Systems Tree for Fast and Accurate Annotation of Music and Videos

7.1 Introduction

The bag-of-system (BoS) representation [139], a *high-level* descriptor of motion in a video, has seen promising results in video texture classification [33]. The BoS representation of videos is analogous to the bag-of-words representation of text documents, where documents are represented by counting the occurrences of each word, or the bag-of-visual-words representation of images, where images are represented by counting the occurrences of visual codewords in the image. Specifically, in the BoS framework the codebook is formed by generative time-series models instead of words, each of them compactly characterizing typical textures and dynamics patterns of pixels or low-level features in a spatio-temporal patch. Hence, each video is represented by a BoS histogram with respect to the codebook, by assigning individual spatio-temporal patches to the most likely codeword, and then counting the frequency with which each codeword is selected. An advantage of the BoS approach is that it decouples modeling content from modeling classes. As a consequence, a codebook of sophisticated generative models can be robustly compiled from a large collection of videos, while simpler models, based on

standard text mining algorithms, are used to capture statistical regularities in the BoS histograms representing the subsets of videos associated to each individual class.

The BoS representation was originally proposed for video texture classification [139], where a codebook of dynamic texture (DT) models was used to quantize prototypical patterns in spatio-temporal cubes corresponding to interest points in videos, and was proven superior to standard methods based on modeling each video with a single DT model [1]. The BoS representation is not limited to video, but is also applicable as a descriptor to any type of time-series data. In particular, the BoS framework has also proven highly promising in automatic *music* annotation and retrieval [56], registering significant improvements with respect to current state of the art systems.

In practice, the efficacy of the BoS descriptor (or any bag-of-words representation) depends on the richness of the codebook, i.e., the ability to effectively quantize the feature space, which directly depends on both the method of learning codewords from training data, and the number of codewords in the codebook. For the former, the learning of good codewords is addressed in [33] by using a hierarchical EM algorithm. For the latter, increasing the number of codewords also increases the computational cost of mapping a video onto the codebook; indeed, the computational complexity is linear in the number of codewords. For the standard bag-of-visual-words, increasing the codebook size is typically not a problem, since the simple L2-distance function is used to identify the visual codeword closest to an image patch. On the other hand, for the BoS in [33], finding the closest codewords to a video patch requires calculating the likelihood of a video patch under each DT codeword using the Kalman filter. For even modest sized codebooks, this results in a heavy computational load. For example, the BoS codebooks of [33] are limited to only 8 codewords.

To address the computational challenges of the BoS representation, in this paper we propose the *BoS Tree*, which combines the expressiveness of a large codebook with

the efficiency of a small codebook. Our proposed approach constructs a bottom-up hierarchy of codewords, and then leverages the tree structure to efficiently index the codewords by choosing only the most-likely branches when traversing the tree. In this way, *the proposed BoS Tree allows for fast look-ups on the codebook and consequently enables the practical use of a larger BoS codebook.*

The contributions of this paper are four-fold. First, we propose the BoS Tree for fast-indexing of large BoS codebooks. Second, we apply the BoS Tree to video classification, reducing the computational cost by at least one order of magnitude versus a standard large codebook, without loss in performance. In fact, the BoS tree *improves* on previous state-of-the-art results on two video texture datasets. Finally, we apply BoS Trees to music annotation experiments, showing suitability of the method to other types of time-series data. The remainder of this paper is organized as follows. In Section 7.2, we first review the BoS representation and DT model, while in Section 7.3 we propose the BoS Tree. Finally, we present experiments on video classification in Section 7.4, and on music annotation in Section 7.5.

7.2 The BoS representation

Analogous to the bag-of-words representation for text documents, the bag-of-systems (BoS) approach [139] represents videos with respect to a vocabulary, where generative time-series models, specifically a linear dynamical *system* or dynamic texture, are used in lieu of words.

7.2.1 The DT codeword

In general, the content of a video is represented by a set of T time series of low-level feature vectors $\mathscr{Y} = \{y_{1:\tau}^{(1)}, \ldots, y_{1:\tau}^{(T)}\}$, which correspond to spatio-temporal cubes sampled from the video, where T depends on the size of the video and the granularity

of the sampling process. Each time series $y_{1:\tau}^{(t)}$ is composed of τ patches extracted at consecutive frames. In the BoS representation, the codebook discretizes the space of time-series using a set of dynamic texture codewords.

The *dynamic texture* (DT) model [54] represents time series data by assuming that it is generated by a doubly embedded stochastic process, in which a lower dimensional hidden Gauss-Markov process $X_t \in \mathbb{R}^n$ encodes the temporal evolution of the observation process $Y_t \in \mathbb{R}^m$. Specifically, the DT model is described by a *linear dynamical system* (LDS):

$$x_t = Ax_{t-1} + v_t \,, \tag{7.1}$$

$$y_t = Cx_t + w_t + \bar{y} \,. \tag{7.2}$$

and is specified by the parameters $\Theta = \{A, Q, C, R, \mu, S, \bar{y}\}$, where the state transition matrix $A \in \mathbb{R}^{n\times n}$ encodes the dynamics of the hidden state variable (e.g., the evolution), the observation matrix $C \in \mathbb{R}^{m\times n}$ encodes the basis functions for representing the sequence, $v_t \sim \mathcal{N}(0,Q)$ and $w_t \sim \mathcal{N}(0,R)$ are the driving respectively observation noises, the vector $\bar{y} \in \mathbb{R}^n$ is the mean feature vector, and $\mathcal{N}(\mu,S)$ specifies the initial condition.

7.2.2 Learning the codebook

The BoS codebook $\mathscr{C}$ is learned from a codebook training set $\mathscr{X}_c$, i.e., a collection of representative videos. A two-stage procedure is typically used, where first each video is summarized with a set of DTs, followed by clustering of the video DTs to obtain the codewords.

In [139], spatio-temporal interest point operators are used to extract interesting motion patches, from which DTs are learned. The video DTs are then embedded into a Euclidean space via non-linear dimensionality reduction in tandem with the Martin

distance. The embedded DTs are clustered in the Euclidean space using the K-means clustering algorithm. Finally, to represent each cluster, the learned DTs, which map the closest to the cluster centers in the embedding, are selected as the codewords.

An alternative approach, presented in [33], is based on the probabilistic framework of the DT. For each video, spatio-temporal patches are extracted using dense sampling, and a dynamic texture mixture (DTM) is learned for each video using the EM algorithm [32]. [33] then directly clusters the video DTs using the hierarchical EM algorithm, producing novel DT cluster centers that are used as the codewords.

While these two approaches effectively produce small-sized codebooks (both [139, 33] use 8 codewords), they are only applicable to small and simple datasets, e.g., UCLA 8-class [139]. Indeed, they are not rich enough to produce accurate classifications when applied to larger or more challenging datasets, as demonstrated in Sections 7.4 and 7.5. A final approach [56] forms a large codebook by directly selecting each DT from the video-level DTMs as a codeword. This forms a very large (and hence rich) codebook, but has significant computational cost when mapping to the codebook.

7.2.3 Projection to the codebook

Once a codebook is available, a video $\mathscr{Y}$ is represented by a BoS histogram $\mathbf{h}_{\mathscr{Y}} \in \mathbb{R}^{|\mathscr{C}|}$ that records how often each codeword appears in that video. To build the BoS histogram of $\mathscr{Y}$, we extract a dense sampling of spatio-temporal cubes from $\mathscr{Y}$. Each cube $y_{1:\tau}^{(t)}$ is compared to each codeword $\Theta_i \in \mathscr{C}$ by mean of its likelihood, efficiently computed with the "innovations" form using the Kalman filter [147] . Defined a quantization threshold $k \in \{1, \ldots, |\mathscr{C}|\}$, each cube is then assigned to the k most likely codewords, and the BoS histogram for $\mathscr{Y}$ is finally built by counting the frequency with which each

codeword is selected. Specifically, the weight of codeword Θ_i is calculated with

$$\mathbf{h}_{\mathscr{Y}}[i] = \frac{1}{|\mathscr{Y}|} \sum_{y^{(t)}_{1:\tau} \in \mathscr{Y}} \frac{1}{k} \mathbb{1}[i \in \operatorname*{argmax}^{k}_{i} P(y^{(t)}_{1:\tau} | \Theta_i)], \tag{7.3}$$

where $\operatorname{argmax}^k_i$ returns the indices of the codewords with the k-largest likelihoods. When the quantization threshold $k = 1$, then (7.3) reduces to the typical notion of the *term frequency* (TF) representation. The effect of $k > 1$ is to counteract quantization errors that can occur when a time series is approximated equally well by multiple codewords.

An alternative to the standard TF representation is the *term frequency-inverse document frequency* (TF-IDF) representation, which takes into account the statistics of the training set by assigning more weight to codewords that appear less frequently in the collection, and down-weighting codewords that are more common. Specifically, given the BoS histogram $\mathbf{h}_{\mathscr{Y}}$ for $\mathscr{Y}$, the corresponding TF-IDF representation is obtained with the following mapping:

$$\hat{\mathbf{h}}_{\mathscr{Y}}[i] = \frac{1}{\alpha} \mathbf{h}_{\mathscr{Y}}[i] \cdot IDF[i], \quad \text{for } i = 1, \ldots, |\mathscr{C}|, \tag{7.4}$$

where the IDF factor is computed as

$$IDF[i] = \log \frac{|\mathscr{X}_c|}{|\{\mathscr{Y} \in \mathscr{X}_c : h_{\mathscr{Y}}[i] > 0\}|} \tag{7.5}$$

and α normalizes the histogram.

Mapping a video $\mathscr{Y}$ to its BoS histogram $\mathbf{h}_{\mathscr{Y}}$ requires a total of $|\mathscr{Y}||\mathscr{C}|$ likelihood computations, i.e., each spatio-temporal cube $y^{(t)}_{1:\tau} \in \mathscr{Y}$ has to be compared to each codeword $\Theta_i \in \mathscr{C}$. When both $|\mathscr{Y}|$ and $|\mathscr{C}|$ are large, projecting one video on the codebook is computationally demanding, especially when using large video patches,

which makes individual likelihood comparisons slow. Therefore, the deployment of a large codebook is impractical due to the associated long delays. However, representing the variety of visual information typical of large and diverse video collections requires a rich, large codebook. In the next section we propose the BoS Tree which, by organizing codewords in a bottom-up hierarchy, reduces the number of computations necessary to index a large collections of codewords.

7.3 The BoS Trees

In this section we propose the *BoS Tree*, which consist of a bottom-up hierarchy of codewords learned from a corpus of representative videos. The bottom level of the tree is formed by a large collection of codewords. A tree structure is then formed by repeatedly using the HEM algorithm to cluster the codewords at one level and using the novel cluster centers as codewords at the new level. Branches are formed between the codewords at a given level and their cluster center at the next higher level.

When mapping a new video onto the codebook, each video patch is first mapped onto the codewords forming the top-level of the BoS Tree. Next, the video patch is *propagated* down the tree, by identifying branches with the most-promising codewords (i.e., with largest likelihood). Selecting the most-likely branches reduces the number of likelihood computations, while also preserving the descriptor quality, since portions of the tree that are not explored are not likely to be codewords for that patch. In this way, the BoS Tree efficiently indexes codewords while preserving the quality of the BoS descriptor, and hence enables the deployment of larger codebooks in practical applications.

In this section, we first discuss the HEM algorithm for clustering dynamic textures, followed by the algorithms used for forming and using the BoS Tree.

7.3.1 The HEM algorithm

Given a collection of DTs, the HEM algorithm for DTMs (HEM-DTM) [33] partitions them into K clusters of DTs that are "similar" in terms of their probability distributions, while also learning a *novel* DT to represent each cluster. This is similar to K-means clustering, with the difference that the data points are DTs instead of Euclidean vectors.

Specifically, the HEM-DTM takes as input a DTM with $K^{(b)}$ components and reduces it to a new DTM with fewer components, i.e., $K^{(r)} < K^{(b)}$. Given the input DTM $\Theta^{(b)} = \{\Theta_i^{(b)}, \pi_i^{(b)}\}_{i=1}^{K^{(b)}}$, the likelihood of a spatio-temporal cube $y_{1:\tau}$ is given by

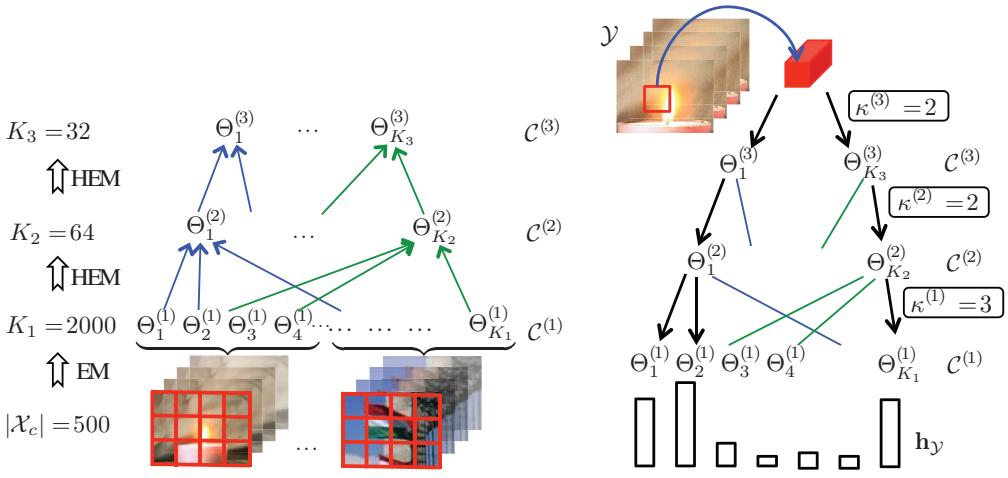
$$p(y_{1:\tau}|\Theta^{(b)}) = \sum_{i=1}^{K^{(b)}} \pi_i^{(b)} p(y_{1:\tau}|z^{(b)} = i, \Theta^{(b)}), \tag{7.6}$$

where $z^{(b)} \sim \text{multinomial}(\pi_1^{(b)}, \cdots \pi_{K^{(b)}}^{(b)})$ is the hidden variable that indexes the mixture components. $p(y_{1:\tau}|z = i, \Theta^{(b)})$ is the likelihood of $y_{1:\tau}$ under the i^{th} mixture component, and $\pi^{(b)}$ is the prior weight for the i^{th} component. The likelihood of the spatio-temporal cube $y_{1:\tau}$ given the reduced mixture $\Theta^{(r)} = \{\Theta_j^{(r)}, \pi_j^{(r)}\}_{j=1}^{K^{(r)}}$ is given by

$$p(y_{1:\tau}|\Theta^{(r)}) = \sum_{j=1}^{K^{(r)}} \pi_j^{(r)} p(y_{1:\tau}|z^{(r)} = j, \Theta^{(r)}), \tag{7.7}$$

where $z^{(r)} \sim \text{multinomial}(\pi_1^{(r)}, \cdots, \pi_{K^{(r)}}^{(r)})$ is the hidden variable for indexing components in $\Theta^{(r)}$.

The HEM-DTM algorithm estimates (7.7) from (7.6) by maximizing the likelihood of N *virtual* spatio-temporal cubes $Y = \{Y^i\}_{i=1}^{K^{(b)}}$ generated accordingly to $\Theta^{(b)}$, where $Y^i = \{y_{1:\tau}^m\}_{m=1}^{N_i}$ is a set of $N_i = \pi_i^{(b)} N$ samples drawn from $\Theta_i^{(b)}$. In order to produce a consistent clustering of the input DTs, the HEM algorithm assigns the whole



(a) Building a BoS Tree. (b) Indexing with a BoS Tree.

Figure 7.1. (a) A BoS Tree is built from a collection of videos $\mathscr{X}_c$ by forming a hierarchy of codewords. (b) The tree structure of the BoS tree enables efficient indexing of codewords.

sample set Y^i to a single component of the reduced model. Assuming that the size of the virtual sample is appropriately large, the law of large number allows the virtual samples to be replaced with an expectation with respect to the input DTs. A complete description of HEM-DTM appears in [33], while here we note that the output of the HEM algorithm is: (1) a clustering of the original $K^{(b)}$ components into $K^{(r)}$ groups, where the cluster membership is encoded by the assignments $\hat{z}_{i,j} = P(z^{(r)} = j | z^{(b)} = i)$, and (2) novel cluster centers represented by the individual mixture components of (7.7), i.e., $\{\Theta_j^{(r)}\}_{j=1}^{K^{(r)}}$.

7.3.2 Building a BoS Tree

A BoS Tree is built from a collection of representative videos $\mathscr{X}_c$ with an unsupervised clustering process based on the HEM-DTM algorithm (Figure 7.1, left). The bottom level of the tree $\mathscr{C}^{(1)} = \{\Theta_i^{(1)}\}_{i=1}^{K_1}$ consists of a large codebook compiled by pooling together the DT codewords extracted from individual videos in $\mathscr{X}_c$ (using the EM-DTM algorithm for DTMs [32]). Starting from the bottom level, a BoS Tree of L levels is built recursively using the HEM-DTM algorithm $L-1$ times. Each new level of the BoS Tree,

i.e., $\mathscr{C}^{(\ell+1)} = \{\Theta_j^{(\ell+1)}\}_{j=1}^{K_{\ell+1}}$, is formed by clustering the K_ℓ DTs at the previous level ℓ into $K_{\ell+1} < K_\ell$ groups using the HEM-DTM algorithm. In particular, the input mixture is given by the DT codewords at level ℓ with uniform weight, i.e., $\Theta^{(b)} = \{\Theta_i^{(\ell)}, \frac{1}{K_\ell}\}_{i=1}^{K_\ell}$, and the novel DT cluster centers learned by the HEM-DTM algorithm are used as codewords at the new level, i.e., $\mathscr{C}^{(\ell+1)} = \{\Theta_j^{(r)}\}_{j=1}^{K_{\ell+1}}$.

The branches between contiguous levels in the BoS Tree are instantiated as dictated by the assignment variables of the HEM-DTM algorithm, which is a function of the Kullback-Leibler (KL) divergence between DTs at each level. In particular, to connect level $\ell+1$ to level ℓ, we define the set of branches for each codeword $j \in [1\, K_{\ell+1}]$ from level $\ell+1$ as

$$B_j^{(\ell+1)} = \{i \in [1\, K_\ell] | j = \underset{h}{\operatorname{argmin}}\, \mathrm{KL}(\Theta_i^{(\ell)} || \Theta_h^{(\ell+1)})\}. \tag{7.8}$$

This is effectively the set of input DTs (at level ℓ) that are assigned to cluster j when constructing level $\ell+1$ of the BoS Tree. Finally, the BoS Tree $\mathscr{T}$ is the collection of codewords at each level and their corresponding branch sets, i.e.,

$$\mathscr{T} = \{\mathscr{C}^{(1)}, \cdots, \mathscr{C}^{(L)}, B^{(2)}, \cdots, B^{(L)}\}.$$

7.3.3 Fast codewords indexing with BoS Trees

The BoS Tree $\mathscr{T}$ allows for quick look-ups in the large codebook $\mathscr{C}^{(1)}$, which forms the bottom level of the tree, by leveraging the hierarchical structure to index the codewords efficiently (Figure 7.1, right). To map a video $\mathscr{Y}$ to its BoS histogram $\mathbf{h}_\mathscr{Y} \in \mathbb{R}^{K_1}$, we extract a dense sampling of spatio-temporal cubes and propagate each cube down only the more promising paths of the BoS Tree. In particular, each cube $y_{1:\tau}$ is initially compared to the codewords at the top level of the BoS Tree (i.e., level L), and

assigned to the $\kappa^{(L)}$ most likely ones,

$$J^{(L)} = \underset{j\in[1\,K_L]}{\overset{\kappa^{(L)}}{\operatorname{argmax}}}\, p(y_{1:\tau}|\Theta_j^{(L)}). \tag{7.9}$$

Next, the cube is propagated down to the successive level following the branches that depart from the codewords selected at the current level,

$$J^{(\ell)} = \underset{i\in\bigcup_{j\in J^{(\ell+1)}} B_j^{(\ell+1)}}{\overset{\kappa^{(\ell)}}{\operatorname{argmax}}}\, p(y_{1:\tau}|\Theta_i^{(\ell)}), \tag{7.10}$$

for $\ell = L-1, L-2, \ldots, 2, 1$.

At the bottom level of the BoS Tree (i.e., $\ell = 1$), the number of occurrences of each codeword is registered, and TF or TF-IDF histograms are then computed. Setting the quantization thresholds $[\kappa^{(1)}, \ldots, \kappa^{(L)}]$ to values larger than 1 counteracts the effect of quantization errors and improves the accuracy of the BoS (in comparison to the full codebook computation), but increases the number of likelihood computations.

An alternative to selecting a *fixed* number of codewords at one level, is to select a *variable* number of codewords based on the uncertainty of the BoS quantization. This is implemented by defining the operator

$$\Omega(J, \mathfrak{T}) = \{j \in J | p(y_{1:\tau}|\Theta_j) \geq \mathfrak{T} \max_{h\in J} p(y_{1:\tau}|\Theta_h)\}$$

which selects all codewords whose likelihood is within a threshold $\mathfrak{T}$ away from the largest, and replacing (7.9) and (7.10) with

$$J^{(L)} = \Omega([1\,K_L], \mathfrak{T}^{(L)}) \tag{7.11}$$

$$J^{(\ell)} = \Omega(\cup_{j\in J^{(\ell+1)}} B_j^{(\ell+1)}, \mathfrak{T}^{(\ell)}). \tag{7.12}$$

The BoS Tree reduces the number of likelihood computations necessary to map a video to its codebook representation. Assuming that a BoS tree has K top-level codewords, L levels, and B branches on average per codeword, the average number of likelihood computations required for the BoS tree look-up is $(K + B(L-1))$, which is much less than the $K \cdot B^{L-1}$ computations required for directly indexing the bottom-level of the tree. Therefore, the BoS Tree enables the use of large and rich codebooks while still maintaining an acceptable look-up time. As the portions of the BoS Tree that are not explored are the ones that are not likely to provide appropriate codewords for a given video (in both the likelihood sense for a tested codeword, and KL-divergence sense for the children of that codeword), there is not expected to be a big loss in performance with respect to linear indexing of a large codebook. We demonstrate this experimentally in Sections 7.4 and 7.5.

7.3.4 Related Work

The bag-of-features "cousin" of our BoS Tree is the tree-structured vector quantizer (TSVQ) [66], which creates a hierarchical quantization of a feature space, and was proposed by Nister *et al.* for efficiently indexing a large vocabulary of image codewords [122], and by Grauman *et al.* to define the bins of multi-resolution histograms [70]. The novelty of our paper is that we apply tree-structured search to a codebook formed by time-series models (i.e., DT models), instead of to a VQ codebook of Euclidean vectors. Although a VQ codebook could be extended to video patches, e.g., by concatenating all frames into a single vector, the resulting image codewords would not handle spatio-temporal variations well, and would be extremely high dimensional.

Efficient indexing of codewords is also related to fast approximate nearest-neighbor (NN) search. Typical approaches to fast NN for real-vectors also exploit a tree data structure, e.g., KD-trees and metric ball trees, and use branch and bound

methods to traverse the tree to find the nearest neighbor [20, 161]. Cayton [31] generalizes metric ball trees to Bregman divergences, enabling fast NN search of histograms using the KL divergence. Adapting approximate search using randomized trees [132] to time-series (using DTs) would be interesting future work.

The BoS Tree proposed here is similar to the Bregman-ball tree in [31], in that both use KL divergence-based clustering to hierarchically construct a tree. The main differences are that our BoS Tree is based on continuous probability distributions (in fact random processes), while [31] is limited to only discrete distributions, and that our nearest-neighbor search is based on data likelihood, not KL divergence. In addition, we use a simple forward search to traverse the tree, whereas [31] uses a more complicated branch-and-bound method. Experimentally, we found that the forward search was both efficient and accurate.[1]

Finally, our BoS Tree is also related to fast image retrieval work by [163], where each image is modeled as a Gaussian distribution and a retrieval tree is constructed using the HEM algorithm for Gaussian mixture models.

7.4 Experiments: video classification

In this section we evaluate the proposed BoS Trees for video classification on three datasets.

7.4.1 Datasets

We consider three datasets, the UCLA-39 dataset [171], the DynTex-35 dataset [177] and the KTH dataset [146]. The first two datasets are composed of video textures, while the third dataset consists in human actions.

[1] We later explored branch-and-bound search for a BoS Tree in [46].

Figure 7.2. Example frames from UCLA-39. Right views (top) are visually different from the corresponding left views (bottom).

Figure 7.3. Example frames from KTH. There are 6 actions.

UCLA-39: The UCLA-39 dataset [171] contains 312 gray-scale videos representing 39 *spatially stationary* classes, which were selected from the original 50 UCLA texture classes [1]. Each video is cropped into a right portion and a left portion (each 48×48), with one used for training and the other for testing. Classification of UCLA-39 is a challenging task, and tests the translation invariance of the feature descriptor, since the training video patch (right portion) is visually quite different from the testing patch (left portion, or vice versa). While other UCLA-based datasets could be used (e.g., [1, 139, 50]), we believe that UCLA-39 is the most challenging variant and hence we adopt it in this paper. Figure 7.2 illustrates typical frames from UCLA-39.

DynTex-35: The DynTex-35 dataset [177] is a collection of videos from 35 different texture classes from everyday surroundings. Originally, the data consisted of a single video of size $192 \times 240 \times 50$ per class. As in [177] , each video is split into 10 non-overlapping sub-videos (each having different spatial and temporal dimensions).

KTH: The KTH dataset [146] consists of 2391 videos of six types of human actions (walking, jogging, running, boxing, hand waving and hand clapping) performed several times by 25 subjects in different scenarios (outdoors, outdoors with scale variation, outdoors with different clothes, and indoors). Each sequence is downsized to 160×120 pixels and have a length of 4 seconds in average. We follow the leave-one-out experimental protocol. Typical frames from KTH are illustrated in Figure 7.3. Note that, although the overall motion in KTH may not be stationary, we demonstrate that our BoS descriptor can represent local patches of motion sufficiently well (Section 7.4.3).

7.4.2 Experiment setup

From each video we extract a dense sampling of spatio-temporal cubes of size $5 \times 5 \times 75$ for UCLA-39, $7 \times 7 \times 30$ for DynTex-35, and $7 \times 7 \times 10$ for KTH. We retain only video cubes with a minimum total variance of 1 for UCLA-39 and DynTex-35, and 5 for KTH.

For each cross-validation split in our datasets, the BoS Tree was learned from the training set only. For each video, a DTM with $K_v = 4$ components is learned from its spatio-temporal cubes using the EM-DTM algorithm. The DTs from all videos are collected to form the DT codewords, i.e., $K_1 = K_v|\mathscr{X}_c|$, where $|\mathscr{X}_c|$ is the size of the training set. The BoS Tree is then formed by successively applying the HEM-DTM algorithm, as described in the previous section. We test different trees for $L \in \{2,3,4\}$ levels, using $K_2 = 64$, $K_3 = 32$ and $K_4 = 16$. We used $\kappa^{(\ell)} = \kappa = 1$ or $\mathfrak{T}(\ell) = \mathfrak{T} = 0.995$ for traversing the BoS Trees.

For video classification, we first map all the videos to their BoS histograms using the learned BoS Tree, to represent the visual content of each video. We then use a k-nearest neighbor (k-NN) classifier or multi-class support vector machine (SVM) for the video classification task. In order to account for the *simplicial* structure of BoS

Table 7.1. Distances and kernels used for classification.

square-root distance	(SR)	$d_s(h_1,h_2) = \arccos(\sum_k \sqrt{h_{1k}h_{2k}})$
χ^2-distance	(CS)	$d_{\chi^2}(h_1,h_2) = \frac{1}{2}\sum_k \frac{\lvert h_{1k}-h_{2k}\rvert}{h_{1k}+h_{2k}}$
χ^2 kernel	(CSK)	$K(h_1,h_2) = 1 - \sum_k \frac{(h_{1k}-h_{2k})^2}{\frac{1}{2}(h_{1k}+h_{2k})}$
Intersection kernel	(HIK)	$K(h_1,h_2) = \sum_k \min(h_{1k},h_{2k})$
Bhattacharyya kernel	(BCK)	$K(h_1,h_2) = \sum_k \sqrt{h_{1k}h_{2k}}$

histograms, we build our k-NN classifier in terms χ^2-distance (CS) or square root-distance (SR) (Table 7.1), which are appropriate distance metrics for histograms. Similarly, for SVM we use the chi-squared kernel (CSK), Bhattacharyya kernel (BCK), or histogram intersection kernel (HIK), as in Table 7.1. The LibSVM software package [36] was used for the SVM, with all parameters selected using 10-fold cross-validation on the training set.

In addition to BoS Trees, we also consider several alternative methods for BoS histograms: direct indexing of the large level-1 codebook (of sizes 624 and 630 on UCLA-39 and DynTex-35); and using a reduced codebook of size $K \in \{16, 32, 64\}$, obtained with the HEM-DTM algorithm as in [33, 114]. For each experiment we registered average (per video) classification accuracy and average (per video) number of likelihood computations executed at test time to produce the BoS histograms, from which we computed the speed up with respect to the large level-1 codebook (X-Speedup). A small number of likelihood computations results in faster look-ups in the codebook, and in a larger speedup. Finally, results are averaged over 2 trials on UCLA-39 (the right sub-video used for training and the left for testing, and vice versa), and leave one sub-video out classification for DynTex-35, with one sub-video from all classes used for testing and the remainder for training.

7.4.3 Video classification results

Table 7.2 reports classification results on UCLA-39 and DynTex-35. Each row refers to the combination of a specific classifier with TF or TF-IDF representation, while columns correspond to different techniques to map videos to BoS histograms: direct-indexed large codebook (large CB), a reduced codebook (reduced CB), and our proposed BoS Tree. Several observations can be made from the results on UCLA-39. First, increasing the codebook size (with direct indexing) substantially increases the classification performance, e.g., with accuracy increasing from 41.35% for 16 codewords to 81.73% for 624 codewords, using TF-IDF and HIK-SVM. However, the computational cost also increases substantially, from requiring 7377 likelihood computations per video (about 5 seconds on a standard desktop PC) to 287,690 (about 182 seconds). Second, using BoS Trees leads to the best performance while requiring only a fraction of the likelihood computations necessary when directly indexing a large codebook. For example with TF-IDF and HIK-SVM, using a 2-level codebook improves accuracy to 82.37%, while also decreasing the average number of likelihood computations to 36,393 (23 seconds), an almost 8 times reduction in computation. For other classifiers, the accuracy is on par, or decreases slightly, compared to the large CB. These results demonstrate that the BoS Trees efficiently and effectively index codewords, and hence allow practical use of a large and rich codebook. Third, although they use about the same number of likelihood operations, BoS Trees significantly outperform the reduced codebooks generated with HEM-DTM in terms of classification accuracy. While the former leverages the hierarchical structure of codewords to access a large collection of codewords, the latter only reduces the size of the codebook which does not result rich enough to produce highly accurate classification. Lastly, using the BoS Tree with $L = 4$ and setting the transversing threshold in (7.9) and (7.10) to $\mathfrak{T} = 0.995$ leads to the best performance. By

executing a limited number of additional likelihood computations (only 30% more the BoS Tree with $L = 4$), the threshold method is able to explore the sub-trees of similar codewords when the patch has near equal preference to both.

Similar conclusions can be drawn on DynTex-35, although the differences in classification accuracy are less substantial due to the easiness of the classification task. Again, we note that the BOS Tree achieves the same accuracy as the direct-indexed large CB, while reducing the computation by almost an order of magnitude.

Finally, the BoS Tree performance improves on the current state-of-the-art reported in the literature [50, 171, 35, 177] on the two textures datasets (last row of Table 7.2). On UCLA-39, the accuracy has improved from 20% [171] or 42.3% [50] to 82.37% for BoS Tree. In contrast to [171], which is based solely on motion dynamics, and [50], which models local appearance and instantaneous motion, the BoS representation is able to leverage both the local appearance (for translation invariance) and motion dynamics of the video to improve the overall accuracy.

In addition, on the KTH dataset our BoS Tree ($L = 2$, $K_1 = 3040$ codewords and $K_2 = 32$) obtains a classification accuracy of 92.3% when using TF-IDF and CSK-SVM, which compares reasonably with results reported in the literature (which range from 87.3% [173] to 95.7% [67]). This shows the potential of BoS Trees as a universal motion descriptor for classification of non-texture, non-stationary videos, such as human actions. However, the comparison with state-of the art system suggests we could improve our BoS Trees with interest point operators, scale-selection, similar to those leveraged by [67] (95.7%) or [25] (93.2%). This is a topic of future work. We also note that the variety of codewords the BoS Tree can index is crucial to achieve good classification results, as a reduced codebook with $K = 32$ performed poorly with 72.8% accuracy.

Table 7.2. Video classification results on UCLA-39 and DynTex-35 using a large codebook, reduced codebooks, and BoS trees. Each row reports the average classification accuracy of a different classifier/kernel combination. The final two rows report the speedup relative to large CB to build the BoS histograms at test time, and reference results (Ref).

Method			UCLA-39								DynTex-35						
			large CB	reduced CB			BoS Tree $\kappa = 1$			$\mathfrak{T} = 0.995$	large CB	reduced CB			BoS Tree $\kappa = 1$		
			$\lvert\mathscr{C}\rvert = 624$	K = 64	K = 32	K = 16	L = 2	L = 3	L = 4	L = 4	$\lvert\mathscr{C}\rvert = 630$	K = 64	K = 32	K = 16	L = 2	L = 3	L = 4
T	N	CS	46.79	46.79	42.95	33.97	43.59	46.47	42.31	42.63	92.86	94.86	92.86	90.86	91.14	92.00	90.57
T	N	SR	62.82	52.88	42.31	34.94	60.90	58.01	55.13	58.33	98.00	98.57	97.71	94.57	98.29	98.00	98.29
F	S	CSK	62.82	52.88	44.87	33.33	60.26	58.97	57.05	59.29	98.29	97.14	96.29	89.43	98.00	98.29	98.86
	V	HIK	78.53	57.69	48.40	39.10	78.53	78.53	73.72	78.85	96.86	96.29	91.71	86.29	97.71	97.71	97.14
	M	BCK	71.79	52.88	45.19	38.78	71.15	71.15	69.55	72.12	96.57	94.57	92.57	84.29	96.29	96.86	96.57
T	N	CS	46.15	45.83	42.95	34.62	43.59	46.15	41.99	42.31	92.29	94.86	92.86	91.14	90.57	92.00	90.86
F	N	SR	65.38	56.09	45.19	36.22	61.22	58.97	56.41	60.58	98.00	98.29	97.43	94.29	98.00	98.00	98.00
I	S	CSK	61.22	53.53	45.51	37.50	61.86	59.94	59.62	60.90	98.29	97.14	96.29	89.43	97.71	97.43	98.00
D	V	HIK	81.73	58.01	48.40	41.35	82.37	82.37	79.81	83.33	97.14	96.57	92.29	85.43	97.43	97.43	97.71
F	M	BCK	74.36	55.13	51.92	40.06	73.72	74.04	72.76	75.32	96.57	95.14	91.71	82.57	96.57	96.29	96.29
Best			81.73	58.01	51.92	41.35	82.37	82.37	79.81	83.33	98.29	98.57	97.71	94.57	98.29	98.29	98.86
X-Speedup			1	9.75	19.50	39	7.91	12.46	17.39	12.74	1	9.84	19.69	39.37	8.31	13.56	19.47
Ref			42.3 [50], 20 [171], 15 [35]								97.14 [177]						

7.5 Experiments: music annotation

In this section, we show the applicability of BoS Trees on an additional type of time-series data, i.e., musical signals.

7.5.1 Dataset

We perform automatic music annotation on the CAL500 dataset (details in [42] and references therein), which is a collection of 502 popular Western songs by as many different artists, and provides binary annotations with respect to a vocabulary of musically relevant tags (annotations or labels), e.g., rock, guitar, romantic. In our experiments we follow the same protocol as in [56] and consider the 97 tags associated to at least 30 songs in CAL500 (11 genre, 14 instrumentation, 25 acoustic quality, 6 vocal characteristics, 35 mood and 6 usage tags).

7.5.2 Experiment setup

The acoustic content of a songs is represented by computing a time-series of 34-bin Mel-frequency spectral features (see [42]), extracted over half-overlapping windows of 92 ms of audio signal. A dense sampling of audio-fragments (analogous to spatio-temporal cubes in videos) is then formed by collecting sequences of $\tau = 125$ consecutive feature vectors (corresponding to approximately 6 seconds of audio), with 80% overlap. A BoS Tree is learned for each cross-validation split from only the training data. The first level of the BoS Tree is formed by estimating a DTM with $K_s = 4$ components from each training song, and then pooling all the DT components together. BoS Trees for $L \in \{2,3,4\}$ levels are tested, with $K_2 = 128$, $K_3 = 64$ and $K_4 = 32$. We use $\kappa(1) = 5$ and $\kappa(\ell) = 2$ for $\ell > 1$.

We cast music annotation as a multi-class multi-label classification task. In particular, given a training set of audio-fragments and their annotations, for each tag we use logistic regression (LR) to learn a linear classifier with a probabilistic interpretation in tandem with the HIK kernel. Given a BoS histogram $\mathbf{h}$ corresponding to a new song, the output of the LR classifiers is normalized to a semantic multinomial, i.e., a vector of tag posterior probabilities. We use the LibLinear software package [58] for the LR classifier, with all parameters selected using 4-fold cross validation on the training set.

On the test set, a novel test song is annotated with the 10 most likely tags, corresponding to the peaks in its semantic multinomial. Retrieval given a one tag query involves rank ordering all songs with respect to the corresponding entry in their semantic multinomials. Performance is measured with the same protocol as in [56]: for annotation, per-tag precision (P), recall (R) and F-score (F), averaged over all tags; and for retrieval, mean average precision (MAP), area under the operating characteristic curve (AROC), and precision at the first 10 retrieved objects (P@10), averaged over all one-tag queries. In

Table 7.3. Music annotation results on CAL500, using a large codebook, reduced codebooks, and BoS Trees. The last column reports the speedup relative to large CB to build the BoS histograms at test time.

		Retrieval			Annotation			
		MAP	AROC	P@10	P	R	F	X-Speedup
large CB	K = 1604	0.454	0.723	0.460	0.406	0.244	0.270	1
reduced CB	K = 128	0.403	0.668	0.402	0.342	0.209	0.227	12.55
	K = 64	0.381	0.649	0.378	0.315	0.192	0.204	25.10
	K = 32	0.368	0.634	0.368	0.298	0.180	0.191	50.20
BoS Tree	L = 2	0.445	0.712	0.451	0.398	0.24	0.261	8.24
	L = 3	0.443	0.712	0.448	0.393	0.235	0.258	11.54
	L = 4	0.439	0.711	0.448	0.394	0.232	0.255	14.31
SML-DTM [42]		0.446	0.708	0.460	0.446	0.217	0.264	1.03

addition, we register the average (per song) number of likelihood computations executed at test time. All reported metrics are result of 5-fold cross validation, where each song appears in the test set exactly once. We compare our BoS Tree to recent results in music annotation based on a large BoS codebook [56], and supervised multi-class labeling with DTM models (SML-DTM) [42].

7.5.3 Music annotation results

In Table 7.3 we report annotation and retrieval performance on the CAL500 dataset. Due to space constraints, we report results only for LR using TF-IDF representation and HIK kernel, but similar trends were noticed for TF representation and other kernels.

We first note that the BoS Trees lead to near optimal performance with respect to the direct-indexed large codebook (implemented with $k = 5$ as in [56]) and SML-DTM, but require an order of magnitude less likelihood computations at test time. In particular, in our experiments, the delay associated to likelihood computations was 8 to 14 seconds per song for the BoS Trees (depending on L), and 2 minutes for direct-indexed large codebook and for SML-DTM. Second, as with video classification, increasing the

codebook size improves the accuracy of music annotation and retrieval, by increasing the richness of the codebook. Again, this justifies the efficacy of large BoS codebooks and our proposed BoS Tree for efficient indexing.

7.6 Conclusions

In this paper we have proposed the BoS Tree, which efficiently indexes DT codewords of a BoS representation using a hierarchical structure. The BoS Tree enables the practical use of larger and richer collections of codewords in the BoS representation. We have proven the efficacy of the BoS Tree on three video datasets, and on a music dataset as well.

7.7 Acknowledgements

Chapter 7, in full, is a reprint of the material as it appears in the Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2012, pp. 1979-1986, Providence (USA), 18-20 June 2012, E. Coviello, A. Mumtaz, A. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 8

Speeding Up NN Search for the Bag of Systems Tree and of High Dimensional Distributions in general

8.1 Introduction

Nearest neighbor (NN) search is a core routine in many applications of machine learning and information retrieval. Formally, given a database X, a query q and a dissimilarity d, the problem consists of finding the $x \in X$ for which $d(x,q)$ is minimum. In this work we are particularly interested in the case where the data points represent probability distributions, and the dissimilarity measure is the Kullback-Leibler (KL) divergence (relative entropy). KL has been extensively used to compare generative models of documents, e.g., in text analysis [130, 24] and content-based image retrieval [133, 136], where documents are often modeled using histograms, as well as in computer vision for video classification [35], where videos are modeled as linear dynamical systems.

Brute-force search is often prohibitive on large datasets. As a consequence, a large body of work, starting from KD-trees [63] and metric-ball trees [125, 161, 111], has investigated the use of spatial data structures to accelerate search. These consist of a hierarchical space decomposition based on geometric properties and database's statistics,

that enables fast search by pruning out portions of the search space via a branch and bound exploration. Cayton [31] extends these ideas to the class of Bregman divergences, of which KL is a member. In particular, Bregman Ball trees (bbtrees) are defined in terms of Bregman balls, and search uses bounds on the Bregman divergence from a query to a Bregman ball. Succesively, Nielsen et al. [120, 121] developed an extension of the bbtree to symmetrized Bregman divergences, Zhang et al. [176] adapted the VA-file and R-tree to decomposable Bregman divergence, and Cayton [30] used the bbtree for efficient range search. Abdullah et al. [2] formally analyze approximated Bregman NN search. These accelerated search methods are not immediately practical in high dimensions [111, 31], where the number of close neighbors is often very large, and consequently the procedure needs to run on many nodes, which results in several additional divergence computations and a total computation time that may become comparable or worse than brute force.

The pruning operation of [31] requires projecting the query onto the shell of a Bregman ball, which is found using a bisection search. However, this has 2 limitations: 1) the procedure is not amenable to distributions outside the exponential family (e.g., latent variable models); 2) two divergence calculations are required for each iteration of the bisection search, which leads to slow performance when the divergence calculation is computationally intensive (e.g., in high dimensions, or latent variable models).

To do more efficient NN search, Bbtrees naturally handle approximate NN operations, by stopping search early after a *fixed budget* of leaves has been visited [31]. This increases efficiency without an excessive degradation of quality.

In this paper we propose a novel branch and bound method based on variational approximations. The corresponding bisection procedure has a small computational overhead, since it requires only one divergence computation at *each backtracked node*, independent of the number of iterations, and then needs to evaluate only scalar functions at each iteration. The algorithm provides a speedup over [31] on medium and

high-dimensionality, and over brute force search at each dimensionality, and returns almost perfect NNs. In addition, our algorithm readily serves search of latent variable distributions.

Other data structures for approximated NN search rely on mapping techniques, for example locality sensitive hashing (reviewed in [149]) adapted to non-metric dissimilarities by [112].

In Section 8.2 we overview Bregman and KL divergences. In Section 8.3 we discuss Cayton's [31] NN search with bbtrees, which lays the basis for our novel contribution, in Section 8.4. Experiments on histogram data and linear dynamical systems are reported in Sections 8.5 and 8.6, and conclusions are drawn in Section 8.7.

8.2 Bregman divergence and Kullback-Leibler divergence

This section provides background on Bregman and KL divergence and describes some related properties.

8.2.1 Bregman divergences

Given a strictly convex differentiable function $f(\cdot)$, the Bregman divergence based on f is $d_f(x,y) = f(x) - f(y) - \langle \nabla f(y), x-y \rangle$. A Bregman divergence $d_f(x,y)$ is convex in x, and the Bregman ball of radius R around μ

$$B(\mu, R) = \{x : d_f(x,\mu) \leq R\} \tag{8.1}$$

is a convex set. The interested reader may refer to [31, 12] for more details.

Let f^* be the dual function of f.[1] Since f is strictly convex, f^* is strictly convex

[1] Given $f(x)$, the dual function is defined as $f^*(y) = \sup_x \{\langle x,y \rangle - f(x)\}$

as well, and f and $f^*$are Legendre dual of each other [141]. One important property is that the gradient ∇f defines a bijective mapping, and the inverse mapping is given by the gradient of the dual function f^*, i.e.:

$$\mu \underset{\nabla f^*(\mu')}{\overset{\nabla f(\mu)}{\longleftrightarrow}} \mu' \tag{8.2}$$

Consistently with the exposition in [31], we use prime (i.e., $'$) to denote the results of applying ∇f.

8.2.2 Regular exponential families, regular Bregman divergences and KL divergence

Given a pair of random variables with p.d.f.s $\mathscr{X}$ and $\mathscr{Y}$, respectively, their KL divergence is the functional

$$D(\mathscr{X}||\mathscr{Y}) = \int \mathscr{X}(\omega) \log \frac{\mathscr{X}(\omega)}{\mathscr{Y}(\omega)} d\omega = \mathbb{E}_{\mathscr{X}} \left[\log \frac{\mathscr{X}}{\mathscr{Y}} \right]. \tag{8.3}$$

The KL divergence is never negative, and is zero iff $\mathscr{X} = \mathscr{Y}$ (a.e.).

There is an interesting correspondence between Bregman divergences and KL divergence, which occurs when we deal with exponential family distributions [12]. In particular, for any *regular* Bregman divergence d_f based on $f(\mu)$, there is associated a *regular* exponential family of distributions

$$\mathscr{F}_{f^*} = \left\{ p_{f^*,\mu'}(\cdot) = \exp\left(\langle \phi(\cdot), \mu' \rangle - f^*(\mu') \right) \right\}, \tag{8.4}$$

where $\phi(\cdot)$ are the sufficient statistics, μ' are the canonical parameters, and the dual function f^* is the partition function. The gradient operation ∇f^* maps from canonical to mean parameters μ [166]. $f(\mu)$ is the negative entropy and its gradient ∇f maps μ to

the canonical parameters μ' [166]. d_f is the KL divergences between members of $\mathscr{F}_{f^*}$.

Hence, when $\mathscr{X}$ and $\mathscr{Y}$ are distributions from the same exponential family with mean parameters x and y respectively, we have that

$$D(\mathscr{X}||\mathscr{Y}) = d_f(x,y) = f(x) - f(y) - \langle \nabla f(y), x - y \rangle, \tag{8.5}$$

where the l.h.s is a functional of two p.d.f.s and the r.h.s. is a function of the mean parameters.

Note that, in (8.3) there is a large contribution to the KL divergence $D(\mathscr{X}||\mathscr{Y})$ from points of large support of $\mathscr{X}$ unless $\mathscr{Y}$ has large support as well.[2]

This has interestingly consequences, one of which concerns an operation we will use later. Given $y_1, \ldots, y_n$ with weights $\theta_1, \ldots, \theta_n$ (where $0 \leq \theta_i \leq 1$, $\sum_i \theta_i = 1$) the *left-sided* centroid [119]:

$$c_L = \arg\min_\mu \sum_i \theta_i d_f(\mu, y_i) \tag{8.6}$$

is *zero forcing*, i.e., the corresponding density will not exceed the support of the input and will likely focus on one mode [119].[3]

8.3 Branch and bound for BB trees

In this section we briefly review the branch and bound method for Bregman ball trees by Cayton [31].

[2] In particular, if the support of $\mathscr{X}$ is not entirely contained in the support of $\mathscr{Y}$, $D(\mathscr{X}||\mathscr{Y})$ will be unbounded.

[3] On the opposite, the right-sided centroid $c_R = \arg\min_x \sum_i \theta_i d_f(y_i, \mu)$ is zero avoiding and will cover well entire the support of the input [119].

8.3.1 BB trees

Bregman ball trees and the associated search routine follow the same principles of KD trees [63] and metric ball trees [125, 161, 111], with the difference that they are based on Bregman balls instead of rectangular cells or metric balls. A bbtree consists of a binary tree that partitions a database $X = \{x_1, \ldots, x_n\}$. Every node i is associated with a subset of points $X_i \subset X$, and defines a Bregman ball with center μ_i and radius R_i such that $\forall x \in X_i : x \in B(\mu_i, R_i)$. Each non-leaf node i is associated with left and right child nodes, l and r, and X_i is consequently split between X_l and X_r. The entirety of leaf nodes covers X. A Bregman Ball tree can be grown from the database X in a top-down fashion, recursively using [12].

8.3.2 Searching with BB trees

Given a query q, we are interested in its left-NN[4]

$$x_L = \arg\min_{x \in X} d_f(x, q). \tag{8.7}$$

Search with a BB tree proceeds as follow. The algorithm initially descends the BB tree, starting from the root. At every non-leaf node the algorithm descends through the most promising child node and temporarily ignores the sibling node. Once the algorithm reaches a leaf node X_i, it selects the candidate nearest neighbor $x_c = \arg\min_{x \in X_i} d_f(x_i, q)$.

At this point, the algorithm backtracks, and explores an originally ignored sibling j if

$$d_f(x_c, q) > \min_{x \in B(\mu_j, R_j)} d_f(x, q). \tag{8.8}$$

where the right side of (8.8) is the Bregman projection of q onto the Bregman ball

[4]Cayton [31] shows that the right-NN can be found using the left-NN algorithm for the divergence d_{f^*} and the database $X' = \{x'_1, \ldots, x'_n\}$.

$B(\mu_j, R_j)$ (see Figure 8.1(a)). Cayton [31] proves that the optimal x_p of (8.8) corresponds to an $x_p{}' = \nabla f(x_p)$ along the line $\theta\mu' + (1-\theta)q'$, $0 \leq \theta \leq 1$,[5] lays on the shell of $B(\mu_j, R_j)$ and can be found by bisection search over θ.

At step i of the bisection search, given θ_i, a point $x'_{\theta_i} = \theta_i\mu' + (1-\theta_i)q'$ is identified along the line, and the corresponding $x_{\theta_i} = \nabla f^*(x'_{\theta_i})$ is recovered. This in fact consists of computing a left-sided centroid [119]

$$x_{\theta_i} = \arg\min_x \theta_i d_f(x,\mu) + (1-\theta_i) d_f(x,q) \tag{8.9}$$

$$= \nabla f^*(\theta_i\mu' + (1-\theta_i)q'). \tag{8.10}$$

8.3.3 Bregman projection vs. stopping early

Since exact evaluation of the Bregman projection (8.8) is not needed, Cayton [31] derives stopping conditions based on upper and lower bounds

$$a \leq \min_{x \in B(\mu_j, R_j)} d_f(x,q) \leq A. \tag{8.11}$$

Upper and lower bounds are computed at each iteration of the bisection, until either $d(x_c, q) < a$ (prune the node) or $d(x_c, q) > A$ (explore the node).

The lower bound a is given by weak duality:

$$\mathscr{L}(\theta) = d_f(x_\theta, q) + \frac{\theta}{1-\theta}\left(d_f(x_\theta, \mu) - R\right), \tag{8.12}$$

where $\mathscr{L}(\theta)$ is the Lagrangian of the right side of (8.8) and $0 \leq \theta \leq 1$. The upper bound

[5] This requires to define the Lagrange dual function of (8.8), $\inf_x d_f(x,q) + \lambda(d_f(x,\mu) - R)$, $\lambda \geq 0$, set the gradient to zero, and use the change of variable $\theta = \frac{\lambda}{1+\lambda}$.

Algorithm 5. `CanPrune by Cayton [31]`

1: **Input**: θ_l, θ_r, q, x_c, μ, R
2: Set $\theta = \frac{\theta_l+\theta_r}{2}$
3: Set $x_\theta = \nabla f^*(\theta\mu' + (1-\theta)q')$
4: Compute $d_f(x_\theta, q)$ and $d_f(x_\theta, \mu)$
5: **if** $\mathscr{L}(\theta) > d_f(x_c, q)$
Return `Yes`
6: **else if** $x_\theta \in B(\mu, R)$ and $d_f(x_\theta, q) < d_f(x_c, q)$
Return `No`
7: **else if** $x_\theta \notin B(\mu, R)$
Return `CanPrune`(θ, θ_r,q,x_c,μ,R)
8: **else if** $x_\theta \in B(\mu, R)$
Return `CanPrune`(θ_l, θ,q,x_c,μ,R)

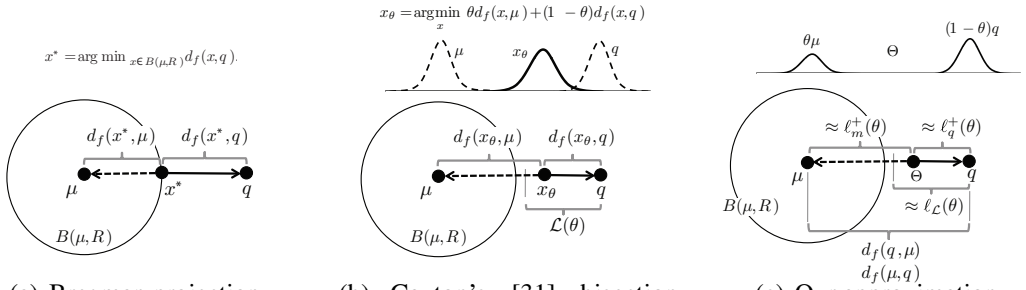
A comes directly from the primal problem:

$$\textbf{if } x_\theta \in B(\mu_j, R_j) \Rightarrow d_f(x_\theta, q) \geq \min_{x \in B(\mu_j, R_j)} d_f(x, q). \tag{8.13}$$

Hence the algorithm operates bisection search on θ as follows (see Figure 8.1(b)). At the i-th step, given θ_i, the algorithm computes the left-sided centroid $\nabla f^*(\theta_i\mu' + (1-\theta_i)q')$ and then checks the bounds. If $\mathscr{L}(\theta_i) > d_f(x_c, q)$, it prunes the node. Else, if $x_{\theta_i} \in B(\mu_j, R_j)$, it updates the upper bound, and if $d_f(x_{\theta_i}, q) < d_f(x_c, q)$ the node must be searched. If neither bound holds, the algorithm continues the bisection. The procedure from [31] is reported in Algorithm 5 for the reader's convenience. Note that in each iteration of `CanPrune`, two divergences are calculated in Step 4.

8.4 Variational Branch and Bound

In this section we present a novel and approximated bisection search based on variational inequalities. The proposed method requires only one divergence operation at each backtracked node (independent of the number of iterations), is efficient in high dimensions, and can be applied to latent variable models. The effect of the approximation



(a) Bregman projection (b) Cayton's [31] bisection search (c) Our approximation

Figure 8.1. a) Bregman projection of q onto the Bregman ball $B(\mu,R)$. The optimum x^* lies at the intersection of shell of $B(\mu,R)$ and the image under ∇f^* of the line segment between μ' and q'. There is no general analytical solution, and x^* can be found with bisection search. b) Algorithm 5 performs bisection search over θ, at each iteration computing a new centroid x_θ between μ and q (Step 3) and the two divergences from x_θ to the node μ and the query q, $d_f(x_\theta,\mu)$ and $d_f(x_\theta,q)$ (Step 4). Conditions to stop the bisection depend on these two divergences. c) Our method uses a mixture Θ with components μ and q and weights θ and $1-\theta$, instead of computing a new centroid x_θ. $d_f(x_\theta,\mu)$ and $d_f(x_\theta,q)$ are approximated with $\ell_m^+(\theta)$ and $\ell_q^+(\theta)$, which are functions only of the scalar θ, given the fixed quantities $d_f(\mu,q)$ and $d_f(q,\mu)$. Conditions to stop the bisection depend only on $\ell_m^+(\theta)$ and $\ell_q^+(\theta)$.

is discussed in Section 8.4.4.

8.4.1 Overview and notation

In this section we assume that d_f is a regular Bregman divergence (i.e., KL divergence for an exponential family). Let $\mathscr{M}$, $\mathscr{Q}$, $\mathscr{X}_c$ be the distributions corresponding, respectively, to the node, the query, and the candidate nearest neighbor. Their mean parameters are, respectively, μ, q and x_c. Define a mixture distribution of components $\mathscr{M}$ and $\mathscr{Q}$ with weights θ and $1-\theta$:

$$\Theta = \theta\mathscr{M} + (1-\theta)\mathscr{Q}. \tag{8.14}$$

Our algorithm works as illustrated in Figure 8.1(c). Instead of explicitly computing the left-sided centroid x_θ (Step 3 of `CanPrune`, Figure 8.1(b)) at each recursive iteration,

it uses a mixture model Θ (equation (8.14)). This allows to approximate $d_f(x_\theta, \mu)$ and $d_f(x_\theta, q)$ with lower bounds to $D(\Theta||\mathscr{M})$ and $D(\Theta||\mathscr{Q})$. The bounds depend only on $d_f(\mu, q)$ and $d_f(q, \mu)$ (which are fixed), and on the mixing parameter θ. As a consequence, bisection search only requires evaluating of simple scalar functions (as opposed to divergences).

8.4.2 Variational lower bounds

Our algorithm builds on the variational approximation to the KL divergence between mixtures by Hershey and Olsen [79],[6] and on the observation that the approximation holds as an *inequality* whenever one of the two terms is unimodal.

Variational approximation to the KL [79]

Let $\mathscr{A} = \{\pi_i, \mathscr{A}_i\}$ be a mixture with weights π_i and components $\mathscr{A}_i$. Similarly, let $\mathscr{B} = \{\omega_j, \mathscr{B}_j\}$ be a different mixture. Assume the mixture components $\mathscr{A}_i$ and $\mathscr{B}_j$ belong to some family for which we can compute the KL divergence. Consider the KL divergence between $\mathscr{A}$ and $\mathscr{B}$:

$$D(\mathscr{A}||\mathscr{B}) = \mathbb{E}_{\mathscr{A}}[\log \frac{\mathscr{A}}{\mathscr{B}}] = \mathbb{E}_{\mathscr{A}}[\log \mathscr{A}] - \mathbb{E}_{\mathscr{A}}[\log \mathscr{B}]. \tag{8.15}$$

Hershey and Olsen [79] derive a lower bound to the expected log-likelihood terms on the right-hand side of (8.15), and in turn an *approximation* (as the difference of two lower bounds) to the KL:

$$D(\mathscr{A}||\mathscr{B}) \approx \sum_i \pi_i \log \frac{\sum_{i'} \pi_{i'} \exp\{-D(\mathscr{A}_i||\mathscr{A}_{i'})\}}{\sum_j \omega_j \exp\{-D(\mathscr{A}_i||\mathscr{B}_j)\}}.$$

[6] Cfr., Section 7 in [79]

Variational lower bound to KL

Assume that $\mathscr{A} = \Theta = \theta\mathscr{M} + (1-\theta)\mathscr{Q}$ is a mixture of two components, and that $\mathscr{B}$ consists of a single component, and consider their KL divergence

$$\begin{aligned} D(\Theta||\mathscr{B}) = &\ \mathbb{E}_\Theta[\log\Theta] - \mathbb{E}_\Theta[\log\mathscr{B}] \\ = &\ \underbrace{\theta\mathbb{E}_\mathscr{M}[\log\Theta] + (1-\theta)\mathbb{E}_\mathscr{Q}[\log\Theta]}_{\mathbb{E}_\Theta[\log\Theta]} \\ &- \underbrace{\theta\mathbb{E}_\mathscr{M}[\log\mathscr{B}] + (1-\theta)\mathbb{E}_\mathscr{Q}[\log\mathscr{B}]}_{\mathbb{E}_\Theta[\log\mathscr{B}]}. \end{aligned} \tag{8.16}$$

The first term cannot be computed exactly, but can be lower bounded [79]. The second term can be computed exactly since it only involves expected log-likelihoods of individual components. Consequently, the variational approximation to $D(\Theta||\mathscr{B})$ holds as a *lower bound*. Dealing with lower bounds (as opposed to approximations) allows to characterize the errors made by Algorithm 6 (see Section 8.4.4).

Lower bounds to $D(\Theta||\mathscr{M})$ and $D(\Theta||\mathscr{Q})$ are easily derived (see Appendix C.1):

$$\begin{aligned} \ell_m = &\ \theta\log\left[\theta + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}\right] \\ &+ (1-\theta)\log\left[\theta + (1-\theta)\exp\{D(\mathscr{Q}||\mathscr{M})\}\right], \end{aligned} \tag{8.17}$$

$$\begin{aligned} \ell_q = &\ \theta\log\left[\theta\exp\{D(\mathscr{M}||\mathscr{Q})\} + (1-\theta)\right] \\ &+ (1-\theta)\log\left[\theta\exp\{-D(\mathscr{Q}||\mathscr{M})\} + 1 - \theta\right]. \end{aligned} \tag{8.18}$$

Monotonicity

When $D(\mathscr{Q}||\mathscr{M})$ is very close to zero, ℓ_m is not always monotonic (see Appendix C.2 for an illustration). This would make the bound useless for bisection search. Luckily, we can easily derive a monotonic bound $\ell_q^+(\theta) = \max(0, \ell_q(\theta))$, i.e., by setting ℓ_m to zero when it is not informative (i.e., when it is negative).

Lemma 8.4.1 $\ell_m^+(\theta) = \max(0, \ell_m(\theta))$ *is monotonic in* $[0\,1]$.

Proof $\ell_m(\theta)$ is convex in $[0\,1]$.[7] As a consequence of convexity, $\ell_m(\theta)$ can cross zero at most twice, and in particular at most twice in $[0\,1]$.
We have that $\ell_m(1) = 0$ and $\ell_m(0) = D(\mathscr{Q}||\mathscr{M}) \geq 0$. If $\ell_m(\theta)$ crosses zero only once (i.e., at $\theta = 1$), then $\ell_m(\theta)$ is monotonic for $\theta \in [0\,1]$. If $\ell_m(\theta)$ crosses zero also at $\theta^* \in (0\,1)$, we have that $\ell_m(\theta)$ is monotonic for $\theta \in [0\,\theta^*]$ (and until it reaches its minimum).[8] The result follows. ∎

Similarly, we can define the lower bound $\ell_q^+(\theta) = \max(0, \ell_q(\theta))$ (which is monotonic as well).

8.4.3 Approximated pruning algorithm

Our algorithm is based on the quantities:

$$\ell_q^+(\theta),\ \ell_m^+(\theta),\ \ell_{\mathscr{L}}(\theta) \equiv \ell_q^+(\theta) + \frac{\theta}{1-\theta}\left(\ell_m^+(\theta) - R\right). \tag{8.19}$$

The algorithm performs bisection search over θ, and attempts to locate the θ for which $l_m^+(\theta) = R$, using early stopping. As a lower bound we use $\ell_{\mathscr{L}}(\theta)$ instead of $\mathscr{L}(\theta)$. We set the upper bound to $\ell_q^+(\theta)$ whenever $\ell_m^+(\theta) < R$.[9]

At each iteration i, the algorithm computes the quantities in (8.19). If $\ell_{\mathscr{L}}(\theta_i) > d_f(x_c, q)$, it prunes the node. Else, if $\ell_m^+(\theta_i) < R$, it updates the upper bound, and if $\ell_q^+(\theta_i) < d_f(x_c, q)$ the node is searched. If neither bound holds, the algorithm continues the bisection. The procedure is summarized in Algorithm 6.

---

[7] This follows from the positivity of the second derivative $\frac{d^2\ell_m(\theta)}{d\theta^2}$ in $[0\,1]$ see see Appendix C.3.

[8] For $\theta \in [\theta^*\,1]$, $\ell_m(\theta)$ is negative, and is a trivial bound.

[9] We could update the upper bound for early stopping using an upper bound to $d_f(x_\theta, q)$ instead of ℓ_q^+. The reason we *do not* is to make the algorithm over-explorative under the circumstances explained in Section 8.4.4.

Algorithm 6. `CanPruneApprox`

1: **Input**: θ_l, θ_r; Const: $d_f(\mu,q)$, $d_f(q,\mu)$, R, $d_f(x_c,q)$
2: Set $\theta = \frac{\theta_l+\theta_r}{2}$
3: Compute $\ell_q^+(\theta)$, $\ell_m^+(\theta)$ and $\ell_{\mathscr{L}}(\theta)$
4: **if** $\ell_{\mathscr{L}}(\theta) > d_f(x_c,q)$
Return `Yes`
5: **else if** $\ell_m^+(\theta) < R$ and $\ell_q^+(\theta) < d_f(x_c,q)$
Return `No`
6: **else if** $\ell_m^+(\theta) > R$
Return `CanPruneApprox`(θ, θ_r)
7: **else if** $\ell_m^+(\theta) < R$
Return `CanPruneApprox`(θ_l, θ)

8.4.4 Discussion

The proposed algorithm executes faster on a node than Cayton's original algorithm. In particular, at *every* recursive iteration the latter requires updating the centroid x_θ plus two divergence operations (i.e., computing $d_f(x_\theta,q)$ and $d_f(x_\theta,\mu)$). On high dimensional data, this results in a significant overhead, since individual divergence operations are expensive, and, most importantly, the procedure needs to be executed on a large fraction of the nodes (due to the large number of close neighbors in high dimensions [111]). Not surprisingly, Algorithm 5 did not work well for exact NN search on high dimensional data [31], where it registered slow-downs (instead of speedups) relative to brute force search.

On the other hand, our method drastically reduces the amount of computation per backtracked node to a single divergence operation, independently of the number of iterations in the bisection search. At each recursive call of Algorithm 6, $\ell_m^+(\theta)$ and $\ell_q^+(\theta)$ are functions of the scalar θ and of the *fixed* quantities $d_f(q,\mu)$ and $d_f(\mu,q)$. Since $d_f(q,\mu)$ is already computed by the search routine when descending the tree (for choosing between a node's left and right child), only one additional divergence needs to be computed. We expect this to give our algorithm an edge for efficiency on

high-dimensional data.

The solution proposed by Cayton [31] to speed up retrieval time uses the bbtree for approximate search, by fixing a maximum budget of leaves that can be explored for each query, after which backtracking is stopped. This is suboptimal, since the approximation is independent of the query and may blindly ignore (once the budget is depleted) promising portions of the search space only because they appear later in the backtracking order. Our approximation, on the other hand, only depends on the parameters of the query and of the nodes of the tree (as opposed to a fixed leaf budget), and the resulting backtracking will adapt better to individual queries and the local structure of the tree, as illustrated below.

In fact, we can argue that Algorithm 6 tends to be over-explorative on nodes close to the query, and under-explorative on nodes further away. Since we are using in sequence an approximation and a lower bound,

$$d_f(x_\theta, q) \approx \quad D(\Theta||\mathscr{Q}) \geq \quad \ell_q^+(\theta), \tag{8.20}$$

$$d_f(x_\theta, \mu) \approx \quad D(\Theta||\mathscr{M}) \geq \quad \ell_m^+(\theta). \tag{8.21}$$

in *general* (C.10) (C.11) are not bounds to $d_f(x_\theta, q)$ and $d_f(x_\theta, \mu)$. However, as shown by the next claim, when the query and the node have very close distributions, (C.10) and (C.11) hold as lower bounds.

Claim 8.4.2 *If $q' = \mu' + \delta'$, for δ' small, we have $d_f(x_\theta, \mu) \geq l_m^+(\theta)$, and, similarly, $d_f(x_\theta, q) \geq \ell_q^+(\theta)$.*

Proof We have $x'_\theta = \mu' + (1-\theta)\delta'$ and $\mu' = x'_\theta - (1-\theta)\delta'$. Consider the Riemannian manifold around x'_θ (with curvature $\nabla f^*(x'_\theta)$), and that the Riemannian metrics associated

to f and f^* have identical infinitesimal length [7, 119]. Consequently we have:[10]

$$d_f(x_\theta,\mu) = \frac{1}{2}(1-\theta)^2\delta'^t\nabla^2 f^*(x'_\theta)\delta' \tag{8.22}$$

$$= \frac{1}{2}(1-\theta)^2\delta^t\nabla^2 f(x_\theta)\delta = (1-\theta)^2\Delta \tag{8.23}$$

where we use the notation $\Delta = \frac{1}{2}\delta^t\nabla^2 f(x_\theta)\delta$ to reduce clutter. Similarly, we have that $d_f(q,\mu) = \frac{1}{2}\Delta$ and $d_f(\mu,q) = \frac{1}{2}\Delta$. Using the approximations $\exp\{a\} = 1+a$ and $\log(1+a) = a$ (for $|a|$ small), we have that:

$$\ell_m(\theta) = (1-\theta)\log[1+(1-\theta)\Delta] + \theta\log[1-(1-\theta)\Delta] \tag{8.24}$$

$$\leq \log\{(1-\theta)[1+(1-\theta)\Delta] + \theta[1-(1-\theta)\Delta]\} \tag{8.25}$$

$$= \log\left\{1+\left[(1-\theta)^2-\theta(1-\theta)\right]\Delta\right\} \tag{8.26}$$

$$= \left[(1-\theta)^2-\theta(1-\theta)\right]\Delta \leq (1-\theta)^2\Delta \tag{8.27}$$

where (C.15) follow from Jensen inequality and (C.17) form the fact that $\theta(1-\theta)\geq 0$ for $\theta\in[0\,1]$. Since $d_f(x_\theta,\mu)\geq 0$ we also have $d_f(x_\theta,\mu)\geq \ell_m^+(\theta)$. ∎

In this case, the decision whether to prune a node is based on a looser lower bound (i.e., $\ell_{\mathscr{L}}(\theta)\leq\mathscr{L}(\theta)$ instead of $\mathscr{L}(\theta)$ in Algorithm 5), and Algorithm 6 will consequently *prune less frequently*. Similarly, checking the condition $\ell_m^+(\theta) < R$ results in inflating the ball around the node, and may determine *more explorations*.

On the opposite, if q is far from μ, (C.10) and (C.11) may actually hold as upper bounds, making the backtracking *under-explorative*.[11] This has the effect that our algorithm better accounts for the *sparsity* of data relative to the dimensionality, reducing

[10] To show (C.12) we can use Legendre duality $d_f(x_\theta,\mu) = f(x_\theta)+f^*(\mu')-\langle\mu',x_\theta\rangle$ and second order Taylor expansion of $f^*(\mu') = f^*(x'_\theta-(1-\theta)\delta')$ around x'_θ.

[11] Since x_θ is zero forcing (as a left-sided centroid), it will have smaller support than Θ, and consequently $d_f(x_\theta,\mu)$ (respectively, $d_f(x_\theta,q)$) will be smaller than $D(\Theta||\mathscr{M})$ (respectively, $D(\Theta||\mathscr{Q})$).

the backtracking for high-dimensional data. Skipping a *large* region of space centered far away form the query determines computational savings, but does not affect NN performance too much. Even if the region could *potentially* contain points very close to query, it most likely does not, since the points it contains are very few (relative to the dimensionality).

Our results in Section 8.5 demonstrate that, on moderate- and high-dimensional data, our algorithm is very efficient and returns almost perfect nearest neighbors. In addition, the approximation does not require empirically tuning any meta-parameter (such as the leaf budget in [31]).

Interestingly, since our algorithm requires only the computation of divergence terms (and bypasses the computation of the centroid), it is by no means limited to the exponential family. In fact, it can be readily applied to NN-search of any family of distributions,[12] as long as the divergence between individual members can be computed (efficiently). In Section 8.6 we illustrate this for time series models with latent variables.

Algorithm 5, on the other hand, cannot be adapted to latent variable models in an efficient way. In particular, the left-sided centroid x_θ cannot be computed analytically, since the hidden-state bases corresponding to the query and the node (i.e., $\mathscr{Q}$ and $\mathscr{M}$) may be mismatched.[13,14]

[12] For distributions not in the exponential family, Algorithm 6 (and in particular the quantities in (8.19)), need to be expressed in terms of KL divergences between the distributions $\mathscr{Q}$, $\mathscr{M}$, $\mathscr{X}_c$, since they do not correspond to a regular Bregman divergence.

[13] For example, for hidden Markov models, the labeling of hidden states may be swapped.

[14] Note that naïvely approximating the centroid with iterative algorithms or numerical techniques would be inefficient.

8.5 Experiments on histogram data

We first consider experiments on histogram data, where we can directly compare our proposed method against Cayton's [31] exact and approximate NN search.[15] We consider the 9 histogram datasets from [31] listed in Table 8.1, most of which are fairly high dimensional. We refer to our algorithm as `Variational`, Cayton's exact and approximated search as `Cayton` and `CaytonApprox`, respectively, and brute force search as `Brute`.

Performance is measured in terms of speedup relative to brute force search, and by the *average* number of elements in the database that are closer to the query than the returned one (NC) [31]. For exact algorithms (e.g., `Cayton` and `Brute`), NC = 0 always. All results are averages over queries not in the database.

In Table 8.1 we compare performance of `Variational` to `Cayton`. In general, on moderate to high dimensional data ($d \geq 32$), our algorithm provides larger speedups than `Cayton`. The gain in computational efficiency has a very modest effect on the quality of `Variational`, as demonstrated by the low NC values. In addition, whereas `Cayton` is *slower* than brute force search on the two datasets of highest dimensionality ($d = 371$ and $d = 1111$), `Variational` always registers a speedup. In particular, on "Sift signatures", `Variational` provides a substantial $6\times$ speedup over brute force search, with an NC of 0.0784. Further examining the NC values, `Variational` finds the NN 95.7% of the time, the 2nd NN 2.6%, and 2+ NN 1.7% (with a low average approximation ratio of $\varepsilon = 0.3$).[16]

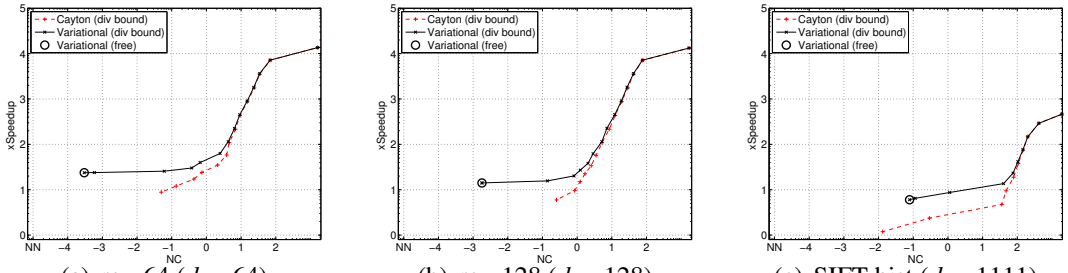
In Figure 8.2 we compare `Variational` to `CaytonApprox`, on three datasets of medium to high dimensionality. For both methods, we set a fixed budget β_d of

[15] For Cayton's [31] NN search we use the code available at `http://lcayton.com`. Our method is implemented in the same framework and is available on the authors' web pages.

[16] The approximation ratio is the smallest ε satisfying $d_f(x,q) \leq (1+\varepsilon) d_f(x_q,q)$, where x_q is q's true NN and x is the returned element.

Table 8.1. Results on histogram data, using different datasets. The speedup is w.r.t. brute force search. The databases for rcv-*n*, Corel hist, Semantic space, and SIFT signature have size 500,000, 60,000, 4,500 and 10,000, respectively. The number of queries is 10,114, 6,616, 500 and 2,360, respectively

		`Cayton`	`Variational`	
DATA SET	DIM	SPEEDUP	SPEEDUP	NC
RCV-8	8	60.75	21.23	0.0001
RCV-16	16	30.49	19.27	0.0000
RCV-32	32	17.18	24.60	0.0016
RCV-64	64	8.19	24.21	0.0003
COREL HIST	64	2.56	4.14	0.0020
RCV-128	128	4.29	14.51	0.0018
RCV-256	256	2.58	2.95	0.0000
SEMANTIC SPACE	371	0.93	1.29	0.0020
SIFT SIGNATURES	1111	0.86	5.88	0.0784



(a) rcv-64 ($d = 64$) (b) rcv-128 ($d = 128$) (c) SIFT hist ($d = 1111$)

Figure 8.2. Log-log plots (in base 10) for approximated NN-search using `Variational` and `CaytonApprox`, $\beta_d \in [2^0\, 2^{\log_2 |X|}]$. The vertical axis is the exponent of the speedup over `Brute`; the horizontal axis is the exponent of the average number of DB points closer to the query than the returned result (NC). Points toward the top-left corner are better.

divergence computations allowed per query, after which backtracking stops.[17] In general, `Variational` is superior to `CaytonApprox`, achieving more effective speedups and lower NC values—note that in Figure 8.2(c) the left-most point for `CaytonApprox` (the dashed red line) corresponds to no speedup, and is hence of no practical interest. In particular, the threshold-free version of `Variational` (from Table 8.1 and circled in Figure 8.2) does not require setting any meta-parameter and is efficient also on high

[17] Backtracking stops either when the budget is consumed or when one of the early stopping conditions is met.

dimensional data. These results show that, by adapting backtracking to the query and the local structure of the tree, our approximation effectively explores promising regions of the search space and achieves computational savings by under-exploring less promising ones.

8.6 Experiments on dynamic textures

In this section we test our algorithm on NN search of dynamic textures (DTs) [54], which have been used to model video [45, 113] and music [13, 42].

A DT model represents a n-dimensional time series of length T, $y_1, \ldots, y_T$, as generated by a *linear dynamical system* (LDS):

$$\begin{aligned} s_t &= As_{t-1} + v_t \ , \\ y_t &= Cs_t + w_t + \bar{y} \ . \end{aligned} \tag{8.28}$$

where $S_t \in \mathbb{R}^m$ is a lower-dimensional hidden state process ($m < n$) that governs the dynamics, A is a $m \times m$ transition matrix, C a $n \times m$ bases matrix, and V_t and W_t are Gaussian noise processes. The KL divergence between two DTs can be computed efficiently with a recursion [35], so we can readily use our branch and bound search.

We consider the 4 datasets used by Coviello et al. [45], three of video and one of music. A dataset is first divided into training/test splits, and a database of DTs X is compiled from the training set, with each DT modeling a portion of a video (or song). The database (of DTs) is then hierarchically organized in a tree structure using the Bag-of-Systems (BoS) Tree from [45], which produces balls around each DT-node $\mathscr{M}$, compact in terms of $D(\cdot \,||\, \mathscr{M})$. Finally, we model portions of test videos (or songs) as DTs, and for each we find its NN in X using the BoS Tree in tandem with our backtracking algorithm.

Videos are preprocessed into a dense sampling of spatio-temporal cubes of pixels

Table 8.2. Results for NN-search of DTs. Speedup is w.r.t. `Brute`. DIM are the degrees of freedom of the DT models.

DATA SET	SOURCE	DIM	$\lvert X \rvert$	SPEEDUP	NC
UCLA-39	VIDEO	211	624	2.11	0.13
DYTEX	VIDEO	355	630	6.16	1.24
KTH	VIDEO	760	3040	1.74	1.42
CAL500	MUSIC	385	1600	2.60	4.52

(as in [45]). Each training video is then modeled as a dynamic texture mixture (DTM) with 4 components, and the databases X are formed by selecting all the video-level DT components. For each test video, we compute a query-DT from each cube, using [54]. Songs are first represented as a dense sampling of sequences of low-level features (as in [45]). Each training song is modeled as a DTM with 4 components, and all the song-level DTs form the database X. For each experiment we compute average speedup over `Brute` and NC, using the same cross validation splits as [45]. Results on the video and music datasets are in Table 8.2. Note that in average `Variational` returns good answers. For example, on the CAL500 music data, where we registered the highest NC value, `Variational` still returns in average answers in the top 0.35%.

8.7 Conclusions

We have presented a branch and bound algorithm based on variational approximations. We have shown nearly perfect and efficient NN-search on histogram data of challenging dimensionality, as well as applicability to more complex generative time series models.

8.8 Acknowledgements

Chapter 8, in full, is a reprint of the material as it appears in the Proceedings of The 30th International Conference on Machine Learning, JMLR W&CP 28 (3) :468-476, Atlanta, Georgia (USA), 16-21 June 2013, E. Coviello, A. Mumtaz, A.B. Chan and G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Appendix A

Derivation of the VHEM algorithm for H3Ms

A.1 Derivation of the Lower Bounds

A.1.1 Appendix A.1 Lower Bound on $\mathbb{E}_{\mathscr{M}_i^{(b)}}\left[\log p(Y_i|\mathscr{M}^{(r)})\right]$

The lower bound (4.15) on $\mathbb{E}_{\mathscr{M}_i^{(b)}}\left[\log p(Y_i|\mathscr{M}^{(r)})\right]$ is computed by introducing the variational distribution $q_i(z_i)$ and applying (4.10)

$$
\begin{aligned}
&\mathbb{E}_{\mathscr{M}_i^{(b)}}\left[\log p(Y_i|\mathscr{M}^{(r)})\right] \\
&\geq \max_{q_i}\sum_j q_i(z_i=j)\left\{\log\frac{p(z_i=j|\mathscr{M}^{(r)})}{q_i(z_i=j)}+\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(Y_i|\mathscr{M}_j^{(r)})]\right\} && \text{(A.1)}\\
&= \max_{q_i}\sum_j q_i(z_i=j)\left\{\log\frac{p(z_i=j|\mathscr{M}^{(r)})}{q_i(z_i=j)}+\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})^{N_i}]\right\} && \text{(A.2)}\\
&= \max_{q_i}\sum_j q_i(z_i=j)\left\{\log\frac{p(z_i=j|\mathscr{M}^{(r)})}{q_i(z_i=j)}+N_i\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]\right\} && \text{(A.3)}\\
&\geq \max_{q_i}\sum_j q_i(z_i=j)\left\{\log\frac{p(z_i=j|\mathscr{M}^{(r)})}{q_i(z_i=j)}+N_i\mathscr{L}_{HMM}^{i,j}\right\} \triangleq \mathscr{L}_{H3M}^{i}\cdot, && \text{(A.4)}
\end{aligned}
$$

where in (A.2) we use the fact that Y_i is a set of N_i i.i.d. samples. In (A.3), $\log p(\mathbf{y}|\mathscr{M}_j^{(r)})$ is the observation log-likelihood of an HMM, which is essentially a mixture distribution.

Since the latter expectation cannot be calculated directly, in (A.4) we use instead the lower bound $\mathscr{L}^{i,j}_{HMM}$ defined in (4.13).

A.1.2 Lower Bound on $\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]$

To calculate the lower bound $\mathscr{L}^{i,j}_{HMM}$,starting with (4.13), we first rewrite the expectation $\mathbb{E}_{\mathscr{M}_i^{(b)}}$ to explicitly marginalize over the HMM state sequence β from $\mathscr{M}_i^{(b)}$,

$$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})] = \mathbb{E}_{\beta|\mathscr{M}_i^{(b)}}\left[\mathbb{E}_{\mathbf{y}|\beta,\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]\right] \tag{A.5}$$

$$= \sum_{\beta} \pi_{\beta}^{(b),i} \mathbb{E}_{\mathbf{y}|\beta,\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]. \tag{A.6}$$

We introduce a variational distribution $q^{i,j}(\rho|\beta)$ on the state sequence ρ, which depends on a particular sequence β from $\mathscr{M}_i^{(b)}$. Applying (4.10) to (A.6), we have

$$\mathbb{E}_{\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\mathscr{M}_j^{(r)})]$$

$$\geq \sum_{\beta} \pi_{\beta}^{(b),i} \max_{q^{i,j}} \sum_{\rho} q^{i,j}(\rho|\beta) \left\{ \log \frac{p(\rho|\mathscr{M}_j^{(r)})}{q^{i,j}(\rho|\beta)} + \mathbb{E}_{\mathbf{y}|\beta,\mathscr{M}_i^{(b)}}[\log p(\mathbf{y}|\rho,\mathscr{M}_j^{(r)})] \right\} \tag{A.7}$$

$$= \sum_{\beta} \pi_{\beta}^{(b),i} \max_{q^{i,j}} \sum_{\rho} q^{i,j}(\rho|\beta) \left\{ \log \frac{p(\rho|\mathscr{M}_j^{(r)})}{q^{i,j}(\rho|\beta)} + \sum_{t} \mathbb{E}_{\mathscr{M}_{i,\beta_t}^{(b)}}[\log p(y_t|\mathscr{M}_{j,\rho_t}^{(r)})] \right\} \tag{A.8}$$

$$\geq \sum_{\beta} \pi_{\beta}^{(b),i} \max_{q^{i,j}} \sum_{\rho} q^{i,j}(\rho|\beta) \left\{ \log \frac{p(\rho|\mathscr{M}_j^{(r)})}{q^{i,j}(\rho|\beta)} + \sum_{t} \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \right\} \triangleq \mathscr{L}^{i,j}_{HMM}, \tag{A.9}$$

where in (A.8) we use the conditional independence of the observation sequence given the state sequence, and in (A.9) we use the lower bound, defined in (4.14), on each expectation.

A.1.3 Lower Bound on $\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho}^{(r)})]$

To derive the lower bound $\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$ for (4.14), we first rewrite the expectation with respect to $\mathscr{M}_{i,\beta}^{(b)}$ to explicitly marginalize out the GMM hidden assignment variable ζ,

$$\begin{aligned}\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho}^{(r)})] &= \mathbb{E}_{\zeta|\mathscr{M}_{i,\beta}^{(b)}}\left[\mathbb{E}_{\mathscr{M}_{i,\beta,\zeta}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho}^{(r)})]\right] \\ &= \sum_{m=1}^{M} c_{\beta,m}^{(b),i}\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}\left[\log p(y|\mathscr{M}_{j,\rho}^{(r)})\right].\end{aligned}$$

Note that $p(y|\mathscr{M}_{j,\rho}^{(r)})$ is a GMM emission distribution as in (4.1). Hence, the observation variable is y, and the hidden variable is ζ. Therefore, we introduce the variational distribution $q_{\beta,\rho}^{i,j}(\zeta|m)$, which is conditioned on the observation y arising from the mth component in $\mathscr{M}_{i,\beta}^{(b)}$, and apply (4.10),

$$\begin{aligned}&\mathbb{E}_{\mathscr{M}_{i,\beta}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho}^{(r)})] \\ &\geq \sum_{m=1}^{M} c_{\beta,m}^{(b),i}\max_{q_{\beta,\rho}^{i,j}}\sum_{\zeta=1}^{M} q_{\beta,\rho}^{i,j}(\zeta|m)\left\{\log\frac{p(\zeta|\mathscr{M}_{j,\rho}^{(r)})}{q_{\beta,\rho}^{i,j}(\zeta|m)}+\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\zeta}^{(r)})]\right\} \quad \text{(A.10)} \\ &\triangleq \mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}.\end{aligned}$$

A.2 Derivation of the E-Step

GMM: Substituting the variational distribution $\eta_{\ell|m}^{(i,\beta),(j,\rho)}$ into (4.17), we have

$$\mathscr{L}_{GMM}^{(i,\beta),(j,\rho)} = \sum_{m=1}^{M} c_{\beta,m}^{(b),i}\max_{\eta_{\ell|m}^{(i,\beta),(j,\rho)}}\sum_{\ell=1}^{M}\eta_{\ell|m}^{(i,\beta),(j,\rho)}\left\{\log\frac{c_{\rho,\ell}^{(r),j}}{\eta_{\ell|m}^{(i,\beta),(j,\rho)}}+\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})]\right\}$$

The maximizing variational parameters are obtained as (see Appendix A.4.2)

$$\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)} = \frac{c_{\rho,\ell}^{(r),j} \exp\left\{\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})]\right\}}{\sum_{\ell'} c_{\rho,\ell'}^{(r),j} \exp\left\{\mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}}[\log p(y|\mathscr{M}_{j,\rho,\ell'}^{(r)})]\right\}}, \tag{A.11}$$

HMM: Substituting the variational distribution $\phi^{i,j}$ into (4.16), we have

$$\mathscr{L}_{HMM}^{i,j} = \sum_{\beta} \pi_{\beta}^{(b),i} \max_{\phi^{i,j}} \sum_{\rho} \phi^{i,j}(\rho|\beta) \left\{ \log \frac{\pi_{\rho}^{(r),j}}{\phi^{i,j}(\rho|\beta)} + \sum_{t} \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \right\}. \tag{A.12}$$

The maximization of (A.12) with respect to $\phi_t^{i,j}(\rho_t|\rho_{t-1},\beta_t)$ and $\phi_1^{i,j}(\rho_1|\beta_1)$ is carried out independently for each pair (i,j), and follow [78]. In particular it uses a backward recursion, starting with $\mathscr{L}_{\tau+1}^{i,j}(\beta_t,\rho_t) = 0$, for $t = \tau,\ldots,2$,

$$\hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) = \frac{a_{\rho_{t-1},\rho_t}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta_t),(j,\rho_t)} + \mathscr{L}_{t+1}^{i,j}(\beta_t,\rho_t)\right\}}{\sum_{\rho}^{S} a_{\rho_{t-1},\rho}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta_t),(j,\rho)} + \mathscr{L}_{t+1}^{i,j}(\beta_t,\rho)\right\}} \tag{A.13}$$

$$\mathscr{L}_t^{i,j}(\beta_{t-1},\rho_{t-1}) = \sum_{\beta=1}^{S} a_{\beta_{t-1},\beta}^{(b),i} \log \sum_{\rho=1}^{S} a_{\rho_{t-1},\rho}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta),(j,\rho)} + \mathscr{L}_{t+1}^{i,j}(\beta,\rho)\right\}, \tag{A.14}$$

and terminates with

$$\hat{\phi}_1^{i,j}(\rho_1|\beta_1) = \frac{\pi_{\rho_1}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta_1),(j,\rho_1)} + \mathscr{L}_2^{i,j}(\beta_1,\rho_1)\right\}}{\sum_{\rho}^{S} \pi_{\rho}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta_1),(j,\rho)} + \mathscr{L}_2^{i,j}(\beta_1,\rho)\right\}} \tag{A.15}$$

$$\mathscr{L}_{\text{HMM}}^{i,j} = \sum_{\beta=1}^{S} \pi_{\beta}^{(b),i} \log \sum_{\rho=1}^{S} \pi_{\rho}^{(r),j} \exp\left\{\mathscr{L}_{\text{GMM}}^{(i,\beta),(j,\rho)} + \mathscr{L}_2^{i,j}(\beta,\rho)\right\} \tag{A.16}$$

where (A.16) is the maxima of the terms in (A.12) in Section 4.3.3.

H3M: Substituting the variational distribution z_{ij} into (4.15), we have

$$\mathscr{L}^i_{H3M} = \max_{z_{ij}} \sum_j z_{ij} \left\{ \log \frac{\omega_j^{(r)}}{z_{ij}} + N_i \mathscr{L}^{i,j}_{HMM} \right\}. \tag{A.17}$$

The maximizing variational parameters of (A.17) are obtained using Appendix A.4.2,

$$\hat{z}_{ij} = \frac{\omega_j^{(r)} \exp(N_i \mathscr{L}^{i,j}_{HMM})}{\sum_{j'} \omega_{j'}^{(r)} \exp(N_i \mathscr{L}^{i,j'}_{HMM})}. \tag{A.18}$$

A.3 Derivation of the M-Step

The M-steps involves maximizing the lower bound in (4.11) with respect to $\mathscr{M}^{(r)}$, while holding the variational distributions fixed,

$$\mathscr{M}^{(r)^*} = \underset{\mathscr{M}^{(r)}}{\operatorname{argmax}} \sum_{i=1}^{K^{(b)}} \mathscr{L}^i_{H3M}. \tag{A.19}$$

Substituting (4.20) and (4.21) into the objective function of (A.19),

$$\mathscr{L}(\mathscr{M}^{(r)}) = \sum_{i=1}^{K^{(b)}} \mathscr{L}^i_{H3M} \tag{A.20}$$

$$= \sum_{i,j} \hat{z}_{ij} \left\{ \log \frac{\omega_j^{(r)}}{\hat{z}_{ij}} + N_i \sum_\beta \pi_\beta^{(b),i} \sum_\rho \hat{\phi}^{i,j}(\rho|\beta) \left[\log \frac{\pi_\rho^{(r),j}}{\hat{\phi}^{i,j}(\rho|\beta)} + \sum_t \mathscr{L}^{(i,\beta_t),(j,\rho_t)}_{GMM} \right] \right\} \tag{A.21}$$

In the following, we detail the update rules for the parameters of the reduced model $\mathscr{M}^{(r)}$.

A.3.1 HMMs Mixture Weights

Collecting terms in (A.21) that only depend on the mixture weights $\{\omega_j^{(r)}\}_{j=1}^{K^{(r)}}$, we have

$$\widetilde{\mathscr{L}}(\{\omega_j^{(r)}\}) = \sum_i \sum_j \hat{z}_{ij} \log \omega_j^{(r)} = \sum_j \left[\sum_i \hat{z}_{ij} \right] \log \omega_j^{(r)} \tag{A.22}$$

Given the constraints $\sum_{j=1}^{K^{(r)}} \omega_j^{(r)} = 1$ and $\omega_j^{(r)} \geq 0$, (A.22) is maximized using the result in Appendix A.4.1, which yields the update in (4.26).

A.3.2 Initial State Probabilities

The objective function in (A.21) factorizes for each HMM $\mathscr{M}_j^{(r)}$, and hence the parameters of each HMM are updated independently. For the j-th HMM, we collect terms in (A.21) that depend on the initial state probabilities $\{\pi_\rho^{(r),j}\}_{\rho=1}^{S}$,

$$\widetilde{\mathscr{L}_j}(\{\pi_\rho^{(r),j}\}) = \sum_i \hat{z}_{ij} N_i \sum_{\beta_1} \pi_{\beta_1}^{(b),i} \sum_{\rho_1} \hat{\phi}_1^{i,j}(\rho_1|\beta_1) \log \pi_{\rho_1}^{(r),j} \tag{A.23}$$

$$= \sum_{\rho_1} \sum_i \hat{z}_{ij} N_i \underbrace{\sum_{\beta_1} \pi_{\beta_1}^{(b),i} \hat{\phi}_1^{i,j}(\rho_1|\beta_1)}_{\hat{v}_1^{i,j}(\rho_1)} \log \pi_{\rho_1}^{(r),j} \tag{A.24}$$

$$= \sum_\rho \sum_i \hat{z}_{ij} N_i \hat{v}_1^{i,j}(\rho) \log \pi_\rho^{(r),j} \tag{A.25}$$

$$\propto \sum_\rho \left[\sum_i \hat{z}_{ij} \omega_i^{(b)} \hat{v}_1^{i,j}(\rho) \right] \log \pi_\rho^{(r),j}, \tag{A.26}$$

where in the (A.25) we have used the summary statistic defined in (4.23). Considering the constraints $\sum_{\rho=1}^{S} \pi_\rho^{(r),j} = 1$ and $\pi_\rho^{(r),j} \geq 0$, (A.26) is maximized using the result in Appendix A.4.1, giving the update formula in (4.27).

A.3.3 State Transition Probabilities

Similarly, for each HMM $\mathscr{M}_j^{(r)}$ and previous state ρ, we collect terms in (A.21) that depend on the transition probabilities $\{a_{\rho,\rho'}^{(r),j}\}_{\rho'=1}^{S}$,

$$\widetilde{\mathscr{L}}_{j,\rho}(\{a_{\rho,\rho'}^{(r),j}\}_{\rho'=1}^{S}) = \sum_i \hat{z}_{ij} N_i \sum_\beta \pi_\beta^{(b),i} \sum_\rho \hat{\phi}^{i,j}(\rho|\beta) \log \pi_\rho^{(r),j} \tag{A.27}$$

$$\propto \sum_i \hat{z}_{ij} N_i \sum_\beta \left[\pi_{\beta_1}^{(b),i} \prod_{t=2}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \right] \sum_\rho \left[\hat{\phi}_1^{i,j}(\rho_1|\beta_1) \prod_{t=2}^{\tau} \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) \right] \left[\sum_{t=2}^{\tau} \log a_{\rho_{t-1},\rho_t}^{(r),j} \right] \tag{A.28}$$

$$= \sum_i \hat{z}_{ij} N_i \sum_{\rho_1} \sum_{\rho_2} \sum_{\beta_2} \sum_{\beta_1} \underbrace{\pi_{\beta_1}^{(b),i} \hat{\phi}_1^{i,j}(\rho_1|\beta_1)}_{\nu_1^{i,j}(\rho_1,\beta_1)} a_{\beta_1,\beta_2}^{(b),i} \hat{\phi}_2^{i,j}(\rho_2|\rho_1,\beta_2) \Big[\log a_{\rho_1,\rho_2}^{(r),j} \tag{A.29}$$

$$\underbrace{\qquad\qquad}_{\xi_2^{i,j}(\rho_1,\rho_2,\beta_2)}$$

$$+ \sum_{\beta_3 \cdots \beta_\tau} \sum_{\rho_3 \cdots \rho_\tau} \prod_{t=3}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \prod_{t=3}^{\tau} \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) \sum_{t=3}^{\tau} \log a_{\rho_{t-1},\rho_t}^{(r),j} \Big] \tag{A.30}$$

$$= \sum_i \hat{z}_{ij} N_i \sum_{\rho_1} \sum_{\rho_2} \sum_{\beta_2} \xi_2^{i,j}(\rho_1,\rho_2,\beta_2) \log a_{\rho_1,\rho_2}^{(r),j} \tag{A.31}$$

$$+ \sum_i \hat{z}_{ij} N_i \sum_{\rho_2} \sum_{\rho_3} \sum_{\beta_3} \sum_{\beta_2} \underbrace{\sum_{\rho_1} \xi_2^{i,j}(\rho_1,\rho_2,\beta_2)}_{\nu_2^{i,j}(\rho_2,\beta_2)} a_{\beta_2,\beta_3}^{(b),i} \hat{\phi}_3^{i,j}(\rho_3|\rho_2,\beta_3) \Big[\log a_{\rho_2,\rho_3}^{(r),j} \tag{A.32}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\xi_3^{i,j}(\rho_2,\rho_3,\beta_3)}$$

$$+ \sum_{\beta_4 \cdots \beta_\tau} \sum_{\rho_4 \cdots \rho_\tau} \prod_{t=4}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \prod_{t=4}^{\tau} \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) \sum_{t=4}^{\tau} \log a_{\rho_{t-1},\rho_t}^{(r),j} \Big] \tag{A.33}$$

$$= \cdots = \sum_i \hat{z}_{ij} N_i \sum_{t=2}^{\tau} \sum_{\rho_{t-1}} \sum_{\rho_t} \sum_{\beta_t} \xi_t^{i,j}(\rho_{t-1},\rho_t,\beta_t) \log a_{\rho_{t-1},\rho_t}^{(r),j} \tag{A.34}$$

$$\propto \sum_i \hat{z}_{ij} N_i \sum_{\rho'} \underbrace{\sum_{t=2}^{\tau} \sum_{\beta} \xi_t^{i,j}(\rho,\rho',\beta)}_{\hat{\xi}^{i,j}(\rho,\rho')} \log a_{\rho,\rho'}^{(r),j} \tag{A.35}$$

$$\propto \sum_{\rho'=1}^{S} \left[\sum_i \hat{z}_{ij} \omega_i^{(b)} \hat{\xi}^{i,j}(\rho,\rho') \right] \log a_{\rho,\rho'}^{(r),j}. \tag{A.36}$$

Considering the constraints $\sum_{\rho'=1}^{S} a_{\rho,\rho'}^{(r),j} = 1$ and $a_{\rho,\rho'}^{(r),j} \geq 0$, (A.36) is maximized using the result in Appendix A.4.1, giving the update in (4.27).

A.3.4 Emission Probability Density Functions

The cost function (A.21) factors also for each GMM indexed by (j,ρ,ℓ). Factoring (A.21),

$$\widetilde{\mathscr{L}}(\mathscr{M}_{j,\rho,\ell}^{(r)}) = \sum_i \hat{z}_{ij} N_i \sum_\beta \pi_\beta^{(b),i} \sum_\rho \hat{\phi}^{i,j}(\rho|\beta) \sum_t \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \tag{A.37}$$

$$= \sum_i \hat{z}_{ij} N_i \sum_\beta \left[\pi_{\beta_1}^{(b),i} \prod_{t=2}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \right] \sum_\rho \left[\hat{\phi}_1^{i,j}(\rho_1|\beta_1) \prod_{t=2}^{\tau} \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) \right] \sum_t \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \tag{A.38}$$

$$= \sum_i \hat{z}_{ij} N_i \sum_{\rho_1} \sum_{\beta_1} \underbrace{\pi_{\beta_1}^{(b),i} \hat{\phi}_1^{i,j}(\rho_1|\beta_1)}_{\nu_1^{i,j}(\rho_1,\beta_1)} \Big[\mathscr{L}_{GMM}^{(i,\beta_1),(j,\rho_1)} \dots \tag{A.39}$$

$$+ \sum_{\beta_2 \cdots \beta_\tau} \prod_{t=2}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \sum_{\rho_2 \cdots \rho_\tau} \prod_{t=2}^{\tau} \hat{\phi}^{i,j}(\rho_t|\rho_{t-1},\beta_t) \sum_{t=2}^{\tau} \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \Big] \tag{A.40}$$

$$= \sum_i \hat{z}_{ij} N_i \sum_{\rho_1} \sum_{\beta_1} \nu_1^{i,j}(\rho_1,\beta_1) \mathscr{L}_{GMM}^{(i,\beta_1),(j,\rho_1)} \tag{A.41}$$

$$+ \sum_i \hat{z}_{ij} N_i \sum_{\rho_2} \sum_{\beta_2} \sum_{\rho_1} \underbrace{\sum_{\beta_1} \left(\nu_1^{i,j}(\rho_1,\beta_1) a_{\beta_1,\beta_2}^{(b),i} \right) \hat{\phi}_2^{i,j}(\rho_2|\rho_1,\beta_2)}_{\nu_2^{i,j}(\rho_2,\beta_2)} \Big[\mathscr{L}_{GMM}^{(i,\beta_2),(j,\rho_2)} \dots \tag{A.42}$$

$$\underbrace{\qquad\qquad\qquad\qquad}_{\xi_2^{i,j}(\rho_1,\rho_2,\beta_2)}$$

$$+ \sum_{\beta_3 \cdots \beta_\tau} \prod_{t=3}^{\tau} a_{\beta_{t-1},\beta_t}^{(b),i} \sum_{\rho_3 \cdots \rho_\tau} \prod_{t=3}^{\tau} \hat{\phi}_t^{i,j}(\rho_t|\rho_{t-1},\beta_t) \sum_{t=3}^{\tau} \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \Big] \tag{A.43}$$

$$= \ldots = \sum_i \hat{z}_{ij} N_i \sum_{t=1}^{\tau} \sum_{\rho_t} \sum_{\beta_t} \nu_t^{i,j}(\rho_t, \beta_t) \mathscr{L}_{GMM}^{(i,\beta_t),(j,\rho_t)} \propto \sum_i \hat{z}_{ij} N_i \sum_{\beta} \underbrace{\sum_{t=1}^{\tau} \nu_t^{i,j}(\rho,\beta)}_{\hat{\nu}_t^{i,j}(\rho,\beta)} \mathscr{L}_{GMM}^{(i,\beta),(j,\rho)}$$

$$\propto \sum_i \hat{z}_{ij} N_i \sum_{\beta=1}^{S} \hat{\nu}^{i,j}(\rho,\beta) \sum_{m=1}^{M} c_{\beta,m}^{(b),i} \, \hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)} \left[\log c_{\rho,\ell}^{(r),j} + \mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}} [\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})] \right] \tag{A.44}$$

$$= \Omega_{j,\rho} \left(\hat{\eta}_{\ell|m}^{(i,\beta),(j,\rho)} \left[\log c_{\rho,\ell}^{(r),j} + \mathbb{E}_{\mathscr{M}_{i,\beta,m}^{(b)}} [\log p(y|\mathscr{M}_{j,\rho,\ell}^{(r)})] \right] \right), \tag{A.45}$$

where in (A.45) we use the weighted-sum operator defined in (4.30), which is over all base model GMMs $\{\mathscr{M}_{i,\beta,m}^{(b)}\}$. The GMM mixture weights are subject to the constraints $\sum_{\ell=1}^{M} c_{\rho,\ell}^{(r),j} = 1, \forall j, \rho$. Taking the derivative with respect to each parameter and setting it to zero,[1] gives the GMM update equations (4.28) and (4.29).

A.4 Useful Optimization Problems

The following two optimization problems are used in the derivation of the E-step and M-step.

A.4.1

The optimization problem

$$\max_{\alpha_\ell} \quad \sum_{\ell=1}^{L} \beta_\ell \log \alpha_\ell \qquad \text{s.t.} \qquad \sum_{\ell=1}^{L} \alpha_\ell = 1, \quad \alpha_\ell \geq 0, \forall \ell \tag{A.46}$$

is optimized by $\alpha_\ell^* = \frac{\beta_\ell}{\sum_{\ell'=1}^{L} \beta_\ell'}$.

[1] We also considered the constraints on the covariance matrices $\Sigma_{\rho,\ell}^{(r),j} \succ 0$.

A.4.2

The optimization problem

$$\max_{\alpha_\ell} \quad \sum_{\ell=1}^{L} \alpha_\ell \left(\beta_\ell - \log \alpha_\ell\right) \qquad \text{s.t.} \qquad \sum_{\ell=1}^{L} \alpha_\ell = 1 \quad \alpha_\ell \geq 0, \forall \ell \tag{A.47}$$

is optimized by $\alpha_\ell^* = \frac{\exp \beta_\ell}{\sum_{\ell'=1}^{L} \exp \beta_\ell'}$.

Appendix E. Clustering Synthetic Data where the Clustering Model Does Not Match the "True" Model.

In this appendix we present two experiments on clustering synthetic data, where the order of the model used for clustering does not necessarily match the order of the "true" model used to generate the data. In both experiments, the true model consists of $C = 4$ HMMs each with $S = 3$ hidden states—we used the HMMs of experiment (c) in Section 5.3. Data sequences are generated from each of the C HMMs, and then corrupted with Gaussian noise. An HMM with S' states is estimated over each sequence. These HMMs are denoted as *noisy* HMMs, to differentiate them from the original HMMs representing the C classes. The entirety of noisy HMMs are then clustered into C' groups using in turn our VHEM-H3M algorithm and PPK-SC.

In the first experiments we set $S' = S(= 3)$, and vary $C' \in \{3,4,5,6\}$. Hence, the number of clusters does not necessarily match the true number of groups. In the second experiments, we fix $C' = C(= 4)$, and vary $S' \in \{2,3,4,5\}$, i.e., the model order of the noisy HMMs does not match the order of the original HMMs. Results are reported in Table A.1 below.

Performance are more sensitive to selecting a sufficient number of clusters than using the right number of HMM states. In particular, when using fewer clusters than the

true number of classes (e.g., $C' < C$), the Rand-index degrades for both VHEM-H3M and PPK-SC, see experiment (d) in Table A.1. On the opposite, performance are relatively stable when the number of HMM states does not match the true one, e.g., $S' \neq S$, see experiment (e) in Table A.1. In particular, when using fewer HMM states (e.g., $S' < S$) the model can still capture some of the dynamics, and the drop in performance is not significant. It is also interesting to note that using a larger number of HMM states (e.g., $S' > S$) leads to slightly better results. The reason is that, when estimating the HMMs on the corrupted data sequences, there are additional states to better account for the effect of the noise.

Table A.1. Results on clustering synthetic data with VHEM-H3M and PPK-SC, when the model order does not match the order of the true model used for generating the data. We fist vary the number of clusters C', keeping $S' = 3$ fixed to the true value. Then, we vary the number of HMM states S', keeping $C' = 4$ fixed to the true value. Performance is measured in terms of Rand-index, and is averaged over $K \in \{2, 4, 8, 16, 32\}$.

	experiment (d) number of clusters C' varies, $S' = 3$				experiment (e) number of HMM states S' varies, $C' = 4$ fixed			
	2	3	4 (true)	5	2	3 (true)	4	5
VHEM-H3M	0.665	0.782	0.811	0.816	0.801	0.811	0.828	0.832
PPK-SC	0.673	0.729	0.768	0.781	0.766	0.768	0.781	0.798

Appendix B

Additional experiments for the BoS representation

B.1 Details of Design Choices Experiments

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

Table B.1. 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
$\mathscr{X}_c$	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500
$\mathscr{X}_t$	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500	$\subset$ CAL500
$\lvert\mathscr{X}_c\rvert$	64	400	400	400
$\lvert\mathscr{X}_t\rvert$	≈ 400	≈ 400	≈ 400	≈ 400
K	128	50-4800	4800	1600
M	1	1	1-5	1
k	10	10	10	1-100

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 $|\mathscr{X}_c| = 64$. 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 $\mathscr{X}_t$ as the codebook song set $\mathscr{X}_c$ (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[1], i.e., $|\mathscr{X}_c| = 400$, we vary the codebook size $K = K_s|\mathscr{X}_c|$ 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 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 $|\mathscr{X}_c'| = 100$.

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., $\mathscr{X}_c = \mathscr{X}_t$). 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 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 ($|\mathscr{X}_c| = 400$) we vary the number of codewords $\tilde{K}$ of each base model by varying K_s. Table B.2 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.

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

Table B.2. 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

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., $\mathscr{X}_c = \mathscr{X}_t$). 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.

B.2 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 $\mathscr{X}_c$, (2) learning small mixtures independently from individual songs in $\mathscr{X}_c$, (3) learning all codewords simultaneously from all available audio data in $\mathscr{X}_c$ using standard EM, and (4) learning all codewords simultaneously from all available audio data in $\mathscr{X}_c$ 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-A1, reported in Table B.3. 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 Figure B.1. While the 2-D embedding and mean pairwise distance capture slightly different information about the

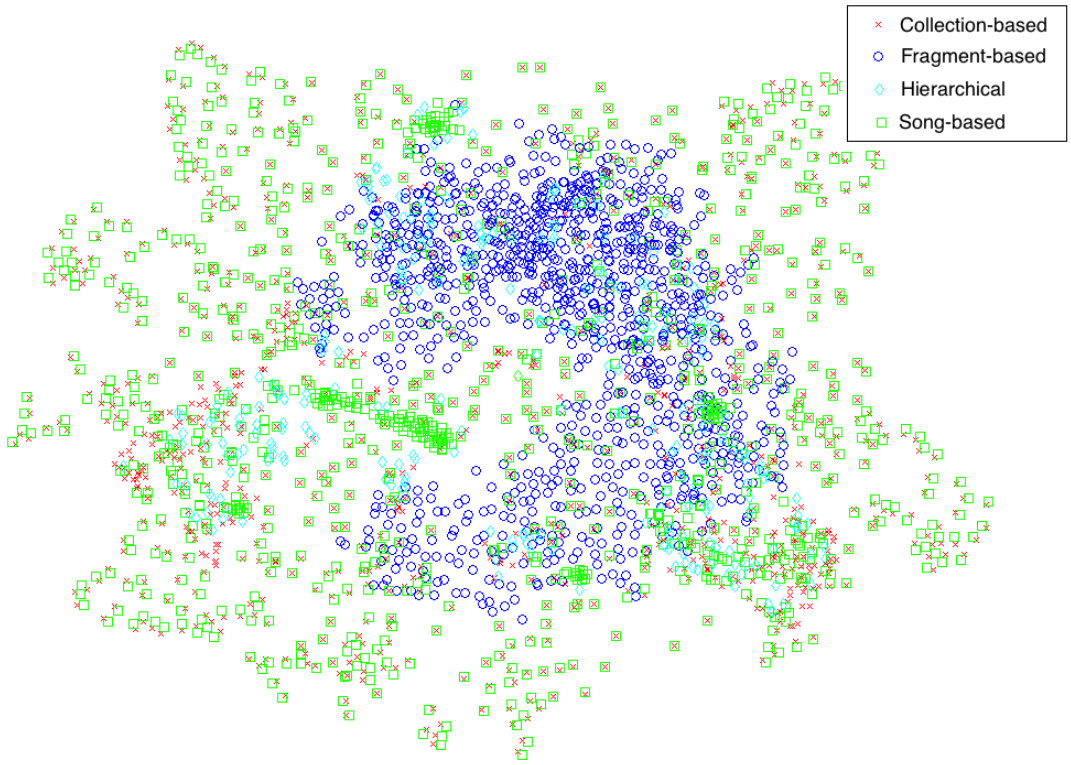


Figure B.1. 2D visualization of Gaussian codeword centers generated using the four different methods of codebook generation. The fragment-based method produced codewords that are more tightly clustered than the other methods, and showed correspondingly low performance in annotation and retrieval experiments (See Table B.3).

spread of the codewords (The cost function 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-A1, Table II); 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

Table B.3. 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

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.

B.3 Details of Unlabeled Songs Experiment

This Appendix provides the details of the parameter settings and design for the experiments in section VII-A3.

Three different codebook song sets are used in this experiment: (1) $\mathscr{X}_c = \text{C400}$, the codebook song set corresponding to the training set for each cross-validation split of CAL500 (as in Section VI-B1), (2) $\mathscr{X}_c = \text{R400}$, a codebook song set that is a subset of 400 randomly selected songs from the CAL10K collection and (3) $\mathscr{X}_c = \text{A5K}$, 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-B2). 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 $\mathscr{X}_c = \text{C400} \cup \text{R400}$, we use $K_s = 4$ to build a codebook of size $K = 9600$, and for the codebook song set $\mathscr{X}_c = \text{C400} \cup \text{A5K}$, 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.

Appendix C

Additional material for Branch and bound

C.1 Derivation of the lower bounds.

The lower bound presented in Section 8.4.2, i.e., ℓ_q and ℓ_m, are easily derived from [79] variational approximation to the KL divergence between mixture models:

$$D(\mathscr{A}||\mathscr{B}) \approx \sum_i \pi_i \log \frac{\sum_{i'} \pi_{i'} \exp\{-D(\mathscr{A}_i||\mathscr{A}_{i'})\}}{\sum_j \omega_j \exp\{-D(\mathscr{A}_i||\mathscr{B}_j)\}}. \tag{C.1}$$

where $\mathscr{A} = \{\pi_i, \mathscr{A}_i\}$ is a mixture with weights π_i and components $\mathscr{A}_i$, and $\mathscr{B} = \{\omega_j, \mathscr{B}_j\}$ is a mixture with weights ω_j and components $\mathscr{B}_j$.

The lower bound to $D(\Theta||\mathscr{Q})$ is found for $\mathscr{A} = \Theta = \{\theta, \mathscr{M}, (1-\theta), \mathscr{Q}\}$ and $\mathscr{B} = \{1, \mathscr{Q}\}$:

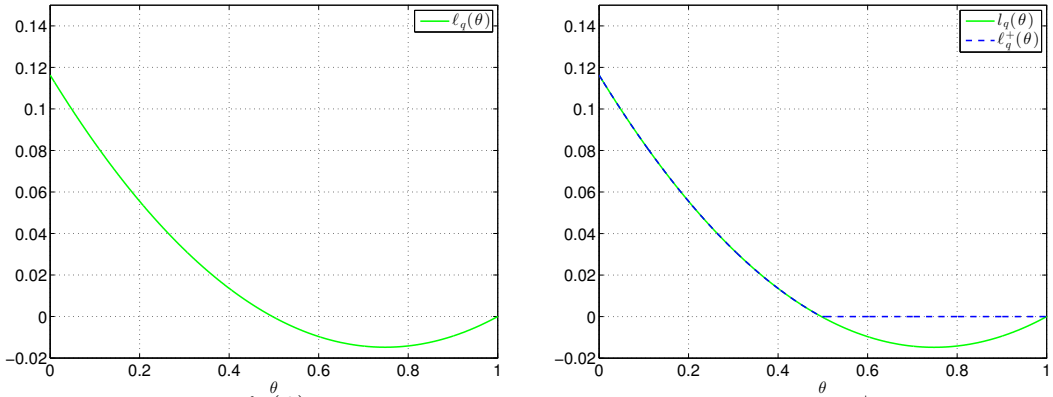
$$\begin{aligned}
\ell_q &= \theta \log \frac{\theta \exp\{-D(\mathscr{M}||\mathscr{M}) + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}}{\exp\{-D(\mathscr{M}||\mathscr{Q})\}} \\
&\quad + (1-\theta)\log \frac{\theta \exp\{-D(\mathscr{Q}||\mathscr{M})\} + (1-\theta)\exp\{-D(\mathscr{Q}||\mathscr{Q})}{\exp\{-D(\mathscr{Q}||\mathscr{Q})\}} \\
&= \theta \log \frac{\theta + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}}{\exp\{-D(\mathscr{M}||\mathscr{Q})\}} + (1-\theta)\log[\theta \exp\{-D(\mathscr{Q}||\mathscr{M})\} + 1 - \theta] \qquad \text{(C.2)} \\
&= \theta \log[\theta \exp\{D(\mathscr{M}||\mathscr{Q})\} + (1-\theta)] + (1-\theta)\log[\theta \exp\{-D(\mathscr{Q}||\mathscr{M})\} + 1 - \theta] \qquad \text{(C.3)}
\end{aligned}$$

where in (C.2) we use the fact that $D(\mathscr{Q}||\mathscr{Q}) = 0$, and in (C.3) we multiply numerator and denominator of the fist term inside the logarithm by $\exp\{D(\mathscr{M}||\mathscr{Q})\}$. Similarly, the lower bound to $D(\Theta||\mathscr{M})$ is found for $\mathscr{A} = \Theta = \{\theta, \mathscr{M}, (1-\theta), \mathscr{Q}\}$ and $\mathscr{B} = \{1, \mathscr{M}\}$:

$$\begin{aligned}
\ell_m &= \theta \log \frac{\theta \exp\{-D(\mathscr{M}||\mathscr{M}) + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}}{\exp\{-D(\mathscr{M}||\mathscr{M})\}} \\
&\qquad + (1-\theta)\log \frac{\theta \exp\{-D(\mathscr{Q}||\mathscr{M})\} + (1-\theta)\exp\{-D(\mathscr{Q}||\mathscr{Q})\}}{\exp\{-D(\mathscr{M}||\mathscr{Q})\}} \\
&= \theta \log [\theta + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}] + (1-\theta)\log \frac{\theta \exp\{-D(\mathscr{Q}||\mathscr{M})\} + 1 - \theta}{\exp\{-D(\mathscr{Q}||\mathscr{M})\}} \quad \text{(C.4)} \\
&= \theta \log [\theta + (1-\theta)\exp\{-D(\mathscr{M}||\mathscr{Q})\}] + (1-\theta)\log [\theta + (1-\theta)\exp\{D(\mathscr{Q}||\mathscr{M})\}] \quad \text{(C.5)}
\end{aligned}$$

Note that, (C.3) and (C.5) hold as lower bounds, as explained in Section 8.4.2.

C.2 Monotonicity of lower bound



(a) $\ell_m(\theta)$ (b) $\ell_m(\theta)$ vs. $\ell_m^+(\theta)$

Figure C.1. $\ell_m^+(\theta)$ is monotonic, whereas $\ell_m(\theta)$ does not need to.

The function $\ell_m(\theta)$ is not always monotonic, as illustrated by the following example.

Consider two histograms $\mathscr{Q}$ and $\mathscr{M}$ with bins:

$$q = [0.2, 0.2, 0.15, 0.25, 0.2] \tag{C.6}$$

$$\mu = [0.2, 0.2, 0.3, 0.1, 0.2] \tag{C.7}$$

for which we have $D(\mathscr{Q}||\mathscr{M}) \approx 0.1163$ and $D(\mathscr{M}||\mathscr{Q}) \approx 0.1251$.

In Figure C.1(a), we plot $\ell_m(\theta)$ for $\theta \in [0\,1]$, which is visibly non monotonic. On the opposite, $\ell_m^+(\theta) = \max(0, \ell_m(\theta))$ is monotonic, as illustrated in Figure C.1(b).

In general, Lemma 1 in Section 8.4.2 gives a proof of the monotonicity of $\ell_m^+(\theta)$. Lemma 1 uses the convexity of $\ell_m(\theta)$, which follows from the positivity of the second derivative (for $\theta \in [0\,1]$). In particular, using $A = \exp\{-D(M||Q)\}$ and $B = \exp\{D(Q||M)\}$ to reduce clutter, we have:

$$\begin{aligned}
\frac{d}{d\theta}\ell_m(\theta) &= \frac{d}{d\theta}\theta\log[\theta+(1-\theta)A] + \frac{d}{d\theta}(1-\theta)\log[\theta+(1-\theta)B] \\
&= \log[\theta(1-A)+A] + \frac{\theta(1-A)}{\theta(1-A)+A} \\
&\quad + \frac{1-B}{\theta(1-B)+B} - \log[\theta(1-B)+B] + \frac{\theta(1-B)}{\theta(1-B)+B}
\end{aligned} \tag{C.8}$$

$$\begin{aligned}
\frac{d^2}{d\theta^2}\ell_m(\theta) &= \frac{d}{d\theta}\frac{d}{d\theta}\ell_m(\theta) \\
&= \frac{(1-A)[\theta(1-A)+2A]}{[\theta(1-A)+A]^2} + \frac{-(1-B)(1-B)}{[\theta(1-B)+B]^2} - \frac{(1-B)[\theta(1-B)+2B]}{[\theta(1-B)+B]^2} \\
&= \frac{(1-A)[\theta(1-A)+2A]}{[\theta(1-A)+A]^2} - \frac{(1-B)[(\theta+1)+B(1-\theta)]}{[\theta(1-B)+B]^2}.
\end{aligned} \tag{C.9}$$

Since $B = \exp\{D(Q||M)\} > 1$, $A = \exp\{-D(M||Q)\} \in [0\,1]$, for $\theta \in [0\,1]$, (C.9) is positive (from which follows the convexity).

C.3 Decisions of the approximated algorithm

In the derivation of Algorithm 6 in, we use in sequence an approximation and a lower bound:

$$d_f(x_\theta, q) \approx \quad D(\Theta||\mathscr{Q}) \geq \quad \ell_q^+(\theta) \tag{C.10}$$

$$d_f(x_\theta, \mu) \approx \quad D(\Theta||\mathscr{M}) \geq \quad \ell_m^+(\theta) \tag{C.11}$$

In general, when (C.10) and (C.11) do not hold as equalities, the algorithm is subject to making two types of incorrect decisions, i.e., exploring parts of the search space that could instead be safely pruned (over-explorative behavior), or pruning away parts that should be explored (under-explorative behavior). Interestingly, we can argue that, instead of making one type of mistake or the other *randomly*, our algorithm is over-explorative on nodes to which the query is relatively close and under-explorative on nodes to which the query is further away.

Conjecture: If $q' = \mu' + \delta'$, for δ' small, we have $d_f(x_\theta, \mu) \geq l_m^+(\theta)$.

Proof: We have $x'_\theta = \mu' + (1-\theta)\delta'$ and $\mu' = x'_\theta - (1-\theta)\delta'$. Consider the Riemannian manifold around x'_θ (with curvature $\nabla f^*(x'_\theta)$), and that the Riemannian metrics associated to f and f^* have identical infinitesimal length [7, 119]. Consequently we have:[1]

$$d_f(x_\theta, \mu) = \frac{1}{2}(1-\theta)^2 \delta'^t \nabla^2 f^*(x'_\theta)\delta' \tag{C.12}$$

$$= \frac{1}{2}(1-\theta)^2 \delta^t \nabla^2 f(x_\theta)\delta = (1-\theta)^2\Delta \tag{C.13}$$

where we use the notation $\Delta = \frac{1}{2}\delta^t \nabla^2 f(x_\theta)\delta$ to reduce clutter. Similarly, we have that $d_f(q, \mu) = \frac{1}{2}\Delta$ and $d_f(\mu, q) = \frac{1}{2}\Delta$. Using the approximations $\exp\{a\} = 1 + a$ and

[1] To show (C.12) we can use Legendre duality $d_f(x_\theta, \mu) = f(x_\theta) + f^*(\mu') - \langle \mu', x_\theta \rangle$ and second order Taylor expansion of $f^*(\mu') = f^*(x'_\theta - (1-\theta)\delta')$ around x'_θ.

$\log a = a - 1$ (for a small), we have that:

$$\begin{aligned}
\ell_m(\theta) &= (1-\theta)\log[1+(1-\theta)\Delta] + \theta\log[1-(1-\theta)\Delta] && \text{(C.14)}\\
&\leq \log\{(1-\theta)[1+(1-\theta)\Delta] + \theta[1-(1-\theta)\Delta]\} && \text{(C.15)}\\
&= \log\left\{1+\left[(1-\theta)^2-\theta(1-\theta)\right]\Delta\right\} && \text{(C.16)}\\
&= \left[(1-\theta)^2-\theta(1-\theta)\right]\Delta \leq (1-\theta)^2\Delta && \text{(C.17)}
\end{aligned}$$

where (C.15) follow from Jensen inequality and (C.17) form the fact that $\theta(1-\theta) \geq 0$ for $\theta \in [0\,1]$. Since $d_f(x_\theta, \mu) \geq 0$ we also have $d_f(x_\theta, \mu) \geq \ell_m^+(\theta)$.

Next, we illustrate the *over-explorative* behavior of our Algorithm 6 when the approximations hold as lower bounds, i.e.:

$$\begin{aligned}
d_f(x_\theta, q) &\geq \ell_q^+(\theta), && \text{(C.18)}\\
d_f(x_\theta, \mu) &\geq \ell_m^+(\theta), && \text{(C.19)}\\
\mathscr{L}(\theta) &\geq \ell_{\mathscr{L}}(\theta) \equiv \ell_q^+(\theta) + \frac{\theta}{1-\theta}\left(\ell_m^+(\theta) - R\right) && \text{(C.20)}
\end{aligned}$$

In Step 4 of our algorithm when comparing $\ell_{\mathscr{L}}(\theta)$ to $c = d_f(x_c, q)$ (where x_c is the candidate NN), we can have the following situations:

- If $\mathscr{L}(\theta) \geq \ell_{\mathscr{L}} > c$ we *safely* prune the node;
- If $\mathscr{L}(\theta) > c > \ell_{\mathscr{L}}$ [loose lower bound] we *incorrectly* decide not to prune the node yet: we may end up exploring more and waste computation;
- If $c > \mathscr{L}(\theta) \geq \ell_{\mathscr{L}}$ [tight lower bound] we correctly not prune the node (yet).

For the rest of the comparisons (i.e., Steps 5, 6 and 7 of our algorithm), we can have:

- If $\ell_m < d_f(x_\theta, \mu) < R$ [tight lower bound]: we correctly assume x_θ is in the ball;
  - $\ell_q \leq d_f(x_\theta, q) \leq c$ [tight lower bound]: we *correctly* decide to explore (e.g., x_θ is better than candidate $\mathscr{X}_c$);
  - $\ell_q \leq c \leq d_f(x_\theta, q)$ [loose lower bound]: we *incorrectly* decide to explore — even if we might have ended up exploring anyway based on later iterations, in general we may waste computations;
  - $c \leq \ell_q \leq d_f(x_\theta, q)$: we *safely* update θ_l;
- If $\ell_m < R < d_f(x_\theta, \mu)$ [loose lower bound] we *incorrectly* assume x_θ is in the ball
  - $\ell_q \leq d_f(x_\theta, q) \leq c$ [tight lower bound]: we *incorrectly* decide to explore: wasteful;
  - $\ell_q \leq c \leq d_f(x_\theta, q)$ [loose lower bound]: we *incorrectly* decide to explore: wasteful;
  - $c \leq \ell_q \leq d_f(x_\theta, q)$: we *incorrectly* update θ_r instead of θ_l: this dilate the effective size of the ball and may incur in wasteful explorations;
- If $R < \ell_m < d_f(x_\theta, \mu)$ we *safely* assume x_θ is not the ball, and update θ_r.

On the opposite, when the query and the node are far away, the approximations can actually hold as *upper* bounds, which has the opposite effect of making the the algorithm under-explorative. Since the left-sided centroid $x_\theta = \arg\max_c \theta d_f(x, \mu) + (1-\theta) d_f(x, q)$ is zero forcing [119], it will have smaller support than the mixture model with modes μ and q, and consequently $d_f(x_\theta, \mu)$ (respectively, $d_f(x_\theta, q)$) will be smaller than $D(\Theta || \mathscr{M})$ (respectively, $D(\Theta || \mathscr{Q})$). If ℓ_q^+ and ℓ_m^+ are not tight, (C.10) and (C.11) will hold as upper bounds.

Bibliography

[1] *Dynamic Texture Recognition*, volume 2, Los Alamitos, CA, USA, 2001. IEEE Computer Society.

[2] A. Abdullah, J. Moeller, and S. Venkatasubramanian. Approximate Bregman near neighbors in sublinear time: Beyond the triangle inequality. *SCG*, 2012.

[3] Dimitris Achlioptas, Frank McSherry, and Bernhard Schölkopf. Sampling techniques for kernel methods. In *Advances in Neural Information Processing Systems 14*, volume 1, pages 335–342. MIT Press, 2002.

[4] A. Agarwal and B. Triggs. Tracking articulated motion using a mixture of autoregressive models. In *ECCV - Lecture notes in computer science*, pages 54–65, 2004.

[5] David Aldous, Illdar Ibragimov, and Jean Jacod. Exchangeability and related topics. volume 1117 of *Lecture Notes in Mathematics*, pages 1–198. Springer Berlin / Heidelberg, 1985.

[6] Jonathan Alon, Stan Sclaroff, George Kollios, and Vladimir Pavlovic. Discovering clusters in motion time-series data. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 368–375. IEEE, 2003.

[7] S. Amari. Information geometry and its applications: convex function and dually flat manifold. *Emerging Trends in Visual Computing*, 2009.

[8] J. Andén and S. Mallat. Multiscale scattering for audio classification. In *Proc. ISMIR*, 2011.

[9] Arthur Asuncion and David J Newman. UCI machine learning repository, 2010.

[10] Jean Aucouturier and François Pachet. Music similarity measures: What's the use? In *Proc. ISMIR*, pages 157–163, 2002.

[11] Claus Bahlmann and Hans Burkhardt. Measuring HMM similarity with the Bayes probability of error and its application to online handwriting recognition. In

Proceedings of the Sixth International Conference on Document Analysis and Recognition, pages 406–411. IEEE, 2001.

[12] Arindam Banerjee, Srujana Merugu, Inderjit S Dhillon, and Joydeep Ghosh. Clustering with Bregman divergences. *The Journal of Machine Learning Research*, 6:1705–1749, 2005.

[13] L. Barrington, A.B. Chan, and G. Lanckriet. Modeling music as a dynamic texture. *IEEE Transactions on Audio, Speech and Language Processing*, 18(3):602–612, 2010.

[14] L. Barrington, D. O'Malley, D. Turnbull, and G. Lanckriet. User-centered design of a social game to tag music. In *Proceedings of the ACM SIGKDD Workshop on Human Computation*, pages 7–10. ACM, 2009.

[15] L. Barrington, M. Yazdani, D. Turnbull, and G. Lanckriet. Combining feature kernels for semantic music retrieval. In *Proc. ISMIR*, pages 614–619, 2008.

[16] Luke Barrington, Reid Oda, and Gert RG Lanckriet. Smarter than genius? human evaluation of music recommender systems. In *ISMIR*, volume 9, pages 357–362, 2009.

[17] Eloi Batlle, Jaume Masip, and Enric Guaus. Automatic song identification in noisy broadcast audio. In *Proceeding of the International Conference on Signal and Image Processing*. IASTED, 2002.

[18] Jerome R Bellegarda and David Nahamoo. Tied mixture continuous parameter modeling for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(12):2033–2045, 1990.

[19] Yoshua Bengio, Jean-François Paiement, Pascal Vincent, Olivier Delalleau, Nicolas Le Roux, and Marie Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Advances in Neural Information Processing Systems*, volume 16, pages 177–184. MIT Press, 2004.

[20] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, pages 509–517, 1975.

[21] A. Berenzweig, B. Logan, P. W. Ellis, D., and B. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

[22] T. Bertin-Mahieux, D. Eck, F. Maillet, and P. Lamere. Autotagger: a model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research*, 37(2):115–135, June 2008.

[23] David M Blei and Michael I Jordan. Variational inference for Dirichlet process mixtures. *Bayesian Analysis*, 1(1):121–143, 2006.

[24] D.M. Blei and J.D. Lafferty. A correlated topic model of science. *The Annals of Applied Statistics*, 2007.

[25] M. Bregonzio, S. Gong, and T. Xiang. Recognizing action as clouds of space-time interest points. In *CVPR*. IEEE, 2009.

[26] William M Campbell. A SVM/HMM system for speaker recognition. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages II–209. IEEE, 2003.

[27] P. Cano and M. Koppenberger. Automatic sound annotation. In *Proc. IEEE Signal Processing Society Workshop on Machine Learning for Signal Processing.*, pages 391–400, 2004.

[28] Gustavo Carneiro, Antoni B Chan, Pedro J Moreno, and Nuno Vasconcelos. Supervised learning of semantic classes for image annotation and retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(3):394–410, March 2007.

[29] Michael Casey, Christophe Rhodes, and Malcolm Slaney. Analysis of minimum distances in high-dimensional musical spaces. *IEEE Transactions on Audio, Speech and Language Processing*, 16(5):1015–1028, 2008.

[30] L. Cayton. Efficient Bregman range search. *NIPS*, 2009.

[31] Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *ICML*, 2008.

[32] A. B. Chan and N. Vasconcelos. Modeling, clustering, and segmenting video with mixtures of dynamic textures. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(5):909–926, 2008.

[33] Antoni B Chan, Emanuele Coviello, and Gert RG Lanckriet. Clustering Dynamic Textures with the Hierarchical EM Algorithm. In *Proceeding of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 2010.

[34] Antoni B Chan, Emanuele Coviello, and Gert RG Lanckriet. Derivation of the hierarchical EM Algorithm for Dynamic Textures. Technical report, City University of Hong Kong, 2010.

[35] Antoni B. Chan and Nuno Vasconcelos. Probabilistic kernels for the classification of auto-regressive visual processes. In *Proc. IEEE CVRP*, Washington, DC, USA, 2005. IEEE Computer Society.

[36] C.C. Chang and C.J. Lin. Libsvm: a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27, 2011.

[37] S. Clifford. Pandora's long strange trip. *Inc.com*, 2007.

[38] Katrien De Cock, Katrien De Cock, Bart De Moor, and Bart De Moor. Subspace angles between linear stochastic models. In *Proc. IEEE CDC*, 2000.

[39] Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning sequence kernels. In *IEEE Workshop on Machine Learning for Signal Processing*, pages 2–8. IEEE, 2008.

[40] E. Coviello, L. Barrington, A.B. Chan, and G.R.G. Lanckriet. Automatic music tagging with time series models. In *Proceedings ISMIR*, 2010.

[41] Emanuele Coviello, Antoni Chan, and Gert Lanckriet. The variational hierarchical EM algorithm for clustering hidden Markov models. In *Advances in Neural Information Processing Systems 25*, pages 413–421, 2012.

[42] Emanuele Coviello, Antoni B Chan, and Gert RG Lanckriet. Time series models for semantic music annotation. *IEEE Transactions on Audio, Speech, and Language Processing*, 5(19):1343–1359, 2011.

[43] Emanuele Coviello, Antoni B. Chan, and Gert R.G. Lanckriet. Clustering hidden markov models with variational hem. *Journal of Machine Learning Research*, 15:697–747, 2014.

[44] Emanuele Coviello, Riccardo Miotto, and Gert RG Lanckriet. Combining content-based auto-taggers with decision-fusion. In *ISMIR*, pages 705–710, 2011.

[45] Emanuele Coviello, Adeel Mumtaz, Antoni B Chan, and Gert RG Lanckriet. Growing a Bag of Systems Tree for fast and accurate classification. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1979–1986. IEEE, 2012.

[46] Emanuele Coviello, Adeel Mumtaz, Antoni B Chan, and Gert RG Lanckriet. That was fast! speeding up nn search of high dimensional distributions. In *Proceedings of the 25th international conference on Machine learning*, 2013.

[47] Emanuele Coviello, Yonatan Vaizman, Antoni B Chan, and Gert RG Lanckriet. Multivariate autoregressive mixture models for music auto-tagging. In *ISMIR*, pages 547–552, 2012.

[48] Imre Csiszár and Gábor Tusnády. Information geometry and alternating minimization procedures. *Statistics and decisions*, 1984.

[49] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society B*, 39:1–38, 1977.

[50] K.G. Derpanis and R.P. Wildes. Dynamic texture recognition based on distributions of spacetime oriented structure. In *CVPR*, 2010.

[51] Inderjit S. Dhillon. Differential entropic clustering of multivariate Gaussians. In *Advances in Neural Information Processing Systems*, volume 19, page 337. MIT Press, 2007.

[52] M .A. Domingues, F. Gouyon, A. M. Jorge, J. P. Leal, J. Vinagre, L. Lemos, and M. Sordo. Combining usage and content in an online recommendation system for music in the long-tail. *International Journal of Multimedia Information Retrieval*, 1, In Press.

[53] G. Doretto and S. Soatto. Editable dynamic textures. In *Proc. IEEE CVRP*, volume 2, pages 137–142, June 2003.

[54] Gianfranco Doretto, Alessandro Chiuso, Ying Nian Wu, and Stefano Soatto. Dynamic textures. *International Journal on Computer Vision*, 51(2):91–109, 2003.

[55] Douglas Eck, Paul Lamere, Thierry Bertin-Mahieux, and Stephen Green. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems*, pages 385–392. MIT Press, 2008.

[56] K. Ellis, E. Coviello, and G. R. G. Lanckriet. Semantic annotation and retrieval of music using a bag of systems representation. In *Proc. ISMIR*, 2011.

[57] S. Essid, G. Richard, and B. David. Inferring efficient hierarchical taxonomies for MIR tasks: Application to musical instruments. In *Proc. ISMIR*, pages 324 –328, 2005.

[58] Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. Liblinear: A library for large linear classification. *Journal of Machine Learning Research*, 9:1871–1874, 2008.

[59] Daniel Fleder and Kartik Hosanagar. Blockbuster culture's next rise or fall: The impact of recommender systems on sales diversity. *Management science*, 55(5):697–712, 2009.

[60] A. Flexer, F. Gouyon, S. Dixon, and G. Widmer. Probabilistic combination of features for music classification. In *Proc. ISMIR*, pages 111–114, 2006.

[61] R Foucard, S Essid, and M Lagrange. Multi-scale temporal fusion by boosting for music classification. In *Proc. ISMIR*, pages 663–668, 2011.

[62] Charless Fowlkes, Serge Belongie, Fan Chung, and Jitendra Malik. Spectral grouping using the nyström method. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(2):214–225, 2004.

[63] J.H. Friedman, J.L. Bentley, and R.A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM TOMS*, 1977.

[64] Zhouyu Fu, Guojun Lu, Kai Ming Ting, and Dengsheng Zhang. Music classification via the bag-of-features approach. *Pattern Recognition Letters*, 32(14):1768 – 1777, 2011.

[65] A. Gelb. *Applied Optimal Estimation*. MIT Press, 1974.

[66] A. Gersho and R.M. Gray. *Vector quantization and signal compression*, volume 159. Springer Netherlands, 1992.

[67] A. Gilbert, J. Illingworth, and R. Bowden. Action recognition using mined hierarchical compound features. *IEEE TPAMI*, 2011.

[68] Jacob Goldberger and Sam Roweis. Hierarchical clustering of a mixture model. In *Advances in Neural Information Processing Systems*, pages 505–512. MIT Press, 2004.

[69] M. Goto and K. Hirata. Recent studies on music information processing. *Acoustical Science and Technology*, 25(6):419–425, 2004.

[70] K. Grauman and T. Darrell. Approximate correspondences in high dimensions. *NIPS*, 2007.

[71] Asela Gunawardana and William Byrne. Convergence theorems for generalized alternating minimization procedures. *Journal of Machine Learning Research*, 6:2049–73, 2005.

[72] Peter Hall, Joel L Horowitz, and Bing-Yi Jing. On blocking rules for the bootstrap with dependent data. *Biometrika*, 82(3):561–574, 1995.

[73] P. Hamel, S. Lemieux, Y. Bengio, and D. Eck. Temporal pooling and multiscale learning for automatic annotation and ranking of music audio. In *Proc ISMIR*, 2011.

[74] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Second edition, 2009.

[75] Trevor Hastie, Robert Tibshirani, Jerome Friedman, and James Franklin. The elements of statistical learning: data mining, inference and prediction. *The Mathematical Intelligencer*, 27(2):83–85, 2005.

[76] Christian Hennig. Cluster-wise assessment of cluster stability. *Computational Statistics & Data Analysis*, 52(1):258–271, 2007.

[77] John R Hershey and Peder A Olsen. Variational Bhattacharyya divergence for hidden Markov models. In *IEEE Computer Society International Conference on Acoustics, Speech and Signal Processing*, pages 4557–4560. IEEE, 2008.

[78] John R Hershey, Peder A Olsen, and Steven J Rennie. Variational Kullback-Leibler divergence for hidden Markov models. In *IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 323–328. IEEE, 2007.

[79] J.R. Hershey and P.A. Olsen. Approximating the Kullback Leibler divergence between Gaussian mixture models. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 4. Ieee, 2007.

[80] M. Hoffman, D. Blei, and P. Cook. Content-based musical similarity computation using the hierarchical Dirichlet process. In *Proc. ISMIR*, pages 349–354, 2008.

[81] Matthew D Hoffman, David M Blei, and Perry R Cook. Easy as CBA: A simple probabilistic model for tagging music. In *Proceedings of the 10th International Symposium on Music Information Retrieval*, pages 369–374, 2009.

[82] Xuedong D Huang and Mervyn A Jack. Semi-continuous hidden Markov models for speech signals. *Computer Speech & Language*, 3(3):239–251, 1989.

[83] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, 1985.

[84] Tommi S. Jaakkola. Tutorial on Variational Approximation Methods. In *Advanced Mean Field Methods: Theory and Practice*, pages 129–159. MIT Press, 2000.

[85] Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *The Journal of Machine Learning Research*, 5:819–844, 2004.

[86] Tony Jebara, Yingbo Song, and Kapil Thadani. Spectral clustering and embedding with hidden Markov models. In *Machine Learning: ECML 2007*, pages 164–175. 2007.

[87] Cyril Joder, Slim Essid, and GaËl Richard. Temporal integration for audio classification with application to musical instrument classification. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(1):174–186, 2009.

[88] Michael I Jordan, Zoubin Ghahramani, Tommi S Jaakkola, and Lawrence K Saul. An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233, 1999.

[89] Biing-Hwang Juang and Lawrence R Rabiner. A probabilistic distance measure for hidden Markov models. *AT&T Technical Journal*, 64(2):391–408, February 1985.

[90] Leonard Kaufman and Peter Rousseeuw. Clustering by means of medoids. *Statistical Data Analysis Based on the L1-Norm and Related Methods*, pages 405–416, 1987.

[91] Eamonn Keogh and Chotirat Ann Ratanamahatana. Exact indexing of dynamic time warping. *Knowledge and Information Systems*, 7(3):358–386, 2005.

[92] Eamonn J Keogh and Michael J Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 285–289. ACM, 2000.

[93] J. Kittler. Combining classifiers: A theoretical framework. *Pattern Analysis & Applications*, 1(1):18–27, 1998.

[94] Ariel Kleiner, Ameet Talwalkar, Purnamrita Sarkar, and Michael I Jordan. Bootstrapping big data. *Advances in Neural Information Processing Systems, Workshop: Big Learning: Algorithms, Systems, and Tools for Learning at Scale*, 2011.

[95] Anders Krogh, Michael Brown, I Saira Mian, Kimmen Sjölander, and David Haussler. Hidden Markov models in computational biology. Applications to protein modeling. *Journal of Molecular Biology*, 235(5):1501–1531, 1994.

[96] Pavel Kuksa, Pai-Hsi Huang, and Vladimir Pavlovic. Scalable algorithms for string kernels with inexact matching. In *Advances in Neural Information Processing Systems*, pages 881–888, 2008.

[97] P. Lamere and O. Celma. Music recommendation tutorial notes. *ISMIR Tutorial*, pages 2–19, September 2007.

[98] E. Law and L. Von Ahn. Input-agreement: a new mechanism for collecting data using human computation games. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1197–1206, 2009.

[99] H. Lee, Y. Largman, P. Pham, and A. Y. Ng. Unsupervised feature learning for audio classification using convolutional deep belief networks. In *Advances in Neural Information Processing Systems*, 2009.

[100] Christina Leslie, Eleazar Eskin, and William Stafford Noble. The spectrum kernel: A string kernel for SVM protein classification. In *Proceedings of the Pacific Symposium on Biocomputing*, volume 7, pages 566–575, 2002.

[101] M. Levy and M. Sandler. A semantic space for music derived from social tags. In *Proc. ISMIR*, pages 411–416, 2007.

[102] Marcus Liwicki and Horst Bunke. Iam-ondb-an on-line english sentence database acquired from handwritten text on a whiteboard. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 956–961. IEEE, 2005.

[103] B. Logan. Mel frequency cepstral coefficients for music modeling. In *Proc. ISMIR*, volume 28, 2000.

[104] Rune B Lyngso, Christian NS Pedersen, and Henrik Nielsen. Metrics and similarity measures for hidden Markov models. In *Proceedings International Conference on Intelligent Systems for Molecular Biology*, pages 178–186, 1999.

[105] David JC MacKay. *Information Theory, Inference and Learning Algorithms*. Cambridge university press, 2003.

[106] Michael I Mandel and Daniel PW Ellis. Multiple-instance learning for music information retrieval. In *Proceedings of the 9th International Symposium on Music Information Retrieval*, pages 577–582, 2008.

[107] M.F. McKinney and J. Breebaart. Features for audio and music classification. In *Proc. ISMIR*, pages 151 –158, 2003.

[108] A. Meng and J. Shawe-Taylor. An investigation of feature models for music genre classification using the support vector classifier. In *Proc. ISMIR*, pages 604–609, 2005.

[109] Nima Mesgarani, Malcolm Slaney, and Shihab A Shamma. Discrimination of speech from nonspeech based on multiscale spectro-temporal modulations. *Audio, Speech, and Language Processing, IEEE Transactions on*, 14(3):920–930, 2006.

[110] R. Miotto, L. Barrington, and G. Lanckriet. Improving auto-tagging by modeling semantic co-occurrences. In *Proc. ISMIR*, pages 297–302, 2010.

[111] A.W. Moore. The anchors hierarchy: Using the triangle inequality to survive high dimensional data. In *UAI*, 2000.

[112] Y. Mu and S. Yan. Non-metric locality-sensitive hashing. In *AAAI CAI*, 2010.

[113] A. Mumtaz, E. Coviello, G. Lanckriet, and A. Chan. Clustering Dynamic Textures with the Hierarchical EM Algorithm for Modeling Video. *IEEE TPAMI*, 2012.

[114] Adeel Mumtaz, Emanuele Coviello, Gert RG Lanckriet, and Antoni B Chan. Clustering dynamic textures with the hierarchical em algorithm for modeling video. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 35(7):1606–1621, 2013.

[115] Nag, Kin-Hong Wong, and Frank Fallside. Script recognition using hidden Markov models. In *IEEE Computer Society International Conference on Acoustics, Speech, and Signal Processing*, volume 11, pages 2071–2074. IEEE, 1986.

[116] Radford M Neal and Geoffrey E Hinton. A view of the EM algorithm that justifies incremental, sparse, and other variants. *Learning in Graphical Models*, 89:355–370, 1998.

[117] S.R. Ness, A. Theocharis, G. Tzanetakis, and L.G. Martins. Improving automatic music tag annotation using stacked generalization of probabilistic svm outputs. In *Proc. ACM MULTIMEDIA*, pages 705–708, 2009.

[118] A. Neumaier and T. Schneider. Estimation of parameters and eigenmodes of multivariate autoregressive models. *ACM Transactions on Mathematical Software (TOMS)*, 27(1):27–57, 2001.

[119] F. Nielsen and R. Nock. Sided and symmetrized Bregman centroids. *IEEE Transactions on IT*, 2009.

[120] F. Nielsen, P. Piro, and M. Barlaud. Bregman vantage point trees for efficient nearest neighbor queries. In *IEEE ICME*, 2009.

[121] F. Nielsen, P. Piro, M. Barlaud, et al. Tailored Bregman ball trees for effective nearest neighbors. In *EuroCG*, 2009.

[122] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *CVPR*, 2006.

[123] Evert J Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[124] Tim Oates, Laura Firoiu, and Paul R Cohen. Clustering time series with hidden markov models and dynamic time warping. In *Joint Conferences on Artificial Intelligence, Workshop on Neural, Symbolic and Reinforcement Learning Methods for Sequence Learning*, pages 17–21, 1999.

[125] S.M. Omohundro. *Five balltree construction algorithms*. International Computer Science Institute, 1989.

[126] E. Pampalk, A. Flexer, and G. Widmer. Improvements of audio-based music similarity and genre classification. In *Proc. ISMIR*, pages 628–633, 2005.

[127] Y. Panagakis, C. Kotropoulos, and G.R. Arce. Non-negative multilinear principal component analysis of auditory temporal modulations for music genre classification. *Audio, Speech, and Language Processing, IEEE Transactions on*, 18(3):576–588, 2010.

[128] Antonello Panuccio, Manuele Bicego, and Vittorio Murino. A hidden Markov model-based approach to sequential data clustering. *Structural, Syntactic, and Statistical Pattern Recognition*, pages 734–743, 2002.

[129] William D Penny and Stephen J Roberts. Notes on variational learning. Technical report, Oxford University, 2000.

[130] F. Pereira, N. Tishby, and L. Lee. Distributional clustering of English words. In *Association for Computational Linguistics*, 1993.

[131] Jose Costa Pereira, Emanuele Coviello, Gabriel Doyle, Nikhil Rasiwasia, Gert R.G. Lanckriet, Roger Levy, and Nuno Vasconcelos. On the role of correlation and abstraction in cross-modal multimedia retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(3):521–535, 2014.

[132] J. Philbin, O. Chum, M. Isard, J. Sivic, and A. Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *CVPR*, 2007.

[133] J. Puzicha, J.M. Buhmann, Y. Rubner, and C. Tomasi. Empirical evaluation of dissimilarity measures for color and texture. In *IEEE ICCV*, 1999.

[134] Yuting Qi, John William Paisley, and Lawrence Carin. Music analysis using hidden Markov mixture models. *IEEE Transactions on Signal Processing*, 55(11):5209–5224, 2007.

[135] Lawrence R Rabiner and Biing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice Hall, Upper Saddle River (NJ, USA), 1993.

[136] N. Rasiwasia, P.J. Moreno, and N. Vasconcelos. Bridging the gap: Query by semantic example. *IEEE Transactions on Multimedia*, 2007.

[137] N. Rasiwasia and N. Vasconcelos. Holistic context modeling using semantic co-occurences. In *Proc. IEEE CVPR*, pages 1889–1895, 2009.

[138] Nikhil Rasiwasia, Jose Costa Pereira, Emanuele Coviello, Gabriel Doyle, Gert RG Lanckriet, Roger Levy, and Nuno Vasconcelos. A new approach to cross-modal multimedia retrieval. In *Proceedings of the international conference on Multimedia*, pages 251–260. ACM, 2010.

[139] A. Ravichandran, R. Chaudhry, and R. Vidal. View-invariant dynamic texture recognition using a bag of dynamical systems. In *Proc. IEEE CVPR*, 2009.

[140] Jeremy Reed and Chin-Hui Lee. A study on music genre classification based on universal acoustic models. In *Proceedings of the 7th International Symposium on Music Information Retrieval*, pages 89–94, 2006.

[141] R.T. Rockafellar. *Convex analysis*. Princeton university press, 1996.

[142] Jose Rodriguez-Serrano and Florent Perronnin. A model-based sequence similarity with application to handwritten word-spotting. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34:2108 – 2120, 2012.

[143] C. Joder S. Essid and G. Richard. Temporal integration for audio classification with application to musical instrument classification. *IEEE Transactions on Audio, Speech, and Language Processing*, 17(1):174–186, 2009.

[144] Nicolas Scaringella and Giorgio Zoia. On the modeling of time information for automatic genre recognition systems in audio signals. In *Proc. 6th International Symposium Music Information Retrieval*, pages 666–671, 2005.

[145] Markus Schedl and Dominik Schnitzer. Hybrid Retrieval Approaches to Geospatial Music Recommendation. In *Proceedings of the 35th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR)*, Dublin, Ireland, July 31–August 1 2013.

[146] C. Schuldt, I. Laptev, and B. Caputo. Recognizing human actions: A local svm approach. In *ICPR 2004*. IEEE, 2004.

[147] R. H. Shumway and D. S. Stoffer. An approach to time series smoothing and forecasting using the EM algorithm. *Journal of Time Series Analysis*, 3(4):253–264, 1982.

[148] M. Slaney. Mixtures of probability experts for audio retrieval and indexing. In *Proc. IEEE Multimedia and Expo*, 2002.

[149] M. Slaney and M. Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE, SPM*, 2008.

[150] M. Slaney, K. Weinberger, and W. White. Learning a metric for music similarity. In *Proc. ISMIR*, pages 313–318, 2008.

[151] Padhraic Smyth. Clustering sequences with hidden Markov models. In *Advances in Neural Information Processing Systems*, pages 648–654. MIT Press, 1997.

[152] D.E. Sturim, DA Reynolds, E. Singer, and JP Campbell. Speaker indexing in large audio databases using anchor models. In *IEEE ICASSP*, pages 429–432, 2001.

[153] B.L. Sturm, Marcela Morvidone, and Laurent Daudet. Musical instrument identification using multiscale mel-frequency cepstral coefficients. In *Proc. EUSIPCO*, number 1, pages 477–481, 2010.

[154] M.J. Swain and D.H. Ballard. Color indexing. *International Journal of Computer Vision*, 7(1):11–32, 1991.

[155] Theodoros Theodoridis and Huosheng Hu. Action classification of 3d human models using dynamic ANNs for mobile robot surveillance. In *IEEE Computer Society International Conference on Robotics and Biomimetics*, pages 371–376. IEEE, 2007.

[156] Derek Tingle, Youngmoo E. Kim, and Douglas Turnbull. Exploring automatic music annotation with "acoustically-objective" tags. In *Proc. MIR*, pages 55–62, New York, NY, USA, 2010. ACM.

[157] B. Tomasik, J.H. Kim, M. Ladlow, M. Augat, D. Tingle, R. Wicentowski, and D. Turnbull. Using regression to combine data sources for semantic music discovery. In *Proc. ISMIR*, 2009.

[158] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Towards musical query-by-semantic description using the cal500 data set. *in Proc. ACM SIGIR*, 2007.

[159] Douglas Turnbull, Luke Barrington, David Torres, and Gert RG Lanckriet. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech and Language Processing*, 16(2):467–476, February 2008.

[160] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002.

[161] J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 1991.

[162] L. van der Maaten and G. Hinton. Visualizing data using t-sne. *J. Mach. Learn. Res.*, 9:2579–2605, 2008.

[163] Nuno Vasconcelos. Image indexing with mixture hierarchies. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2001.

[164] Nuno Vasconcelos and Andrew Lippman. Learning mixture hierarchies. In *Advances in Neural Information Processing Systems*, 1998.

[165] J. Stearns W. Glaser, T. Westergren and J. Kraft. Consumer item matching method and system. *US Patent Number 7003515*, 2006.

[166] Martin J Wainwright and Michael I Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[167] B. Whitman and D. Ellis. Automatic record reviews. In *Proc. ISMIR*, pages 470–477, 2004.

[168] B. Whitman and R. Rifkin. Musical query-by-description as a multi-class learning problem. In *Proc. IEEE Multimedia Signal Processing Conference*, December 2002.

[169] Ben H Williams, Marc Toussaint, and Amos J Storkey. Extracting motion primitives from natural handwriting data. In *Proceeding of the 16th International Conference on Artificial Neural Networks*, pages 634–643. Springer, 2006.

[170] Lior Wolf and Amnon Shashua. Learning over sets using kernel principal angles. *J. Mach. Learn. Res.*, 4:913–931, 2003.

[171] F. Woolfe and A. Fitzgibbon. Shift-invariant dynamic texture recognition. *Computer Vision–ECCV*, 2006.

[172] Bo Xie, Wei Bian, Dacheng Tao, and Parag Chordia. Music tagging with regularized logistic regression. In *ISMIR*, number Ismir, pages 711–716, 2011.

[173] M. Yang, F. Lv, W. Xu, K. Yu, and Y. Gong. Human action detection by boosting efficient motion features. In *ICCV*. IEEE, 2009.

[174] Y.H. Yang, Y.C. Lin, A. Lee, and H. Chen. Improving musical concept detection by ordinal regression and context fusion. In *Proc. ISMIR*, pages 147–152, 2009.

[175] Jie Yin and Qiang Yang. Integrating hidden Markov models and spectral analysis for sensory time series clustering. In *IEEE Computer Society International Conference on Data Mining*. IEEE, 2005.

[176] Z. Zhang, B.C. Ooi, S. Parthasarathy, and A.K.H. Tung. Similarity search on Bregman divergence: towards non-metric indexing. *VLDB E*, 2009.

[177] Guoying Zhao and M. Pietikainen. Dynamic texture recognition using local binary patterns with an application to facial expressions. *IEEE TPAMI*, 29(6):915 –928, june 2007.

[178] Shi Zhong and Joydeep Ghosh. A unified framework for model-based clustering. *The Journal of Machine Learning Research*, 4:1001–1037, 2003.