UNIVERSITY OF CALIFORNIA, SAN DIEGO

More like this: machine learning approaches to music similarity

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

in

Computer Science

by

Brian McFee

Committee in charge:

Professor Sanjoy Dasgupta, Co-Chair
Professor Gert Lanckriet, Co-Chair
Professor Serge Belongie
Professor Lawrence Saul
Professor Nuno Vasconcelos

2012

Copyright
Brian McFee, 2012
All rights reserved.

The dissertation of Brian McFee is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Co-Chair

University of California, San Diego

2012

DEDICATION

To my parents.

Thanks for the genes, and everything since.

EPIGRAPH

I'm gonna hear my favorite song,
if it takes all night.[1]

Frank Black, "If It Takes All Night."

[1] Clearly, the author is lamenting the inefficiencies of broadcast radio programming.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENTS

It is almost certainly impossible to spend the better part of a decade in one place and not rack up an extensive list of people who have made a positive impact in your life. (Even if it could be done, I can't imagine wanting to.) The following people in one way or another (and often several) have made my time at UCSD enjoyable, and have my endless gratitude.

First and foremost, thanks to my advisor, Gert Lanckriet, for teaching me more than I bargained for, and generally showing me how it's done.

Doug Turnbull showed me the fun side of research, and is a limitless supply of good advice. Thanks also to Luke Barrington, for balancing my rampant cynicism with good-natured optimism, and playing a mean trombone. Without these two, there's no telling where I would have wound up.

Thanks to Carolina Galleguillos, for having and knowing vision, and always finding a way to make things work.

Thanks to my committee: Serge Belongie, Sanjoy Dasgupta, Lawrence Saul, and Nuno Vasconcelos, who always manage to teach me something new.

The past and present residents of 41xx, especially Boris Babenko, Evan Ettinger, Daniel Hsu, Mayank Kabra, Samory Kpotufe, Matus Telgarsky, and Nakul Verma, kept me thinking and laughing the whole way through. Mostly laughing.

Steve Checkoway, Mike Stepp, Cynthia Taylor, Matt Tong, and Paul Ruvolo always reminded me that there's more to life than the lab. (Sorry about the annoy-a-tron, Steve.)

To Leo Porter, Ming Kawaguchi, Zach Tatlock, and Karyn Benson, thanks for keeping me running, figuratively and literally. Also thanks to Ross Tate for many a round of Tetris Attack.

Thanks to all members of the Computer Audition Lab, past and present: Emanuele Coviello, Kat Ellis, Andrew Hyunh, Janani Kalyanam, Omer Lang, and Daryl Lim.

Finally, thanks to my various collaborators: Noam Koenigstein, Yuval Shavitt, Thierry Bertin-Mahieux, Dan Ellis, Amélie Anglade, Òscar Celma, Ben Fields, and Paul Lamere. It's been fun every time.

Chapter 2, in full, is a reprint of material in the Journal of Machine Learning

Research, volume 12, pages 491–523, February, 2011, B. McFee and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 3, in full, is a reprint of material in the International Conference on Machine Learning, pages 775–782, 2010, B. McFee and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 5, in full, is a reprint of material that has been accepted for publication in the IEEE Transactions on Audio, Speech and Language Processing, 2012, B. McFee, L. Barrington, and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 6, in full, is a reprint of material in the International Society for Music Information Retrieval Conference, pages 55–60, 2011, B. McFee and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 7, in part, is a reprint of material in the International Society for Music Information Retrieval Conference, pages 537–541, 2011, B. McFee and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

Chapter 7, in part, is a reprint of material that has been submitted to the International Society for Music Information Retrieval Conference, 2012, B. McFee and G.R.G. Lanckriet. The dissertation author was the primary investigator and co-author of this paper.

VITA

2003	B. S. in Computer Science, University of California, Santa Cruz
2008	M. S. in Computer Science, University of California, San Diego
2012	Ph. D. in Computer Science, University of California, San Diego

PUBLICATIONS

B. McFee and G.R.G. Lanckriet. Partial order embedding with multple kernels. In *Proceedings of the 26th International Conference on Machine Learning*, ICML, pages 721–728, 2009b.

B. McFee and G.R.G. Lanckriet. Heterogeneous embeddding for subjective artist similarity. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009a.

B. McFee and G.R.G. Lanckriet. Metric learning to rank. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, ICML, pages 775–782, Haifa, Israel, June 2010.

C. Galleguillos, B. McFee, S. Belongie, and G.R.G. Lanckriet. Multi-class object localization by combining local contextual interactions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, pages 113–120, 2010.

B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning similarity from collaborative filters. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, ISMIR, 2010.

N. Koenigstein, G. R. G. Lanckriet, B. McFee, and Y. Shavitt. Collaborative filtering based on P2P networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, ISMIR, August 2010.

B. McFee and G.R.G. Lanckriet. Learning multi-modal similarity. *Journal of Machine Learning Research*, 12:491–523, February 2011.

B. McFee, C. Galleguillos, and G.R.G. Lanckriet. Contextual object localization with multiple kernel nearest neighbor. *IEEE Transactions on Image Processing*, 20(2):570–585, 2011. ISSN 1057-7149.

C. Galleguillos, B. McFee, S. Belongie, and G.R.G. Lanckriet. From region similarity to category discovery. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2011.

B. McFee and G.R.G. Lanckriet. Large-scale music similarity search with spatial trees. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 55–60, 2011a.

B. McFee and G.R.G. Lanckriet. The natural language of playlists. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 537–541, 2011b.

B. McFee, T. Bertin-Mahieux, D.P.W. Ellis, and G.R.G. Lanckriet. The million song dataset challenge. In *Proceedings of the 2nd International Workshop on Advances in Music Information Retrieval*, AdMIRe, 2012b.

B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning content similarity for music recommendation. *IEEE Transactions on Audio, Speech, and Language Processing*, 2012a. To appear.

ABSTRACT OF THE DISSERTATION

More like this: machine learning approaches to music similarity

by

Brian McFee

Doctor of Philosophy in Computer Science

University of California, San Diego, 2012

Professor Sanjoy Dasgupta, Co-Chair
Professor Gert Lanckriet, Co-Chair

The rise of digital music distribution has provided users with unprecedented access to vast song catalogs. In order to help users cope with large collections, music information retrieval systems have been developed to automatically analyze, index, and recommend music based on a user's preferences or search criteria.

This dissertation proposes machine learning approaches to content-based, query-by-example search, and investigates applications in music information retrieval. The proposed methods automatically infer and optimize content-based similarity, fuse heterogeneous feature modalities, efficiently index and search under the optimized distance metric, and finally, generate sequential playlists for a specified context or style. Robust

evaluation procedures are proposed to counteract issues of subjectivity and lack of explicit ground truth in music similarity and playlist generation.

Chapter 1

Introduction

1.1 Music information retrieval

Within the span of a decade, digital distribution has radically changed the ways in which people discover and interact with recorded music. In the past, the musical content available to a person was generally limited to a personal collection (perhaps a few thousand songs), or broadcast media (typically a few dozen stations in a given region). Now, online services such as iTunes,[1] Spotify,[2] Rdio,[3] and Mog[4] offer immediate access to tens of millions of songs at the click of a button. This rapid increase in the quantity of available songs can lead to severe *information overload*. Paradoxically, the overabundance of content can make it *more* difficult for a user to decide what to listen to next, and inhibit the discovery new music (Schwartz, 2004). This has motivated the development of *music information retrieval* (MIR) technology, which broadly aims to assist users in interacting with musical data.

At a high level, present industrial-strength MIR technology fall into two broad categories: those based on meta-data and semantic annotation, and those based on collaborative filtering.[5] Services based on semantic annotation, such as the Music Genome

[1] http://www.apple.com/itunes
[2] http://www.spotify.com
[3] http://www.rdio.com
[4] http://www.mog.com
[5] Of course, this is a vast over-simplification; successful systems in all likelihood combine a variety of techniques.

Project,[6] rely on human experts to annotate each song with musically relevant words. These semantic annotations can then be used to index, search, and recommend music. While highly accurate, this approach is expensive and labor-intensive, requiring a trained expert to spend up to 30 minutes to annotate a single song (Pandora Media, Inc., 2010).

By contrast, services based on collaborative filtering forego expensive (manual) analysis, and instead rely upon co-occurrence patterns in the purchasing (listening) histories of users to index music. Undoubtedly, the most famous example of a collaborative filtering recommendation engine is that of Netflix,[7] but iTunes, Amazon,[8] and Last.fm[9] are also known to use collaborative filtering to provide recommendations (Barrington et al., 2009).

For popular, commercially successful music, a wealth of information is available on the Internet, which can be used to represent and index musical content. This information can take a variety of forms, including lyrics, artist biography text, genre annotations, critical reviews, purchase data and chart ratings, *etc.*. When such *meta-data* is abundant and readily available, it may be indexed and searched by standard text-based information retrieval technology. Similarly, by its very definition, popular music tends to be consumed by a large number of users, ensuring good representation in a collaborative filter.

Unfortunately, collaborative filters do not degrade gracefully when faced with novel or less popular content. Descriptive information is harder to collect for unpopular items; in extreme cases, the only available content might be an audio file uploaded to an artist's profile on a social network. Again, by definition, unpopular items receive little representation in a collaborative filter. Both systems, therefore, are susceptible to the *cold start problem*: certain items will remain perpetually invisible to both the search or recommendation engine, and the end users.

This lack of high-quality meta-data and listening history for less-popular artists — the "long tail" (Anderson, 2006) — has motivated researchers to pursue *content-*

[6] http://www.pandora.com/about/mgp
[7] http://www.netflix.com
[8] http://www.amazon.com
[9] http://www.last.fm/

based information retrieval systems, which operate upon representations extracted directly from the music signal itself. A large component of the research on this topic attempts to automate the semantic annotation process by using machine learning algorithms to predict genre (Tzanetakis and Cook, 2002, McKinney and Breebaart, 2003) or more general textual descriptors (tags) (Barrington et al., 2007, Turnbull et al., 2008, Hoffman et al., 2009). A parallel line of work has focused on the problem of directly modeling similarity between songs (Aucouturier and Pachet, 2002, Ellis et al., 2002, Logan, 2004, Vignoli and Pauws, 2005, Pampalk et al., 2005, Flexer et al., 2008).

From a user's perspective, the interface can be just as important as the back-end technology. Numerous search interfaces have been proposed, starting from familiar text-based search for simple meta-data descriptors such as artist or song title, and ranging to more exotic interfaces, such as *query-by-humming*, *-beat-boxing*, *-tapping*, and *-keyboard* (Dannenberg et al., 2004, Kapur et al., 2004, Eisenberg et al., 2004, Typke, 2006). While the latter interfaces are perhaps more musically oriented, they require a non-trivial amount of musical skill on behalf of the user to be used effectively.

This dissertation investigates the *query-by-example* search paradigm for music information retrieval. Rather than force a user to express search criteria in vague or imprecise language — the canonical example being *rock* — an example-based music search allows the user to use known songs to query the database. This approach has numerous advantages: (1) it provides a simple, but powerful search interface for novice users, "find me songs that sound like ____"; (2) it obviates the need to define and educate users about a search vocabulary ; (3) the underlying methodology integrates well with a variety of musically relevant tasks, including recommendation and sequential playlist generation ; and (4) as will be demonstrated in this dissertation, the core similarity and retrieval functions are amenable to optimization by machine learning algorithms. Indeed, many successful online radio services are built around the query-by-example principle: Pandora[10] and Last.fm both allow users to seed a personalized radio station with one or more example songs and artists.

Ultimately, the quality of a query-by-example system depends on the ability to accurately determine similarity between the query and each item in the database. In the

[10] http://www.pandora.com

context of music information retrieval, this observation raises several natural questions:

- How should similarity between songs be computed?
- What is ground truth for evaluating music similarity?
- How can we take advantage of rich, multi-media content associated with music?
- Can similarity search be done efficiently, and scale to millions of songs?
- How can similarity search be integrated with common modes of music delivery, such as sequential playlist generation?

The goal of this dissertation is to address these questions, first by developing the core machine learning algorithms to optimize content-based similarity functions, and then by evaluating the resulting techniques in realistic application settings.

1.2 Summary of contributions

This dissertation is organized into two parts. The first part (chapters 2 to 4) develops the core machine learning framework for query-by-example information retrieval, while the second part (chapters 5 to 7) explores various applications in music similarity search and playlist generation.

Chapter 2 describes a multiple kernel framework to integrate descriptive features across multiple heterogeneous modalities, such as audio waveforms and plain text documents. In order to accomplish this task, chapter 2 also develops a graph-theoretic approach to cope with issues of label noise and subjectivity when processing human-generated similarity measurements between abstract objects (like songs or artists).

Chapter 3 develops an algorithm for learning a distance metric to optimize structured outputs, specifically, rankings induced by nearest-neighbor search. chapter 4 provides an improved, efficient optimization algorithm to solve the *structural metric learning* problem described in chapter 3.

Chapter 5 proposes an efficient and robust technique for assessing similarity between musical items through an implicit-feedback collaborative filter. This definition of

musical similarity is then used to train a content-based similarity metric via the algorithm described in chapter 3, which can be extended to out-of-sample, long-tail data.

Chapter 6 contains an evaluation of spatial data structures to accelerate content-based similarity search over a large music database of one million songs.

Finally, chapter 7 proposes an efficient and objective evaluation framework for playlist generation algorithms, and develops a generative playlist model based upon hyper-graph random walks. The proposed model robustly integrates heterogeneous features, and readily scales to hundreds of thousands of songs.

1.3 Preliminaries

Several concepts and notations are used throughout the dissertation. This section provides a brief summary of frequently used symbols and terms; infrequently used notation will be defined as needed in each chapter. Table 1.1 provides a brief summary of the remaining, standard notation and symbols used throughout the dissertation.

A great deal of the dissertation deals with defining and learning distances over vectors, which will typically be denoted in lower-case, *e.g.*, $x \in \mathbb{R}^d$ denotes a d-dimensional real-valued vector. Matrices will be denoted by upper-case letters, *e.g.*, $A \in \mathbb{R}^{d \times n}$ denotes a $d \times n$ matrix of reals, and A_i denotes the i^{th} column vector of A. A square, symmetric matrix $A \in \mathbb{S}^d$ has a *spectral decomposition*, denoted

$$A = V \Lambda V^{\mathsf{T}},$$

where each column of V is an eigenvector of A, and Λ is a diagonal matrix containing the eigenvalues λ_i of A. By convention, the eigenvalues (and corresponding eigenvectors) will be assumed to be arranged in non-increasing order: $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_d$.

Of particular interest are the $d \times d$ real symmetric matrices, denoted by $\mathbb{S}^d$, and the *positive semi-definite* matrices $\mathbb{S}^d_+$:

$$\mathbb{S}^d_+ := \left\{ W \;\middle|\; W \in \mathbb{S}^d, \forall x \in \mathbb{R}^d : \; x^{\mathsf{T}} W x \geq 0 \right\}.$$

A matrix $W \in \mathbb{S}^d_+$ defines a *Mahalanobis distance* between vectors $a, b \in \mathbb{R}^d$ (Maha-

lanobis, 1936):[11]

$$\|a - b\|_W := \sqrt{(a-b)^\intercal W (a-b)}.$$

Many of the methods described in this dissertation involve finding an optimum of a real-valued, (convex) function on a convex set $\mathcal{U}$, $f : \mathcal{U} \to \mathbb{R}$. In general, optima will be indicated by an over-line, *i.e.*, $\forall W \in \mathcal{U} : \; f\big(\overline{W}\big) \leq f(W)$. When a function f is differentiable, its gradient at W is denoted by $\nabla f(W)$. If f is convex, but not differentiable, $\partial f(W)$ will denote the *sub-differential* set at W:

$$\partial f(W) := \{\, v \mid \; \forall W' \in \mathcal{U} : \; f(W') \geq f(W) + \langle v, W' - W \rangle \,\},$$

and $\nabla f(W) \in \partial f(W)$ will indicate a *sub-gradient* at W.

For a point $x \in \mathcal{U}$ and a convex set $S \subseteq \mathcal{U}$, the *orthogonal* projection of x onto S is denoted by

$$\Pi_S[x] := \operatorname*{argmin}_{x' \in S} \|x - x'\|,$$

where the underlying distance metric $\|\cdot\|$ will be clear from context.

[11] More precisely, the distance defined by Mahalanobis (1936) uses the particular choice of $W = \Sigma^{-1}$ (the inverse covariance matrix), but the term is often broadly applied to a distance derived from any $W \in \mathbb{S}_+^d$.

Table 1.1: Symbols and notation used throughout this dissertation.

Symbol	Definition
$\mathbb{R}^d$ ($\mathbb{R}^d_+$)	d-dimensional, real-valued, (non-negative) vectors
$\mathbb{S}^d$ ($\mathbb{S}^d_+$)	$d \times d$ symmetric, real, (positive semi-definite) matrices
$\mathbf{0}, \mathbf{1}$	the all zeros or all ones vector
e_i	the i^{th} standard basis vector
A_i	the i^{th} column vector of a matrix A: $A\mathsf{e}_i$
$\langle a, b \rangle$	Euclidean inner product: $a^\mathsf{T} b = \sum_i a_i b_i$
$\langle A, B \rangle_\mathsf{F}$	Frobenius inner product: $\operatorname{tr}(A^\mathsf{T} B) = \sum_{i,j} A_{ij} B_{ij}$
$\|a - b\|_W$	Mahalanobis distance
$\Pi_S[x]$	the orthogonal projection of x onto a convex set S
$\partial f(W)$	the sub-differential set of a convex function f at W
$\nabla f(W)$	the gradient of a differentiable function $f(W)$
$\nabla f(W)$	a sub-gradient of f at W (element of $\partial f(W)$)
$[\![z]\!]$	0-1 indicator of the event z

Chapter 2

Learning multi-modal similarity

2.1 Introduction

In applications such as content-based recommendation systems, the definition of a proper similarity measure between items is crucial to many tasks, including nearest-neighbor retrieval and classification. In some cases, a natural notion of similarity may emerge from domain knowledge, for example, cosine similarity for bag-of-words models of text. However, in more complex, multi-media domains, there is often no obvious choice of similarity measure. Rather, viewing different aspects of the data may lead to several different, and apparently equally valid notions of similarity. For example, if the corpus consists of musical data, each song or artist may be represented simultaneously by acoustic features (such as rhythm and timbre), semantic features (tags, lyrics), or social features (collaborative filtering, artist reviews and biographies, *etc.*). Although domain knowledge may be incorporated to endow each representation with an intrinsic geometry—and, therefore, a sense of similarity—the different notions of similarity may not be mutually consistent. In such cases, there is generally no obvious way to combine representations to form a unified similarity space which optimally integrates heterogeneous data.

Without extra information to guide the construction of a similarity measure, the situation seems hopeless. However, if some side-information is available, for example, as provided by human labelers, it can be used to formulate a learning algorithm to optimize the similarity measure.

This idea of using side-information to optimize a similarity function has received a great deal of attention in recent years. Typically, the notion of similarity is captured by a distance metric over a vector space (*e.g.*, Euclidean distance in $\mathbb{R}^d$), and the problem of optimizing similarity reduces to finding a suitable embedding of the data under a specific choice of the distance metric. *Metric learning* methods, as they are known in the machine learning literature, can be informed by various types of side-information, including class labels (Xing et al., 2003, Goldberger et al., 2005, Globerson and Roweis, 2006, Weinberger et al., 2006), or binary *similar/dissimilar* pairwise labels (Wagstaff et al., 2001, Shental et al., 2002, Bilenko et al., 2004, Globerson and Roweis, 2007, Davis et al., 2007). Alternatively, multidimensional scaling (MDS) techniques are typically formulated in terms of quantitative (dis)similarity measurements (Torgerson, 1952, Kruskal, 1964, Cox and Cox, 1994, Borg and Groenen, 2005). In these settings, the representation of data is optimized so that distance (typically Euclidean) conforms to side-information. Once a suitable metric has been learned, similarity to new, unseen data can be computed either directly (if the metric takes a certain parametric form, for example, a linear projection matrix), or via out-of-sample extensions (Bengio et al., 2004).

To guide the construction of a similarity space for multi-modal data, we adopt the idea of using similarity measurements, provided by human labelers, as side-information. However, it has to be noted that, especially in heterogeneous, multi-media domains, similarity may itself be a highly subjective concept and vary from one labeler to the next (Ellis et al., 2002). Moreover, a single labeler may not be able to consistently decide if or to what extent two objects are similar, but she may still be able to reliably produce a rank-ordering of similarity over pairs (Kendall and Gibbons, 1990). Thus, rather than rely on quantitative similarity or hard binary labels of pairwise similarity, it is now becoming increasingly common to collect similarity information in the form of triadic or *relative* comparisons (Schultz and Joachims, 2004, Agarwal et al., 2007), in which human labelers answer questions of the form:

"Is x more similar to y or z?"

Although this form of similarity measurement has been observed to be more stable than quantitative similarity (Kendall and Gibbons, 1990), and clearly provides a richer representation than binary pairwise similarities, it is still subject to problems of consistency

and inter-labeler agreement. It is therefore imperative that great care be taken to ensure some sense of robustness when working with perceptual similarity measurements.

In the present work, our goal is to develop a framework for integrating multi-modal data so as to optimally conform to perceptual similarity encoded by relative comparisons. In particular, we follow three guiding principles in the development of our framework:

1. The algorithm should be robust against subjectivity and inter-labeler disagreement.

2. The algorithm must be able to integrate multi-modal data in an optimal way, that is, the distances between embedded points should conform to perceptual similarity measurements.

3. It must be possible to compute distances to new, unseen data as it becomes available.

We formulate this problem of heterogeneous feature integration as a learning problem: given a data set, and a collection of relative comparisons between pairs, we learn a representation of the data that optimally reproduces the similarity measurements. This type of embedding problem has been previously studied by Agarwal et al. (2007) and Schultz and Joachims (2004). However, Agarwal et al. (2007) provide no out-of-sample extension, and neither support heterogeneous feature integration, nor do they address the problem of noisy similarity measurements.

A common approach to optimally integrate heterogeneous data is based on *multiple kernel learning*, where each kernel encodes a different modality of the data. Heterogeneous feature integration via multiple kernel learning has been addressed by previous authors in a variety of contexts, including classification (Lanckriet et al., 2004, Zien and Ong, 2007, Kloft et al., 2009, Jagarlapudi et al., 2009), regression (Sonnenburg et al., 2006, Bach, 2008, Cortes et al., 2009), and dimensionality reduction (Lin et al., 2009). However, none of these methods specifically address the problem of learning a unified data representation which conforms to perceptual similarity measurements.

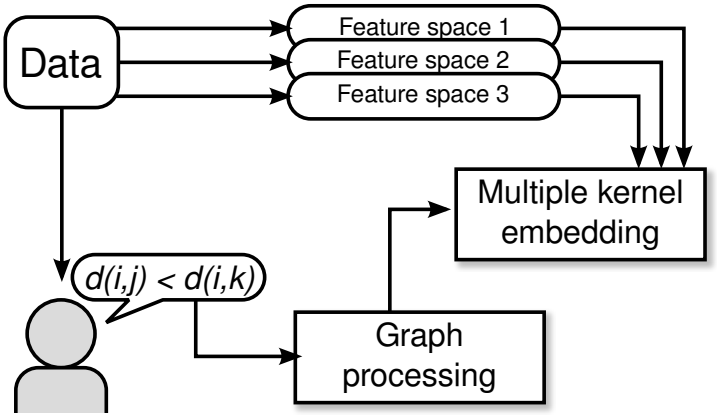


Figure 2.1: An overview of our proposed framework for multi-modal feature integration. Data is represented in multiple feature spaces, and humans supply perceptual similarity measurements in the form of relative pairwise comparisons , which processed and used as constraints to optimize the multiple kernel embedding.

2.1.1 Contributions

Our contributions in this chapter are two-fold. First, we develop the *partial order embedding* (POE) framework (McFee and Lanckriet, 2009b), which allows us to use graph-theoretic algorithms to filter a collection of subjective similarity measurements for consistency and redundancy. We then formulate a novel multiple kernel learning (MKL) algorithm which learns an ensemble of feature space projections to produce a unified similarity space. Our method is able to produce non-linear embedding functions which generalize to unseen, out-of-sample data. Figure 2.1 provides a high-level overview of the proposed methods.

The remainder of this chapter is structured as follows. In section 2.2, we develop a graphical framework for interpreting and manipulating subjective similarity measurements. In section 2.3, we derive an embedding algorithm which learns an optimal transformation of a single feature space. In section 2.4, we develop a novel multiple-kernel learning formulation for embedding problems, and derive an algorithm to learn an optimal space from heterogeneous data. Section 2.5 provides experimental results illustrating the effects of graph-processing on noisy similarity data, and the effectiveness of the multiple-kernel embedding algorithm on a music similarity task with human perception

measurements. Finally, we prove hardness of dimensionality reduction in this setting in section 2.6, and conclude in section 2.7.

2.1.2 Preliminaries

A *(strict) partial order* is a binary relation R over a set Z ($R \subseteq Z^2$) which satisfies the following properties:[1]

- Irreflexivity: $(a, a) \notin R$,
- Transitivity: $(a, b) \in R \wedge (b, c) \in R \Rightarrow (a, c) \in R$,
- Anti-symmetry: $(a, b) \in R \Rightarrow (b, a) \notin R$.

Every partial order can be equivalently represented as a directed acyclic graph (DAG), where each vertex is an element of Z and an edge is drawn from a to b if $(a, b) \in R$. For any partial order, R may refer to either the set of ordered tuples $\{ (a, b) \}$ or the graph (DAG) representation of the partial order; the use will be clear from context.

For a directed graph G, we denote by G^∞ its *transitive closure*, that is, G^∞ contains an edge (i, j) if and only if there exists a path from i to j in G. Similarly, the *transitive reduction* (denoted $G^{\min}$) is the minimal graph with equivalent transitivity to G, that is, the graph with the fewest edges such that $\left(G^{\min}\right)^\infty = G^\infty$.

Let $\mathcal{X} := \{ x_1, x_2, \ldots, x_n \}$ denote the training set of n items. A *Euclidean embedding* is a function $g : \mathcal{X} \to \mathbb{R}^d$ which maps $\mathcal{X}$ into a d-dimensional space equipped with the Euclidean (ℓ_2) metric:

$$\|x_i - x_j\|_2 := \sqrt{\langle x_i - x_j, x_i - x_j \rangle}.$$

For any matrix A, let A_i denote its i^{th} column vector. A symmetric matrix $A \in \mathbb{S}^n$ has a spectral decomposition $A = V \Lambda V^\mathsf{T}$, where $\Lambda := \operatorname{diag}(\lambda_1, \lambda_2, \ldots, \lambda_n)$ is a diagonal matrix containing the eigenvalues of A, and V contains the eigenvectors of A. We adopt the convention that eigenvalues (and corresponding eigenvectors) are sorted in descending order. W is *positive semi-definite* (PSD), denoted by $W \in \mathbb{S}^d_+$, if each

[1] The standard definition of a (non-strict) partial order also includes the *reflexive* property: $\forall a, (a, a) \in \mathbb{R}$. For reasons that will become clear in section 2.2, we take the *strict* definition here, and omit the reflexive property.

eigenvalue is non-negative: $\lambda_i \geq 0,\ i = 1, \ldots, d$. Finally, a PSD matrix W gives rise to the Mahalanobis distance function

$$\|x_i - x_j\|_W := \sqrt{(x_i - x_j)^\mathsf{T} W (x_i - x_i)}.$$

2.2 A graphical view of similarity

Before we can construct an embedding algorithm for multi-modal data, we must first establish the form of side-information that will drive the algorithm, that is, the similarity measurements that will be collected from human labelers. There is an extensive body of work on the topic of constructing a geometric representation of data to fit perceptual similarity measurements. Primarily, this work falls under the umbrella of multi-dimensional scaling (MDS), in which perceptual similarity is modeled by numerical responses corresponding to the perceived "distance" between a pair of items, for example, on a similarity scale of 1–10. (See Cox and Cox 1994 and Borg and Groenen 2005 for comprehensive overviews of MDS techniques.)

Because "distances" supplied by test subjects may not satisfy metric properties — in particular, they may not correspond to Euclidean distances — alternative *non-metric* MDS (NMDS) techniques have been proposed (Kruskal, 1964). Unlike classical or metric MDS techniques, which seek to preserve quantitative distances, NMDS seeks an embedding in which the rank-ordering of distances is preserved.

Since NMDS only needs the rank-ordering of distances, and not the distances themselves, the task of collecting similarity measurements can be simplified by asking test subjects to order pairs of points by similarity:

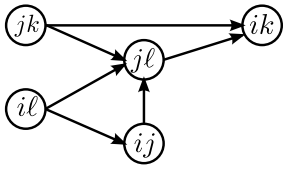
"Are i and j more similar than k and ℓ?"

or, as a special case, the "triadic comparison"

"Is i more similar to j or ℓ?"

Based on this kind of *relative comparison* data, the embedding problem can be formulated as follows. Given is a set of objects $\mathcal{X}$, and a set of similarity measurements

$$\mathcal{A} := \{\ (i, j, k, \ell)\ \} \subseteq \mathcal{X}^4,$$



$$\mathcal{A} = \left\{ \begin{array}{ll} (j,k,j,\ell), & (j,k,i,k), \\ (j,\ell,i,k), & (i,\ell,j,\ell), \\ (i,\ell,i,j), & (i,j,j,\ell) \end{array} \right\}$$

Figure 2.2: The graph representation (left) of a set of relative comparisons (right).

where a tuple (i,j,k,ℓ) is interpreted as "i and j are more similar than k and ℓ." (This formulation subsumes the triadic comparisons model when $i = k$.) The goal is to find an embedding function $g : \mathcal{X} \to \mathbb{R}^d$ such that

$$\forall (i,j,k,\ell) \in \mathcal{A} : \ \|g(i) - g(j)\|^2 + 1 < \|g(k) - g(\ell)\|^2. \tag{2.1}$$

The unit margin is forced between the constrained distances for numerical stability.

Agarwal et al. (2007) work with this kind of relative comparison data and describe a generalized NMDS algorithm (GNMDS), which formulates the embedding problem as a semi-definite program. Schultz and Joachims (2004) derive a similar algorithm which solves a quadratic program to learn a linear, axis-aligned transformation of data to fit relative comparisons.

Previous work on relative comparison data treats each measurement (i,j,k,ℓ) as effectively independent (Schultz and Joachims, 2004, Agarwal et al., 2007). However, due to their semantic interpretation as encoding pairwise similarity comparisons, and the fact that a pair (i,j) may participate in several comparisons with other pairs, there may be some *global* structure to $\mathcal{A}$ which these previous methods are unable to exploit.

In section 2.2.1, we develop a graphical framework to infer and interpret the global structure exhibited by the constraints of the embedding problem. Graph-theoretic algorithms presented in section 2.2.2 then exploit this representation to filter this collection of noisy similarity measurements for consistency and redundancy. The final, reduced set of relative comparison constraints defines a partial order, making for a more robust and efficient embedding problem.

2.2.1 Similarity graphs

To gain more insight into the underlying structure of a collection of comparisons $\mathcal{A}$, we can represent $\mathcal{A}$ as a directed graph over $\mathcal{X}^2$. Each vertex in the graph corresponds to a pair $(i, j) \in \mathcal{X}^2$, and an edge from (i, j) to (k, ℓ) corresponds to a similarity measurement (i, j, k, ℓ) (see fig. 2.2). Interpreting $\mathcal{A}$ as a graph will allow us to infer properties of *global* (graphical) structure of $\mathcal{A}$. In particular, two facts become immediately apparent:

1. If $\mathcal{A}$ contains cycles, then there exists no embedding which can satisfy $\mathcal{A}$.
2. If $\mathcal{A}$ is acyclic, any embedding that satisfies the transitive reduction $\mathcal{A}^{\min}$ also satisfies $\mathcal{A}$.

The first fact implies that no algorithm can produce an embedding which satisfies all measurements if the graph is cyclic. In fact, the converse of this statement is also true: if $\mathcal{A}$ is acyclic, then an embedding exists in which all similarity measurements are preserved (see section 2.A). If $\mathcal{A}$ is cyclic, however, by analyzing the graph, it is possible to identify an "unlearnable" subset of $\mathcal{A}$ which must be violated by any embedding.

Similarly, the second fact exploits the transitive nature of distance comparisons. In the example depicted in fig. 2.2, any g that satisfies (j, k, j, ℓ) and (j, ℓ, i, k) must also satisfy (j, k, i, k). In effect, the constraint (j, k, i, k) is redundant, and may also be safely omitted from $\mathcal{A}$.

These two observations allude to two desirable properties in $\mathcal{A}$ for embedding methods: *transitivity* and *anti-symmetry*. Together with irreflexivity, these fit the defining characteristics of a *partial order*. Due to subjectivity and inter-labeler disagreement, however, most collections of relative comparisons will not define a partial order. Some graph processing, presented next, based on an approximate maximum acyclic subgraph algorithm, can reduce them to a partial order.

2.2.2 Graph simplification

Because a set of similarity measurements $\mathcal{A}$ containing cycles cannot be embedded in any Euclidean space, $\mathcal{A}$ is inherently inconsistent. Cycles in $\mathcal{A}$ therefore

Algorithm 2.1 Approximate maximum acyclic subgraph (Aho et al., 1972)

Input: Directed graph $G = (V, E)$

Output: Acyclic graph $\overline{G}$

1: $\overline{E} \leftarrow \emptyset$
2: **for** each $(u, v) \in E$ in random order **do**
3: **if** $\overline{E} \cup \{ (u, v) \}$ is acyclic **then**
4: $\overline{E} \leftarrow \overline{E} \cup \{ (u, v) \}$
5: **end if**
6: **end for**
7: $\overline{G} \leftarrow (V, \overline{E})$

constitute a form of *label noise*. As noted by Angelova (2004), label noise can have adverse effects on both model complexity and generalization. This problem can be mitigated by detecting and pruning noisy (confusing) examples, and training on a reduced, but certifiably "clean" set (Angelova et al., 2005, Vezhnevets and Barinova, 2007).

Unlike most settings, where the noise process affects each label independently—for example, random classification noise (Angluin and Laird, 1988) — the graphical structure of interrelated relative comparisons can be exploited to detect and prune inconsistent measurements. By eliminating similarity measurements which cannot be realized by any embedding, the optimization procedure can be carried out more efficiently and reliably on a reduced constraint set.

Ideally, when eliminating edges from the graph, we would like to retain as much information as possible. Unfortunately, this is equivalent to the *maximum acyclic subgraph* problem, which is NP-complete (Garey and Johnson, 1979). A $1/2$-approximate solution can be achieved by a simple greedy algorithm (algorithm 2.1) (Berger and Shor, 1990).

Once a consistent subset of similarity measurements has been produced, it can be simplified further by pruning redundancies. In the graph view of similarity measurements, redundancies can be easily removed by computing the transitive reduction of the graph (Aho et al., 1972).

By filtering the constraint set for consistency, we ensure that embedding algo-

rithms are not learning from spurious information. Additionally, pruning the constraint set by transitive reduction focuses embedding algorithms on the most important core set of constraints while reducing overhead due to redundant information.

2.3 Partial order embedding

Now that we have developed a language for expressing similarity between items, we are ready to formulate the embedding problem. In this section, we develop an algorithm that learns a representation of data consistent with a collection of relative similarity measurements, and allows to map unseen data into the learned similarity space after learning. In order to accomplish this, we will assume a feature representation for $\mathcal{X}$. By parameterizing the embedding function g in terms of the feature representation, we will be able to apply g to any point in the feature space, thereby generalizing to data outside of the training set.

2.3.1 Linear projection

To start, we assume that the data originally lies in some Euclidean space, that is, $\mathcal{X} \subset \mathbb{R}^D$. There are of course many ways to define an embedding function $g : \mathbb{R}^D \to \mathbb{R}^d$. Here, we will restrict attention to embeddings parameterized by a linear projection matrix M, so that for a vector $x \in \mathbb{R}^D$,

$$g(x) := Mx.$$

Collecting the vector representations of the training set as columns of a matrix $X \in \mathbb{R}^{D \times n}$, the inner product matrix of the embedded points can be characterized as

$$A := X^\mathsf{T} M^\mathsf{T} M X.$$

Now, for a relative comparison (i, j, k, ℓ), we can express the distance constraint (eq. (2.1)) between embedded points as follows:

$$(X_i - X_j)^\mathsf{T} M^\mathsf{T} M (X_i - X_j) + 1 \leq (X_k - X_\ell)^\mathsf{T} M^\mathsf{T} M (X_k - X_\ell).$$

These inequalities can then be used to form the constraint set of an optimization problem to solve for an optimal $\overline{M}$. Because, in general, $\mathcal{A}$ may not be satisfiable by a linear

projection of $\mathcal{X}$, we soften the constraints by introducing a slack variable $\xi_{ijk\ell} \geq 0$ for each constraint, and minimize the empirical hinge loss over constraint violations $1/|\mathcal{A}| \sum_{\mathcal{A}} \xi_{ijk\ell}$. This choice of loss function can be interpreted as a convex approximation to a generalization of the area under an ROC curve (see section 2.C).

To avoid over-fitting, we introduce a regularization term $\mathrm{tr}(M^\mathsf{T} M) = \langle M, M \rangle_\mathsf{F}$, and a trade-off parameter $C > 0$ to control the balance between regularization and loss minimization. This leads to a regularized risk minimization objective:

$$\begin{aligned} &\min_{M, \xi \geq 0} \langle M, M \rangle_\mathsf{F} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} \xi_{ijk\ell} \qquad (2.2) \\ &\text{s. t.} \forall (i,j,k,\ell) \in \mathcal{A} : \\ &\quad (X_i - X_j)^\mathsf{T} M^\mathsf{T} M (X_i - X_j) + 1 \leq (X_k - X_\ell)^\mathsf{T} M^\mathsf{T} M (X_k - X_\ell) + \xi_{ijk\ell}. \end{aligned}$$

After learning M by solving this optimization problem, the embedding can be extended to out-of-sample points x' by applying the projection: $x' \mapsto Mx'$.

Note that the distance constraints in eq. (2.2) involve differences of quadratic terms, and are therefore not convex. However, since M only appears in the form $M^\mathsf{T} M$ in eq. (2.2), the optimization problem can be expressed in terms of a positive semi-definite matrix

$$W := M^\mathsf{T} M \in \mathbb{S}_+^D.$$

This change of variables results in algorithm 2.2, a (convex) semi-definite programming (SDP) problem (Boyd and Vandenberghe, 2004), since objective and constraints are linear in W, including the constraint $W \in \mathbb{S}_+^D$. The corresponding inner product matrix is

$$A = X^\mathsf{T} W X.$$

Finally, after the optimal $\overline{W}$ is found, the embedding function $g : \mathbb{R}^D \to \mathbb{R}^D$ can be recovered from the spectral decomposition of $\overline{W}$:

$$\overline{W} = V \Lambda V^\mathsf{T} \quad \Rightarrow \quad g(x) := \Lambda^{1/2} V^\mathsf{T} x,$$

and a d-dimensional approximation can be recovered by taking the leading d eigenvectors of $\overline{W}$.

Algorithm 2.2 Linear partial order embedding (LPOE)

Input: n objects $\mathcal{X}$, partial order $\mathcal{A}$, data matrix $X \in \mathbb{R}^{D \times n}$, $C > 0$

Output: mapping $g : \mathcal{X} \to \mathbb{R}^d$

$$\min_{W,\xi} \langle W, I \rangle_{\mathsf{F}} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} \xi_{ijk\ell}$$

$$\text{s.t.} \; \forall (i,j,k,\ell) \in \mathcal{A} : \quad \mathbf{d}(x_i, x_j) + 1 \leq \mathbf{d}(x_k, x_\ell) + \xi_{ijk\ell}$$

$$0 \leq \xi_{ijk\ell}$$

$$W \in \mathbb{S}_+^D$$

$$\mathbf{d}(x_i, x_j) \coloneqq (X_i - X_j)^{\mathsf{T}} W (X_i - X_j)$$

2.3.2 Non-linear projection via kernels

The formulation in algorithm 2.2 can be generalized to support non-linear embeddings by the use of kernels. Following the method of Globerson and Roweis (2007), we first map the data into a reproducing kernel Hilbert space (RKHS) $\mathcal{H}$ via a feature map ϕ with corresponding kernel function

$$k(x, y) \coloneqq \langle \phi(x), \phi(y) \rangle_{\mathcal{H}}.$$

Then, the data is mapped to $\mathbb{R}^d$ by a linear projection $M : \mathcal{H} \to \mathbb{R}^d$. The embedding function $g : \mathcal{X} \to \mathbb{R}^d$ is the therefore the composition of the projection M with ϕ:

$$g(x) = M(\phi(x)).$$

Because ϕ may be non-linear, this allows us to learn a non-linear embedding g.

More precisely, we consider M as being comprised of d elements of $\mathcal{H}$, that is,

$$M = (\omega_1, \omega_2, \ldots, \omega_d) \subseteq \mathcal{H}.$$

The embedding g can thus be expressed as

$$g(x) = \left(\langle \omega_p, \phi(x) \rangle_{\mathcal{H}} \right)_{p=1}^{d},$$

where $(\cdot)_{p=1}^{d}$ denotes concatenation.

Note that in general, $\mathcal{H}$ may be infinite-dimensional, so directly optimizing M may not be feasible. However, by appropriately regularizing M, we may invoke the generalized representer theorem (Schölkopf et al., 2001). Our choice of regularization is the Hilbert-Schmidt norm of M, which, in this case, reduces to

$$\|M\|_{\mathsf{HS}}^2 = \sum_{p=1}^{d} \langle \omega_p, \omega_p \rangle_{\mathcal{H}}.$$

With this choice of regularization, it follows from the generalized representer theorem that at an optimum, each $\overline{\omega}_p$ must lie in the span of the training data, that is,

$$\overline{\omega}_p = \sum_{i=1}^{n} \overline{N}_{pi} \phi(x_i), \quad p = 1, \ldots, d,$$

for some real-valued matrix $\overline{N} \in \mathbb{R}^{d \times n}$. If Φ is a matrix representation of $\mathcal{X}$ in $\mathcal{H}$ (*i.e.*, $\Phi_i = \phi(x_i)$ for $x_i \in \mathcal{X}$), then the optimal projection operator $\overline{M}$ can be expressed as

$$\overline{M} = \overline{N} \Phi^{\mathsf{T}}. \tag{2.3}$$

We can now reformulate the embedding problem as an optimization over N rather than M. Using eq. (2.3), the regularization term can be expressed as

$$\left\|\overline{M}\right\|_{\mathsf{HS}}^2 = \left\langle \overline{N}^{\mathsf{T}} \overline{N}, K \right\rangle_{\mathsf{F}},$$

where $K \in \mathbb{S}_+^n$ is the kernel matrix over $\mathcal{X}$ associated with $\mathcal{H}$:

$$K_{ij} := \langle \phi(x_i), \phi(x_j) \rangle_{\mathcal{H}} = k(x_i, x_j).$$

To formulate the distance constraints in terms of N, we first express the embedding g in terms of N and the kernel function:

$$g(x) = M(\phi(x)) = N\Phi^{\mathsf{T}}(\phi(x)) = N\left(\langle \Phi_i, \phi(x) \rangle_{\mathcal{H}}\right)_{i=1}^{n} = N\left(k(x_i, x)\right)_{i=1}^{n} = NK_x,$$

where K_x is the column vector formed by evaluating the kernel function k at x against the training set. The inner product matrix of embedded points can therefore be expressed as

$$A = KN^{\mathsf{T}}NK,$$

which allows to express the distance constraints in terms of N and the kernel matrix K:

$$(K_i - K_j)^\mathsf{T} N^\mathsf{T} N (K_i - K_j) + 1 \leq (K_k - K_\ell)^\mathsf{T} N^\mathsf{T} N (K_k - K_\ell).$$

The embedding problem thus amounts to solving the following optimization problem in N and ξ:

$$\begin{aligned} &\min_{N,\xi \geq 0} \left\langle N^\mathsf{T} N, K \right\rangle_\mathsf{F} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} \xi_{ijk\ell} \\ &\text{s. t.} \forall (i,j,k,\ell) \in \mathcal{A} : \\ &\qquad (K_i - K_j)^\mathsf{T} N^\mathsf{T} N (K_i - K_j) + 1 \leq (K_k - K_\ell)^\mathsf{T} N^\mathsf{T} N (K_k - K_\ell) + \xi_{ijk\ell}. \end{aligned} \tag{2.4}$$

Again, the distance constraints in eq. (2.4) are non-convex due to the differences of quadratic terms. And, as in the previous section, N only appears in the form of inner products $N^\mathsf{T} N$ in eq. (2.4) — both in the constraints, and in the regularization term — so we can again derive a convex optimization problem by changing variables to

$$W := N^\mathsf{T} N \in \mathbb{S}^n_+.$$

The resulting embedding problem is listed as algorithm 2.3, again a semi-definite programming problem (SDP), with an objective function and constraints that are linear in W.

After solving for $\overline{W}$, the embedding function $g(\cdot)$ can be recovered by computing the spectral decomposition $\overline{W} = V \Lambda V^\mathsf{T}$, and defining $\tilde{N} := \Lambda^{1/2} V^\mathsf{T}$. The resulting embedding function takes the form:

$$g(x) := \Lambda^{1/2} V^\mathsf{T} K_x.$$

As in Schultz and Joachims (2004), this formulation can be interpreted as learning a Mahalanobis distance metric $\Phi W \Phi^\mathsf{T}$ over $\mathcal{H}$. More generally, we can view this as a form of kernel learning, where the kernel matrix A is restricted to the set

$$A \in \left\{ KWK \;\middle|\; W \in \mathbb{S}^n_+ \right\}. \tag{2.5}$$

Algorithm 2.3 Kernel partial order embedding (KPOE)

Input: n objects $\mathcal{X}$, partial order $\mathcal{A}$, kernel matrix K, $C > 0$

Output: mapping $g : \mathcal{X} \to \mathbb{R}^n$

$$\min_{W,\xi} \langle W, K \rangle_{\mathsf{F}} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} \xi_{ijk\ell}$$

$$\text{s. t. } \forall (i,j,k,\ell) \in \mathcal{A}: \quad \mathbf{d}(x_i, x_j) + 1 \leq \mathbf{d}(x_k, x_\ell) + \xi_{ijk\ell}$$

$$0 \leq \xi_{ijk\ell}$$

$$W \in \mathbb{S}_+^n$$

$$\mathbf{d}(x_i, x_j) := (K_i - K_j)^{\mathsf{T}} W (K_i - K_j)$$

2.3.3 Connection to GNMDS

We conclude this section by drawing a connection between algorithm 2.3 and the generalized non-metric MDS (GNMDS) algorithm of Agarwal et al. (2007).

First, we observe that the i-th column, K_i, of the kernel matrix K can be expressed in terms of K and the i^{th} standard basis vector e_i:

$$K_i = K\mathsf{e}_i.$$

From this, it follows that distance computations in algorithm 2.3 can be equivalently expressed as

$$\begin{aligned} \mathbf{d}(x_i, x_j) &= (K_i - K_j)^{\mathsf{T}} W (K_i - K_j) \\ &= (K(\mathsf{e}_i - \mathsf{e}_j))^{\mathsf{T}} W (K(\mathsf{e}_i - \mathsf{e}_j)) \\ &= (\mathsf{e}_i - \mathsf{e}_j)^{\mathsf{T}} K^{\mathsf{T}} W K (\mathsf{e}_i - \mathsf{e}_j). \end{aligned} \tag{2.6}$$

If we consider the extremal case where $K = I$, that is, we have no prior feature-based knowledge of similarity between points, then eq. (2.6) simplifies to

$$\mathbf{d}(x_i, x_j) = (\mathsf{e}_i - \mathsf{e}_j)^{\mathsf{T}} I W I (\mathsf{e}_i - \mathsf{e}_j) = W_{ii} + W_{jj} - W_{ij} - W_{ji}.$$

Therefore, in this setting, rather than defining a feature transformation, W directly encodes the inner products between embedded training points. Similarly, the regularization

term becomes

$$\langle W, K \rangle_{\mathsf{F}} = \langle W, I \rangle_{\mathsf{F}} = \mathrm{tr}(W).$$

Minimizing the regularization term can be interpreted as minimizing a convex upper bound on the rank of W (Boyd and Vandenberghe, 2004), which expresses a preference for low-dimensional embeddings. Thus, by setting $K = I$ in algorithm 2.3, we directly recover the GNMDS algorithm.

Note that directly learning inner products between embedded training data points rather than a feature transformation does not allow a meaningful out-of-sample extension, to embed unseen data points. On the other hand, by eq. (2.5), it is clear that the algorithm optimizes over the entire cone of PSD matrices. Thus, if $\mathcal{A}$ defines a DAG, we could exploit the fact that a partial order over distances always allows an embedding which satisfies all constraints in $\mathcal{A}$ (see section 2.A) to eliminate the slack variables from the program entirely.

2.4 Multiple kernel embedding

In the previous section, we derived an algorithm to learn an optimal projection from a kernel space $\mathcal{H}$ to $\mathbb{R}^d$ such that Euclidean distance between embedded points conforms to perceptual similarity. If, however, the data is heterogeneous in nature, it may not be realistic to assume that a single feature representation can sufficiently capture the inherent structure in the data. For example, if the objects in question are images, it may be natural to encode texture information by one set of features, and color in another, and it is not immediately clear how to reconcile these two disparate sources of information into a single kernel space.

However, by encoding each source of information independently by separate feature spaces $\mathcal{H}^1, \mathcal{H}^2, \ldots, \mathcal{H}^m$ — equivalently, kernel matrices $K^1, K^2, \ldots, K^m$ — we can formulate a multiple kernel learning algorithm to optimally combine all feature spaces into a single, unified embedding space. In this section, we will derive a novel, projection-based approach to multiple-kernel learning and extend algorithm 2.3 to support heterogeneous data in a principled way.

2.4.1 Unweighted combination

Let $K^1, K^2, \ldots, K^m$ be a set of kernel matrices, each with a corresponding feature map ϕ^p and RKHS $\mathcal{H}^p$, for $p \in 1, \ldots, m$. One natural way to combine the kernels is to look at the product space, which is formed by concatenating the feature maps:

$$\phi(x_i) := \left(\phi^1(x_i), \phi^2(x_i), \ldots, \phi^m(x_i)\right) = (\phi^p(x_i))_{p=1}^m.$$

Inner products can be computed in this space by summing across each feature map:

$$\langle \phi(x_i), \phi(x_j) \rangle := \sum_{p=1}^m \langle \phi^p(x_i), \phi^p(x_j) \rangle_{\mathcal{H}^p}.$$

resulting in the *sum-kernel*—also known as the *average kernel* or *product space kernel*. The corresponding kernel matrix can be conveniently represented as the unweighted sum of the base kernel matrices:

$$\widehat{K} := \sum_{p=1}^m K^p. \tag{2.7}$$

Since $\widehat{K}$ is a valid kernel matrix itself, we could use $\widehat{K}$ as input for algorithm 2.3. As a result, the algorithm would learn a kernel from the family

$$\begin{aligned}\mathcal{K}_1 &:= \left\{ \left(\sum_{p=1}^m K^p\right) W \left(\sum_{p=1}^m K^p\right) \;\middle|\; W \in \mathbb{S}_+^n \right\} \\ &= \left\{ \sum_{p,q=1}^m K^p W K^q \;\middle|\; W \in \mathbb{S}_+^n \right\}.\end{aligned}$$

2.4.2 Weighted combination

Note that $\mathcal{K}_1$ treats each kernel equally; it is therefore impossible to distinguish *good* features (*i.e.*, those which can be transformed to best fit $\mathcal{A}$) from *bad* features, and as a result, the quality of the resulting embedding may be degraded. To combat this phenomenon, it is common to learn a scheme for weighting the kernels in a way which is optimal for a particular task. The most common approach to combining the base kernels is to take a positive-weighted sum

$$\widehat{K} := \sum_{p=1}^m \mu_p K^p \qquad \mu \in \mathbb{R}_+^m,$$

where the weights μ_p are learned in conjunction with a predictor (Lanckriet et al., 2004, Sonnenburg et al., 2006, Bach, 2008, Cortes et al., 2009). Equivalently, this can be viewed as learning a feature map

$$\phi(x_i) := \left(\sqrt{\mu_p}\phi^p(x_i)\right)_{p=1}^m,$$

where each base feature map has been scaled by the corresponding weight $\sqrt{\mu_p}$.

Applying this reasoning to learning an embedding that conforms to perceptual similarity, one might consider a two-stage approach to parameterizing the embedding (fig. 2.3a): first construct a weighted kernel combination, and then project from the combined kernel space. Lin et al. (2009) formulate a dimensionality reduction algorithm in this way. In the present setting, this would be achieved by simultaneously optimizing W and μ_p to choose an inner product matrix A from the set

$$\begin{aligned}\mathcal{K}_2 &:= \left\{ \left(\sum_{p=1}^m \mu_p K^p\right) W \left(\sum_{p=1}^m \mu_p K^p\right) \;\middle|\; W \in \mathbb{S}_+^n, \mu \in \mathbb{R}_+^m \right\} \\ &= \left\{ \sum_{p,q=1}^m \mu_p K^p W \mu_q K^q \;\middle|\; W \in \mathbb{S}_+^n, \mu \in \mathbb{R}_+^m \right\}.\end{aligned}$$

The corresponding distance constraints, however, contain differences of terms cubic in the optimization variables W and μ_p:

$$\sum_{p,q} \left(K_i^p - K_j^p\right)^\mathsf{T} \mu_p W \mu_q \left(K_i^q - K_j^q\right) + 1 \leq \sum_{p,q} \left(K_k^p - K_\ell^p\right)^\mathsf{T} \mu_p W \mu_q \left(K_k^q - K_\ell^q\right),$$

and are therefore non-convex and difficult to optimize. Even simplifying the class by removing cross-terms, that is, restricting A to the form

$$\mathcal{K}_3 := \left\{ \sum_{p=1}^m \mu_p^2 K^p W K^p \;\middle|\; W \in \mathbb{S}_+^n, \mu \in \mathbb{R}_+^m \right\},$$

still leads to a non-convex problem, due to the difference of positive quadratic terms introduced by distance calculations:

$$\sum_{p=1}^m \left(K_i^p - K_j^p\right)^\mathsf{T} \mu_p^2 W \left(\mu_p K_i^p - K_j^p\right) + 1 \leq \sum_{p=1}^m \left(K_k^p - K_\ell^p\right)^\mathsf{T} \mu_p^2 W \left(\mu_p K_k^p - K_\ell^p\right).$$

However, a more subtle problem with this formulation lies in the assumption that a single weight can characterize the contribution of a kernel to the optimal embedding. In

general, different kernels may be more or less informative on different subsets of $\mathcal{X}$ or different regions of the corresponding feature space. Constraining the embedding to a single metric W with a single weight μ_p for each kernel may be too restrictive to take advantage of this phenomenon.

2.4.3 Concatenated projection

We now return to the original intuition behind eq. (2.7). The sum-kernel represents the inner product between points in the space formed by concatenating the base feature maps ϕ^p. The sets $\mathcal{K}_2$ and $\mathcal{K}_3$ characterize projections of the weighted combination space, and turn out to not be amenable to efficient optimization (fig. 2.3a). This can be seen as a consequence of prematurely combining kernels prior to projection.

Rather than projecting the (weighted) concatenation of $\phi^p(\cdot)$, we could alternatively concatenate learned projections $M^p(\phi^p(\cdot))$, as illustrated by fig. 2.3b. Intuitively, by defining the embedding as the concatenation of m different projections, we allow the algorithm to learn an ensemble of projections, each tailored to its corresponding domain space and jointly optimized to produce an optimal space. By contrast, the previously discussed formulations apply essentially the same projection to each (weighted) feature space, and are thus much less flexible than our proposed approach. Mathematically, an embedding function of this form can be expressed as the concatenation

$$g(x) := \left(M^p\left(\phi^p(x)\right)\right)_{p=1}^{m}.$$

Now, given this characterization of the embedding function, we can adapt algorithm 2.3 to optimize over multiple kernels. As in the single-kernel case, we introduce regularization terms for each projection operator M^p

$$\sum_{p=1}^{m} \|M^p\|_{\mathsf{HS}}^2$$

to the objective function. Again, by invoking the representer theorem for each M^p, it follows that

$$\overline{M}^p = \overline{N}^p \left(\Phi^p\right)^{\mathsf{T}},$$

for some $\overline{N}^p \in \mathbb{R}^{d \times n}$, which allows to reformulate the embedding problem as a joint optimization over $N^p,\ p = 1, \ldots, m$ rather than $M^p,\ p = 1, \ldots, m$. Indeed, for optimal

$\overline{M}^p$, the regularization terms can be expressed as

$$\sum_{p=1}^{m} \|\overline{M}^p\|_{\mathsf{HS}}^2 = \sum_{p=1}^{m} \left\langle \overline{N}^{p\mathsf{T}} \overline{N}^p, K^p \right\rangle_{\mathsf{F}}. \tag{2.8}$$

The embedding function can now be rewritten as

$$g(x) = \left(M^p \left(\phi^p(x)\right)\right)_{p=1}^m = \left(N^p K_x^p\right)_{p=1}^m, \tag{2.9}$$

and the inner products between embedded points take the form:

$$\begin{aligned} A_{ij} = \langle g(x_i), g(x_j) \rangle &= \sum_{p=1}^{m} \left(N^p K_i^p\right)^{\mathsf{T}} \left(N^p K_j^p\right) \\ &= \sum_{p=1}^{m} (K_i^p)^{\mathsf{T}} (N^p)^{\mathsf{T}} (N^p) (K_j^p). \end{aligned}$$

Similarly, squared Euclidean distance also decomposes by kernel:

$$\|g(x_i) - g(x_j)\|^2 = \sum_{p=1}^{m} \left(K_i^p - K_j^p\right)^{\mathsf{T}} (N^p)^{\mathsf{T}} (N^p) \left(K_i^p - K_j^p\right). \tag{2.10}$$

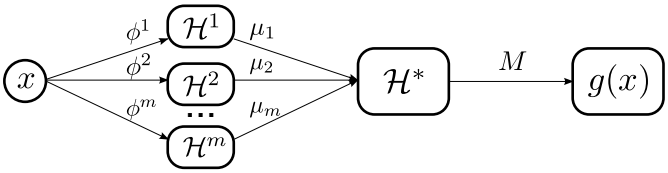
Finally, since the matrices $N^p,\ p = 1, \ldots, m$ only appear in the form of inner products in eqs. (2.8) and (2.10), we may instead optimize over PSD matrices $W^p := (N^p)^{\mathsf{T}}(N^p)$. This renders the regularization terms (eq. (2.8)) and distances (eq. (2.10)) linear in the optimization variables W^p. Extending algorithm 2.3 to this parameterization of $g(\cdot)$ therefore results in an SDP, which is listed as algorithm 2.4. To solve the SDP, we implemented a gradient descent solver, which is described in section 2.B.

The class of kernels over which algorithm 2.4 optimizes can be expressed simply as the set

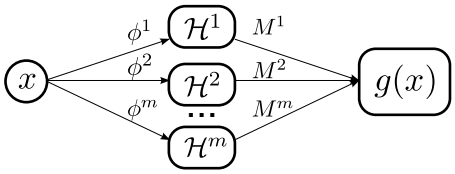
$$\mathcal{K}_4 := \left\{ \sum_{p=1}^{m} K^p W^p K^p \;\middle|\; \forall p,\ W^p \in \mathbb{S}_+^n \right\}.$$

Note that $\mathcal{K}_4$ contains $\mathcal{K}_3$ as a special case when all W^p are positive scalar multiples of each-other. However, $\mathcal{K}_4$ leads to a convex optimization problem, where $\mathcal{K}_3$ does not.

Table 2.1 lists the block-matrix formulations of each of the kernel combination rules described in this section. It is worth noting that it is certainly valid to first form the unweighted combination kernel $\widehat{K}$ and then use $\mathcal{K}_1$ (algorithm 2.3) to learn an optimal



(a) Weighted combination ($\mathcal{K}_2$)



(b) Concatenated projection ($\mathcal{K}_4$)

Figure 2.3: Two variants of multiple-kernel embedding. (a) $x \in \mathcal{X}$ is mapped into m feature spaces via $\phi^1, \phi^2, \ldots, \phi^m$, which are scaled by $\mu_1, \mu_2, \ldots, \mu_m$ to form $\mathcal{H}^*$ and finally projected via M. (b) x is first mapped into each kernel's feature space and then its images are directly projected via the corresponding projections M^p.

projection of the product space. However, as we will demonstrate in section 2.5, our proposed multiple-kernel formulation ($\mathcal{K}_4$) outperforms the simple unweighted combination rule in practice.

2.4.4 Diagonal learning

The MKPOE optimization is formulated as a semi-definite program over m different $n \times n$ matrices W^p — or, as shown in table 2.1, a single $mn \times mn$ PSD matrix with a block-diagonal sparsity structure. Scaling this approach to large data sets can become problematic, as they require optimizing over multiple high-dimensional PSD matrices.

To cope with larger problems, the optimization problem can be refined to constrain each W^p to the set of diagonal matrices. If W^p are all diagonal, positive semi-definiteness is equivalent to non-negativity of the diagonal values (since they are also the eigenvalues of the matrix). This allows the constraints $W^p \in \mathbb{S}^n_+$ to be replaced by linear constraints $\mathrm{diag}(W^p) \in \mathbb{R}^n_+$, and the resulting optimization problem is a linear program (LP), rather than an SDP. This modification reduces the flexibility of the model, but leads to a much more efficient optimization procedure.

Table 2.1: Block-matrix formulations of metric learning for multiple-kernel formulations ($\mathcal{K}_1$–$\mathcal{K}_4$). Each W^p is taken to be positive semi-definite. Note that all sets are equal when there is only one base kernel.

Kernel class	Learned kernel matrix
$\mathcal{K}_1 = \left\{ \sum_{p,q} K^p W K^q \right\}$	$[K^1 + K^2 + \cdots + K^m]\,[W]\,[K^1 + K^2 + \cdots + K^m]$
$\mathcal{K}_2 = \left\{ \sum_{p,q} \mu_p \mu_q K^p W K^q \right\}$	$\begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mu_1^2 W & \mu_1\mu_2 W & \cdots & \mu_1\mu_m W \\ \mu_2\mu_1 W & \mu_2^2 W & \cdots & \vdots \\ \vdots & & \ddots & \\ \mu_m\mu_1 W & & & \mu_m^2 W \end{bmatrix} \begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}$
$\mathcal{K}_3 = \left\{ \sum_{p} \mu_p^2 K^p W K^p \right\}$	$\begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}^\mathsf{T} \begin{bmatrix} \mu_1^2 W & 0 & \cdots & 0 \\ 0 & \mu_2^2 W & \cdots & \vdots \\ \vdots & & \ddots & \\ 0 & & & \mu_m^2 W \end{bmatrix} \begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}$
$\mathcal{K}_4 = \left\{ \sum_{p} K^p W^p K^p \right\}$	$\begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}^\mathsf{T} \begin{bmatrix} W^1 & 0 & \cdots & 0 \\ 0 & W^2 & \cdots & \vdots \\ \vdots & & \ddots & \\ 0 & & & W^m \end{bmatrix} \begin{bmatrix} K^1 \\ K^2 \\ \vdots \\ K^m \end{bmatrix}$

Algorithm 2.4 Multiple kernel partial order embedding (MKPOE)

Input: n objects $\mathcal{X}$, partial order $\mathcal{A}$, m kernel matrices $K^1, K^2, \ldots, K^m$, $C > 0$

Output: mapping $g : \mathcal{X} \to \mathbb{R}^{mn}$

$$\min_{W^p, \xi} \sum_{p=1}^{m} \langle W^p, K^p \rangle_{\mathsf{F}} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} \xi_{ijk\ell}$$

$$\text{s.t.}\ \forall (i,j,k,\ell) \in \mathcal{A}: \quad \mathbf{d}(x_i, x_j) + 1 \leq \mathbf{d}(x_k, x_\ell) + \xi_{ijk\ell}$$

$$0 \leq \xi_{ijk\ell}$$

$$\forall p \in \{\, 1, 2, \cdots, m \,\}: \quad W^p \in \mathbb{S}^n_+$$

$$\mathbf{d}(x_i, x_j) := \sum_{p=1}^{m} \left(K^p_i - K^p_j\right)^{\mathsf{T}} W^p \left(K^p_i - K^p_j\right)$$

More specifically, our implementation of algorithm 2.4 operates by alternating sub-gradient descent on W^p and projection onto the feasible set $\mathbb{S}^n_+$ (see section 2.B for details). For full matrices, this projection is accomplished by computing the spectral decomposition of each W^p, and thresholding the eigenvalues at 0. For diagonal matrices, this projection is accomplished simply by

$$W^p_{ii} \mapsto \max\{\, 0, W^p_{ii} \,\},$$

which can be computed in $\mathcal{O}(mn)$ time, compared to the $\mathcal{O}(mn^3)$ time required to compute m spectral decompositions.

Restricting W^p to be diagonal not only simplifies the problem to linear programming, but carries the added interpretation of weighting the contribution of each (kernel, training point) pair in the construction of the embedding. A large value at W^p_{ii} corresponds to point i being a landmark for the features encoded in K^p. Note that each of the formulations listed in table 2.1 has a corresponding diagonal variant, however, as in the full matrix case, only $\mathcal{K}_1$ and $\mathcal{K}_4$ lead to convex optimization problems.

2.5 Experiments

To evaluate our framework for learning multi-modal similarity, we first test the multiple kernel learning formulation on a simple toy taxonomy data set, and then on a real-world data set of musical perceptual similarity measurements.

2.5.1 Toy experiment: taxonomy embedding

For our first experiment, we generated a toy data set from the Amsterdam Library of Object Images (ALOI) data set (Geusebroek et al., 2005). ALOI consists of RGB images of 1000 classes of objects against a black background. Each class corresponds to a single object, and examples are provided of the object under varying degrees of out-of-plane rotation.

In our experiment, we first selected 10 object classes, and from each class, sampled 20 examples. We then constructed an artificial taxonomy over the label set, as depicted in fig. 2.4. Using the taxonomy, we synthesized relative comparisons to span subtrees via their least common ancestor. For example,

$$(Lemon\,\#1,\ Lemon\,\#2,\ Lemon\,\#1,\ Pear\#1),$$
$$(Lemon\,\#1,\ Pear\#,1,\ Lemon\,\#1,\ Sneaker\#1),$$

and so on. These comparisons are consistent and therefore can be represented as a directed acyclic graph. They are generated so as to avoid redundant, transitive edges in the graph.

For features, we generated five kernel matrices. The first is a simple linear kernel over the gray-scale intensity values of the images, which, roughly speaking, compares objects by shape. The other four are Gaussian kernels over histograms in the (background-subtracted) red, green, blue, and intensity channels, and these kernels compare objects based on their color or intensity distributions.

We augment this set of kernels with five "noise" kernels, each of which was generated by sampling random points from the unit sphere in $\mathbb{R}^3$ and applying the linear kernel.

The data was partitioned into five 80/20 training and test set splits. To tune C, we further split the training set for 5-fold cross-validation, and swept over $C \in$

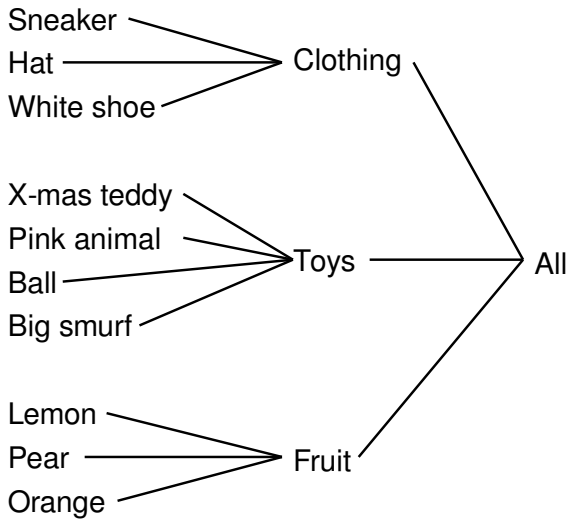


Figure 2.4: The label taxonomy for the experiment in section 2.5.1.

$\{ 10^{-2}, 10^{-1}, \ldots, 10^{6} \}$. For each fold, we learned a diagonally-constrained embedding with algorithm 2.4, using the subset of relative comparisons (i, j, k, ℓ) with i, j, k and ℓ restricted to the training set. After learning the embedding, the held out data (validation or test) was mapped into the space, and the accuracy of the embedding was determined by counting the fraction of correctly predicted relative comparisons. In the validation and test sets, comparisons were processed to only include comparisons of the form (i, j, i, k) where i belongs to the validation (or test) set, and j and k belong to the training set.

We repeat this experiment for each base kernel individually (that is, optimizing over $\mathcal{K}_1$ with a single base kernel), as well as the unweighted sum kernel ($\mathcal{K}_1$ with all base kernels), and finally MKPOE ($\mathcal{K}_4$ with all base kernels). The results are averaged over all training/test splits, and collected in fig. 2.5. For comparison purposes, we include the prediction accuracy achieved by computing distances in each kernel's native space before learning. In each case, the optimized space indeed achieves higher accuracy than the corresponding native space. (Of course, the random noise kernels still predict randomly after optimization.)

As illustrated in fig. 2.5, taking the unweighted combination of kernels significantly degrades performance (relative to the best kernel) both in the native space (0.718 accuracy versus 0.862 for the linear kernel) and the optimized sum-kernel space (0.861

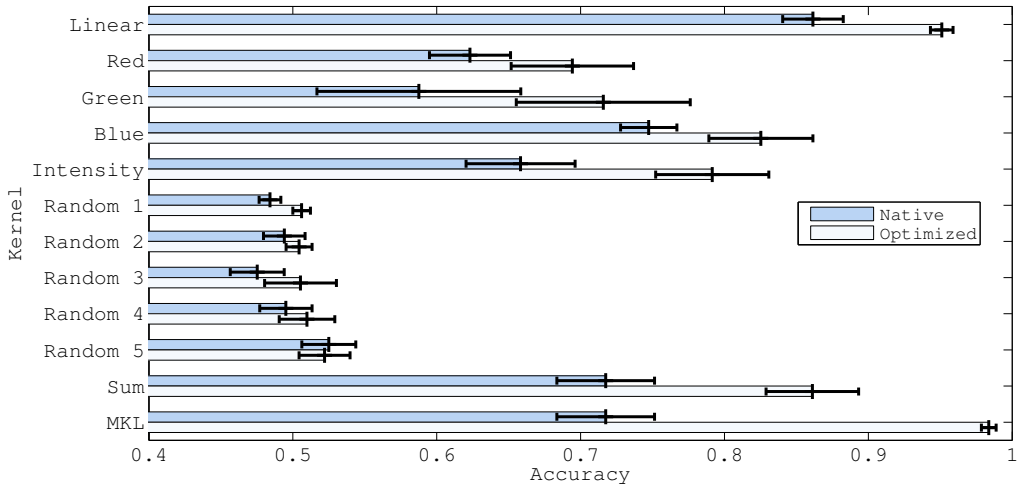


Figure 2.5: Test accuracy for the taxonomy embedding experiment: the fraction of correctly predicted comparisons in the native space, and in the optimized space produced by KPOE (single kernel) and MKPOE (multiple kernel). Error bars correspond to one standard deviation across folds.

accuracy for the sum versus 0.951 for the linear kernel), that is, the unweighted sum kernel optimized by algorithm 2.3. However, MKPOE ($\mathcal{K}_4$) correctly identifies and omits the random noise kernels by assigning them negligible weight, and achieves higher accuracy (0.984) than any of the single kernels (0.951 for the linear kernel, after learning).

2.5.2 Musical artist similarity

To test our framework on a real data set, we applied the MKPOE algorithm to the task of learning a similarity function between musical artists. The artist similarity problem is motivated by several real-world applications, including recommendation and playlist-generation for online radio. Because artists may be represented by a wide variety of different features (*e.g.*, tags, acoustic features, social data), such applications can benefit greatly from an optimally integrated similarity metric.

The training data is derived from the *aset400* corpus of Ellis et al. (2002), which consists of 412 popular musicians and 16385 relative comparisons of the form (i, j, i, k). Relative comparisons were acquired from human test subjects through a web survey;

subjects were presented with a query artist (i), and asked to choose what they believe to be the most similar artist (j) from a list of 10 candidates. From each single response, 9 relative comparisons are synthesized, indicating that j is more similar to i than the remaining 9 artists (k) which were not chosen.

Our experiments here replicate and extend previous work on this data set (McFee and Lanckriet, 2009a). In the remainder of this section, we will first give an overview of the various types of features used to characterize each artist in section 2.5.2. We will then discuss the experimental procedure in more detail in section 2.5.2. The MKL embedding results are presented in section 2.5.2, and are followed by an experiment detailing the efficacy of our constraint graph processing approach in section 2.5.2.

Features

We construct five base kernels over the data, incorporating acoustic, semantic, and social views of the artists.

MFCC for each artist, we collected between 1 and 10 songs (mean 4). For each song, we extracted a short clip consisting of 10000 half-overlapping 23ms windows. For each window, we computed the first 13 Mel Frequency Cepstral Coefficients (MFCCs) (Davis and Mermelstein, 1990), as well as their first and second instantaneous derivatives. This results in a sequence of 39-dimensional vectors (delta-MFCCs) for each song. Each artist i was then summarized by a Gaussian mixture model (GMM) p_i over delta-MFCCs extracted from the corresponding songs. Each GMM has 8 components and diagonal covariance matrices. Finally, the kernel between artists i and j is the probability product kernel (Jebara et al., 2004) between their corresponding delta-MFCC distributions p_i, p_j:

$$K_{ij}^{\text{mfcc}} := \int \sqrt{p_i(x) p_j(x)}\, \mathrm{d}x.$$

Auto-tags (AT) Using the MFCC features described above, we applied the automatic tagging algorithm of Turnbull et al. (2008), which for each song yields a multinomial distribution over a set T of 149 musically-relevant tag words (*auto-tags*). Artist-level tag distributions q_i were formed by averaging model parameters (*i.e.*,

tag probabilities) across all of the songs of artist i. The kernel between artists i and j for auto-tags is a radial basis function applied to the χ^2-distance between the multinomial distributions q_i and q_j:

$$K_{ij}^{\text{at}} := \exp\left(-\sigma \sum_{t \in T} \frac{\left(q_i(t) - q_j(t)\right)^2}{q_i(t) + q_j(t)}\right).$$

In these experiments, we fixed $\sigma = 256$.

Social tags (ST) For each artist, we collected the top 100 most frequently used tag words from Last.fm,[2] a social music website which allows users to label songs or artists with arbitrary tag words or *social tags*. After stemming and stop-word removal, this results in a vocabulary of 7737 tag words. Each artist is then represented by a bag-of-words vector in $\mathbb{R}^{7737}$, and processed by TF-IDF. The kernel between artists for social tags is the cosine similarity (linear kernel) between TF-IDF vectors.

Biography (Bio) Last.fm also provides textual descriptions of artists in the form of user-contributed biographies. We collected biographies for each artist in *aset400*, and after stemming and stop-word removal, we arrived at a vocabulary of 16753 biography words. As with social tags, the kernel between artists is the cosine similarity between TF-IDF bag-of-words vectors.

Collaborative filter (CF) Celma (2010) collected collaborative filter data from Last.fm in the form of a bipartite graph over users and artists, where each user is associated with the artists in her listening history. We filtered this data down to include only the aset400 artists, of which all but 5 were found in the collaborative filtering graph. The resulting graph has 336527 users and 407 artists, and is equivalently represented by a binary matrix F where each row u corresponds to a user, and each column i corresponds to an artist. F_{ui} is 1 if we observe a user-artist association, and 0 otherwise. The kernel between artists in this view is the cosine of the angle between corresponding columns in the matrix, which can be interpreted as counting the amount of overlap between the sets of users listening to each artist

[2] Last.fm can be found at http://last.fm.

and normalizing for overall artist popularity. For the 5 artists not found in the graph, we fill in the corresponding rows and columns of the kernel matrix with the identity matrix.

Experimental procedure

The data was randomly partitioned into ten 90/10 training/test splits. Given the inherent ambiguity in the task, and format of the survey, there is a great deal of conflicting information in the survey responses. To obtain a more accurate and internally consistent set of training comparisons, directly contradictory comparisons (*e.g.*, (i, j, i, k) and (i, k, i, j)) were removed from both the training and test sets. Each training set was further cleaned by finding an acyclic subset of comparisons and taking its transitive reduction, resulting in a minimal partial order. (No further processing was performed on test comparisons.)

After training, test artists were mapped into the learned space (by eq. (2.9)), and accuracy was measured by counting the number of measurements (i, j, i, k) correctly predicted by distance in the learned space, where i belongs to the test set, and j, k belong to the training set.

For each experiment, C is chosen from $\{ 10^{-2}, 10^{-1}, \ldots, 10^{7} \}$ by holding out 30% of the training constraints for validation. (Validation splits are generated from the unprocessed training set, and the remaining training constraints are processed as described above.) After finding the best-performing C, the embedding is trained on the full (processed) training set.

Embedding results

For each base kernel, we evaluate the test-set performance in the native space (*i.e.*, by distances calculated directly from the entries of the kernel matrix), and by learned metrics, both diagonal and full (optimizing over $\mathcal{K}_1$ with a single base kernel). Figure 2.6 illustrates the results.

We then repeated the experiment by examining different groupings of base kernels: acoustic (MFCC and Auto-tags), semantic (Social tags and Bio), social (Collaborative filter), and combinations of the groups. The different sets of kernels were combined

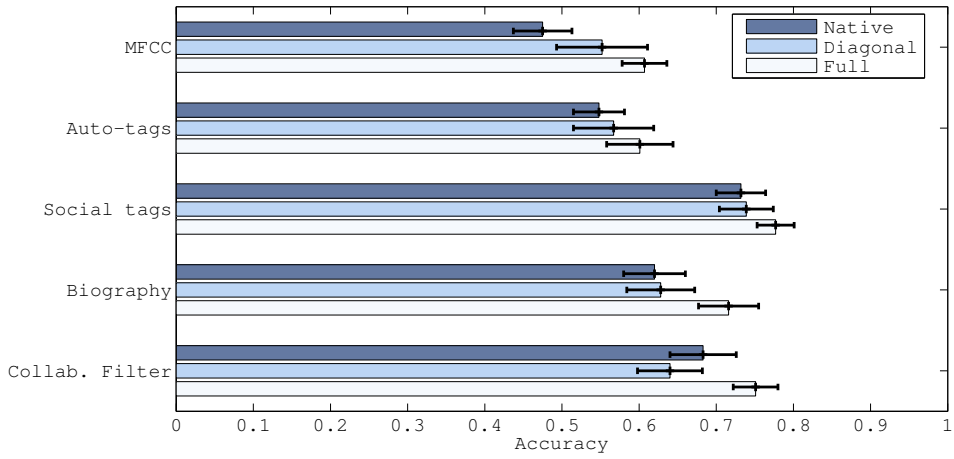


Figure 2.6: aset400 embedding results for each base kernel. Accuracy is computed in each kernel's native feature space, and in the space produced by algorithm 2.3 ($\mathcal{K}_1$ with a single kernel) with either the diagonal or full-matrix formulation. Error bars correspond to one standard deviation across training/test splits.

by algorithm 2.4 (optimizing over $\mathcal{K}_4$). The results are listed in fig. 2.7.

In all cases, MKPOE improves over the unweighted combination of base kernels. Moreover, many combinations outperform the single best kernel (ST, 0.777 ± 0.02 after optimization), and the algorithm is generally robust in the presence of poorly-performing kernels (MFCC and AT). Note that the poor performance of MFCC and AT kernels may be expected, as they derive from song-level rather than artist-level features, whereas ST provides high-level semantic descriptions which are generally more homogeneous across the songs of an artist, and Bio and CF are directly constructed at the artist level. For comparison purposes, we trained metrics on the sum kernel with $\mathcal{K}_1$ (algorithm 2.3), resulting in accuracies of 0.676 ± 0.05 (diagonal) and 0.765 ± 0.03 (full). The proposed approach (algorithm 2.4) applied to all kernels results in 0.754 ± 0.03 (diagonal), and 0.795 ± 0.02 (full).

Figure 2.8 illustrates the weights learned by algorithm 2.4 using all five kernels and diagonally-constrained W^p matrices. Note that the learned metrics are both sparse (many 0 weights) and non-uniform across different kernels. In particular, the (lowest-performing) MFCC kernel is eliminated by the algorithm, and the majority of the weight

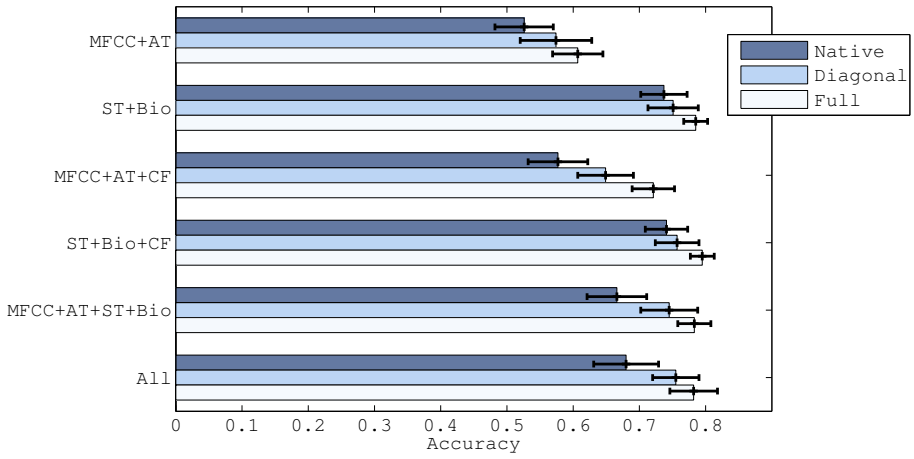


Figure 2.7: aset400 embedding results with multiple-kernel embedding: the learned metrics are optimized over $\mathcal{K}_4$ by algorithm 2.4. *Native* corresponds to distances calculated according to the unweighted sum of base kernels.

is assigned to the (highest-performing) social tag (ST) kernel.

A t-SNE (van der Maaten and Hinton, 2008) visualization of the space produced by MKPOE is illustrated in fig. 2.9. The embedding captures a great deal of high-level genre structure: for example, the *classic rock* and *metal* genres lie at the opposite end of the space from *pop* and *hip-hop*.

Graph processing results

To evaluate the effects of processing the constraint set for consistency and redundancy, we repeat the experiment of the previous section with different levels of processing applied to $\mathcal{A}$. Here, we focus on the Biography kernel, since it exhibits the largest gap in performance between the native and learned spaces.

As a baseline, we first consider the full set of similarity measurements as provided by human judgements, including all inconsistencies. To first deal with what appear to be the most egregious inconsistencies, we prune all directly inconsistent training measurements; that is, whenever (i, j, i, k) and (i, k, i, j) both appear, both are removed.[3]

[3] A more sophisticated approach could be used here, for example, majority voting, provided there is sufficient over-sampling of comparisons. The aset400 data lacks sufficient over-sampling for majority voting, so we default to this relatively simple approach.

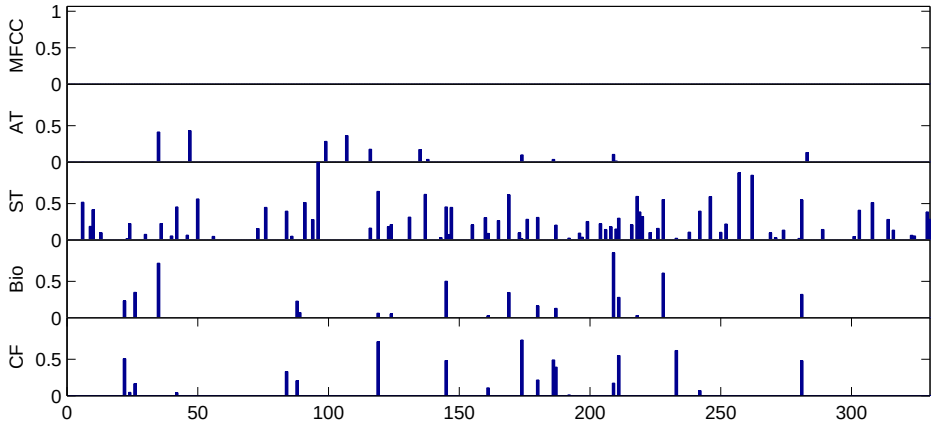


Figure 2.8: The weighting learned by algorithm 2.4 using all five kernels and diagonal W^p. Each bar plot contains the diagonal of the corresponding kernel's learned metric. The horizontal axis corresponds to the index in the training set, and the vertical axis corresponds to the learned weight in each kernel space.

Finally, we consider the fully processed case by finding a maximal consistent subset (partial order) of $\mathcal{A}$ and removing all redundancies. Table 2.2 lists the number of training constraints retained by each step of processing (averaged over the random splits).

Using each of these variants of the training set, we test the embedding algorithm with both diagonal and full-matrix formulations. The results are presented in table 2.2. Each level of graph processing results in a significant reduction in the number of training comparisons (and, therefore, computational overhead of algorithm 2.3), while not degrading the quality of the resulting embedding.

Finally, to test the sensitivity of the algorithm to randomness in the acyclic subgraph routine, we repeated the above experiment ten times, each with a different random maximal acyclic constraint set and the full matrix formulation of the algorithm. As depicted in fig. 2.10, the randomness in the constraint generation has little impact on the accuracy of the learned metric: the largest standard deviation is 0.007 (split #7).

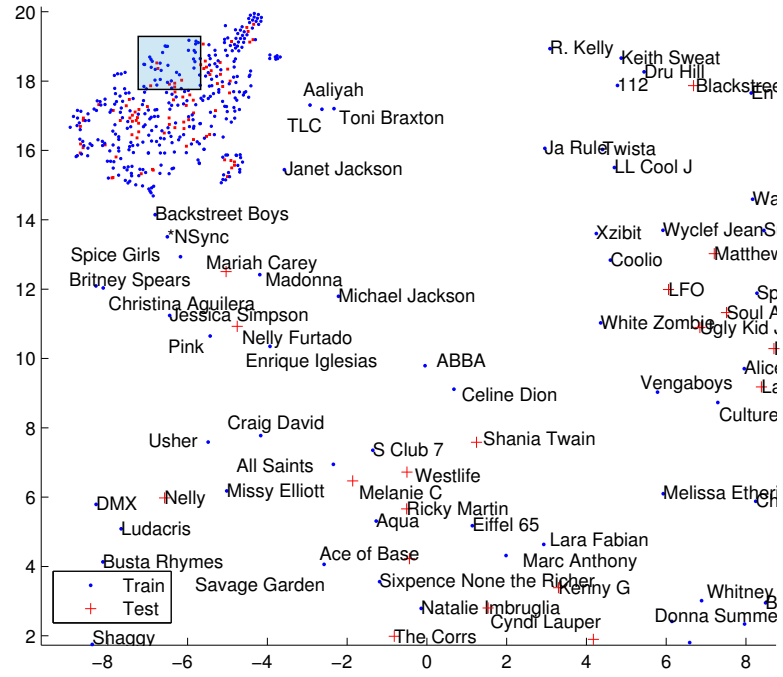


Figure 2.9: t-SNE visualizations of an embedding of aset400 produced by MKPOE. The embedding is constructed by optimizing over $\mathcal{K}_4$ with all five base kernels. The two clusters shown roughly correspond to (a) pop/hip-hop, and (b) classic rock/metal genres. Out-of-sample points are indicated by a red +.

Table 2.2: aset400 embedding results (Biography kernel) for three refinements of the constraint set. *Full* includes all constraints with no pruning. *Length-2* removes all length-2 cycles (*i.e.*, (i, j, k, ℓ) and (k, ℓ, i, j)). *Processed* finds a maximal consistent subset, and removes redundant constraints.

		Accuracy	
$\mathcal{A}$	$\lvert\mathcal{A}\rvert$ (Avg.)	Diagonal	Full
Full	8951.3	0.622±0.05	0.715±0.04
Length-2	6684.5	0.630±0.05	0.714±0.04
Processed	4814.5	0.628±0.05	0.716±0.04

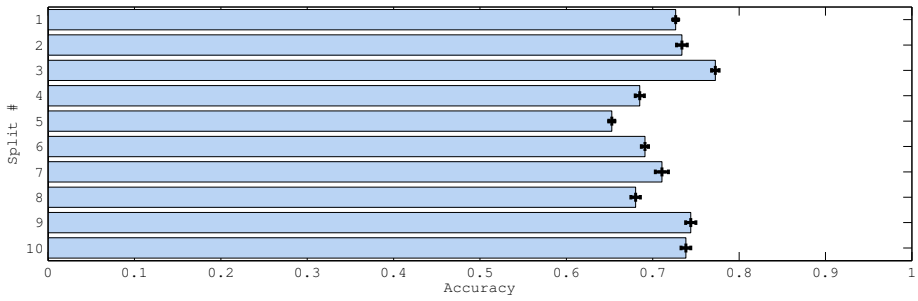


Figure 2.10: Accuracy of the learned embedding for each training/test split, averaged over ten trials with random maximal acyclic constraint subgraphs. Error bars correspond to one standard deviation.

2.6 Hardness of dimensionality reduction

The algorithms given in section 2.3 and section 2.4 attempt to produce low-dimensional solutions by regularizing W, which can be seen as a convex approximation to the rank of the embedding. In general, because rank constraints are not convex, convex optimization techniques cannot efficiently minimize dimensionality. This does not necessarily imply other techniques could not work. So, it is natural to ask if exact solutions of minimal dimensionality can be found efficiently, particularly in the multi-dimensional scaling scenario, that is, when $K = I$ (section 2.3.3).

As a special case, one may wonder if any instance $(\mathcal{X}, \mathcal{A})$ can be satisfied in $\mathbb{R}^1$. As fig. 2.11 demonstrates, not all instances can be realized in one dimension. Even

more, we show that it is NP-complete to decide if a given $\mathcal{A}$ can be satisfied in $\mathbb{R}^1$. Given an embedding, it can be verified in polynomial time whether $\mathcal{A}$ is satisfied or not by simply computing the distances between all pairs and checking each comparison in $\mathcal{A}$, so the decision problem is in NP. It remains to show that the $\mathbb{R}^1$ partial order embedding problem (hereafter referred to as *1-POE*) is NP-hard.[4] We reduce from the *Betweenness* problem (Opatrny, 1979), which is known to be NP-complete.

Definition 2.1 (Betweenness)**.** *Given a finite set S and a collection T of ordered triples (a, b, c) of distinct elements from S, decide if there exists an injection $f : S \to \mathbb{Z}$ such that for each $(a, b, c) \in T$, either $f(a) < f(b) < f(c)$ or $f(c) < f(b) < f(a)$.*

Theorem 2.1. *1-POE is NP-hard.*

Proof. Let (S, T) be an instance of Betweenness. Let $\mathcal{X} := S$, and for each $(a, b, c) \in T$, introduce constraints (a, b, a, c) and (b, c, a, c) to $\mathcal{A}$. Since Euclidean distance in $\mathbb{R}^1$ is simply line distance, these constraints exactly encode the constraint that $g(b)$ lies between $g(a)$ and $g(c)$.

If a solution g to the constructed 1-POE problem exists, it can be used to construct a solution f to Betweenness by sorting $g(\cdot)$ in ascending order. If a solution f exists to Betweenness, then $g = f$ is also a solution to 1-POE, since $\mathbb{Z} \subset \mathbb{R}^1$ and all constraints $\mathcal{A}$ are satisfied. Therefore, $(S, T) \in$ Betweenness if and only if $(\mathcal{X}, \mathcal{A}) \in$ 1-POE. Since Betweenness is NP-hard, 1-POE is NP-hard as well. □

Since 1-POE can be reduced to the general optimization problem of finding an embedding of minimal dimensionality, we conclude that dimensionality reduction subject to partial order constraints is also NP-hard.

2.7 Conclusion

We have demonstrated a novel method for optimally integrating heterogeneous data to conform to measurements of perceptual similarity. By interpreting a collection of relative similarity comparisons as a directed graph over pairs, we are able to apply

[4] We use the real numbers $\mathbb{R}^1$ only for notational convenience, and should be understood as referring to computable numbers here.

(a) (b)

Figure 2.11: (a) The vertices of a square in $\mathbb{R}^2$. (b) The partial order over distances induced by the square: each side is less than each diagonal. This constraint set cannot be satisfied in $\mathbb{R}^1$.

graph-theoretic techniques to isolate and prune inconsistencies in the training set and reduce computational overhead by eliminating redundant constraints in the optimization procedure.

Our multiple-kernel formulation offers a principled way to integrate multiple feature modalities into a unified similarity space. Our formulation carries the intuitive geometric interpretation of concatenated projections, and results in a semidefinite program. By incorporating diagonal constraints as well, we are able to reduce the computational complexity of the algorithm, and learn a model which is both flexible — only using kernels in the portions of the space where they are informative — and interpretable — each diagonal weight corresponds to the contribution to the optimized space due to a single point within a single feature space.

Acknowledgments

The contents of this chapter originally appeared in the following publication: B. McFee and G.R.G. Lanckriet. Learning multi-modal similarity. *Journal of Machine Learning Research*, 12:491–523, February 2011.

Algorithm 2.5 Naïve total order construction

Input: objects $\mathcal{X}$, partial order $\mathcal{A}$

Output: symmetric dissimilarity matrix $D \in \mathbb{R}^{n \times n}$

for each i in $1 \ldots n$ **do**
  $D_{ii} \leftarrow 0$ // Distance to self is 0
end for
for each (k, ℓ) in topological order **do**
  if in-degree$(k, \ell) = 0$ **then**
    $D_{k\ell} \leftarrow D_{\ell k} \leftarrow 1$
  else
    $D_{k\ell} \leftarrow D_{\ell k} \leftarrow \max_{i,j:(i,j,k,\ell) \in \mathcal{A}} D_{ij} + 1$
  end if
end for

2.A Embedding partial orders

In this appendix, we prove that any set $\mathcal{X}$ with a partial order over distances $\mathcal{A}$ can be embedded into $\mathbb{R}^n$ while satisfying all distance comparisons.

In the special case where $\mathcal{A}$ is a total ordering over all pairs (*i.e.*, a chain graph), the problem reduces to non-metric multidimensional scaling (Kruskal, 1964), and a constraint-satisfying embedding can always be found by the constant-shift embedding algorithm of Roth et al. (2003). In general, $\mathcal{A}$ is not a total order, but a $\mathcal{A}$-respecting embedding can always be produced by reducing the partial order to a (weak) total order by topologically sorting the graph (see algorithm 2.5).

Let D be the dissimilarity matrix produced by algorithm 2.5 on $(\mathcal{X}, \mathcal{A})$. An embedding can be found by first applying classical multidimensional scaling (MDS) (Cox and Cox, 1994) to D:

$$A = -\frac{1}{2} HDH,$$

where $H \coloneqq I - \frac{1}{n}\mathbf{1}\mathbf{1}^\mathsf{T}$ is the $n \times n$ centering matrix, and $\mathbf{1}$ is a vector of 1s. Shifting the spectrum of A yields

$$\widehat{A} \coloneqq A - \lambda_n(A) I \in \mathbb{S}^n_+,$$

where $\lambda_n(A)$ is the minimum eigenvalue of A. The embedding g can be found by

decomposing $\widehat{A} = V\widehat{\Lambda}V^{\mathsf{T}}$, so that $g(x_i)$ is the i^{th} column of $\widehat{\Lambda}^{1/2}V^{\mathsf{T}}$; this is the solution constructed by the constant-shift embedding non-metric MDS algorithm of Roth et al. (2003).

Applying this transformation to A affects distances by

$$\begin{aligned}\|g(x_i) - g(x_j)\|^2 = \widehat{A}_{ii} + \widehat{A}_{jj} - 2\widehat{A}_{ij} &= (A_{ii} - \lambda_n) + (A_{jj} - \lambda_n) - 2A_{ij} \\ &= A_{ii} + A_{jj} - 2A_{ij} - 2\lambda_n.\end{aligned}$$

Since adding a constant ($-2\lambda_n$) preserves the ordering of distances, the total order (and hence $\mathcal{A}$) is preserved by this transformation. Thus, for any instance $(\mathcal{X}, \mathcal{A})$, an embedding can be found in $\mathbb{R}^{n-1}$.

2.B Solver

Our implementation of algorithm 2.4 is based on projected sub-gradient descent. To simplify exposition, we show the derivation of the single-kernel SDP version of algorithm 2.3 with unit margins. (It is straightforward to extend the derivation to the multiple-kernel and LP settings.)

We first observe that a kernel matrix column K_i can be expressed as $K^{\mathsf{T}}\mathsf{e}_i$ where e_i is the i^{th} standard basis vector. We can then denote the distance calculations in terms of Frobenius inner products:

$$\begin{aligned}\mathbf{d}(x_i, x_j) &= (K_i - K_j)^{\mathsf{T}} W (K_i - K_j) \\ &= (\mathsf{e}_i - \mathsf{e}_j)^{\mathsf{T}} KWK(\mathsf{e}_i - \mathsf{e}_j) \\ &= \operatorname{tr}(KWK(\mathsf{e}_i - \mathsf{e}_j)(\mathsf{e}_i - \mathsf{e}_j)^{\mathsf{T}}) = \operatorname{tr}(WKE_{ij}K) \\ &= \langle W, KE_{ij}K\rangle_{\mathsf{F}}\,,\end{aligned}$$

where $E_{ij} = (\mathsf{e}_i - \mathsf{e}_j)(\mathsf{e}_i - \mathsf{e}_j)^{\mathsf{T}}$.

A margin constraint (i, j, k, ℓ) can now be expressed as:

$$\begin{aligned}& \mathbf{d}(x_i, x_j) + 1 \leq \mathbf{d}(x_k, x_\ell) + \xi_{ijk\ell} \\ \Rightarrow \quad & \langle W, KE_{ij}K\rangle_{\mathsf{F}} + 1 \leq \langle W, KE_{k\ell}K\rangle_{\mathsf{F}} + \xi_{ijk\ell} \\ \Rightarrow \quad & \xi_{ijk\ell} \geq 1 + \langle W, K(E_{ij} - E_{k\ell})K\rangle_{\mathsf{F}}\,.\end{aligned}$$

The slack variables $\xi_{ijk\ell}$ can be eliminated from the program by rewriting the objective in terms of the constraints:

$$\min_{W \in \mathbb{S}_+^n} \quad \underbrace{\langle W, K \rangle_{\mathsf{F}} + \frac{C}{|\mathcal{A}|} \sum_{\mathcal{A}} h\left(1 + \langle W, K(E_{ij} - E_{k\ell})K \rangle_{\mathsf{F}}\right)}_{f(W)},$$

where

$$h(x) \coloneqq x \cdot [\![x > 0]\!]$$

is the hinge loss.

A sub-gradient $\nabla f(W) \in \partial f(W)$ has two components: one due to regularization, and one due to the hinge loss. The gradient due to regularization is

$$\nabla_W \langle W, K \rangle_{\mathsf{F}} = K.$$

The loss term decomposes linearly, and for each $(i, j, k, \ell) \in \mathcal{A}$, a sub-gradient is

$$\nabla_W \; h\left(1 + \mathbf{d}(x_i, x_j) - \mathbf{d}(x_k, x_\ell)\right) = \begin{cases} 0 & \mathbf{d}(x_i, x_j) + 1 \leq \mathbf{d}(x_k, x_\ell) \\ K(E_{ij} - E_{k\ell})K & \text{otherwise.} \end{cases} \tag{2.11}$$

Rather than computing each sub-gradient direction independently, we observe that each violated constraint contributes a matrix of the form $K(E_{ij} - E_{k\ell})K$. By linearity, we can collect all $(E_{ij} - E_{k\ell})$ terms and then pre- and post-multiply by K to obtain a more efficient calculation of ∇f:

$$\nabla f(W) = K + \frac{C}{|\mathcal{A}|} K \left(\sum_{(i,j,k,\ell) \in \mathcal{A}'(W)} E_{ij} - E_{k\ell} \right) K \in \partial f,$$

where $\mathcal{A}'(W) \subseteq \mathcal{A}$ is the set of constraints violated by W.

After each step $W \mapsto W - \alpha \, \nabla f(W)$, the updated W is projected back onto the set of positive semidefinite matrices by computing its spectral decomposition and thresholding the eigenvalues by $\lambda_i \mapsto \max(0, \lambda_i)$.

To extend this derivation to the multiple-kernel case (algorithm 2.4), we can define

$$\mathbf{d}(x_i, x_j) \coloneqq \sum_{p=1}^{m} \mathbf{d}^p(x_i, x_j),$$

and exploit linearity to compute each partial sub-gradient $\nabla_{W^p} f(W^p)$ independently.

For the diagonally-constrained case, it suffices to substitute

$$K(E_{ij} - E_{k\ell})K \quad \mapsto \quad \mathrm{diag}(K(E_{ij} - E_{k\ell})K)$$

in eq. (2.11). After each step in the diagonal case, the PSD constraint on W can be enforced by the projection $W_{ii} \mapsto \max(0, W_{ii})$.

2.C Relationship to AUC

In this appendix, we formalize the connection between partial orders over distances and query-by-example ranking. Recall that algorithm 2.2 minimizes the loss $1/|\mathcal{A}| \sum_{\mathcal{A}} \xi_{ijk\ell}$, where each $\xi_{ijk\ell} \geq 0$ is a slack variable associated with a margin constraint

$$\mathbf{d}(i, j) + 1 \leq \mathbf{d}(k, \ell) + \xi_{ijk\ell}.$$

As noted by Schultz and Joachims (2004), the fraction of relative comparisons satisfied by an embedding g is closely related to the area under the receiver operating characteristic curve (AUC). To make this connection precise, consider the following information retrieval problem. For each point $x_i \in \mathcal{X}$, we are given a partition of $\mathcal{X} \setminus \{ i \}$:

$$\mathcal{X}_i^+ = \{ x_j \mid x_j \in \mathcal{X} \text{ relevant for } x_i \}, \text{ and}$$
$$\mathcal{X}_i^- = \{ x_k \mid x_k \in \mathcal{X} \text{ irrelevant for } x_i \}.$$

If we embed each $x_i \in \mathcal{X}$ into a Euclidean space, we can then rank the rest of the data $\mathcal{X} \setminus \{ x_i \}$ by increasing distance from x_i. Truncating this ranked list at the top τ elements (*i.e.*, closest τ points to x_i) will return a certain fraction of relevant points (true positives), and irrelevant points (false positives). Averaging over all values of τ defines the familiar AUC score, which can be compactly expressed as:

$$\mathrm{AUC}(x_i|g) = \frac{1}{|\mathcal{X}_i^+| \cdot |\mathcal{X}_i^-|} \sum_{(x_j, x_k) \in \mathcal{X}_i^+ \times \mathcal{X}_i^-} [\![\|g(x_i) - g(x_j)\| < \|g(x_i) - g(x_k)\|]\!].$$

Intuitively, AUC can be interpreted as an average over all pairs $(x_j, x_k) \in \mathcal{X}_i^+ \times \mathcal{X}_i^-$ of the number of times x_i was mapped closer to a relevant point x_j than an irrelevant

point x_k. This in turn can be conveniently expressed by a set of relative comparisons for each $x_i \in \mathcal{X}$:

$$\mathcal{A}_i \coloneqq \left\{ (i,j,i,k) \;\middle|\; (x_j, x_k) \in \mathcal{X}_i^+ \times \mathcal{X}_i^- \right\}, \qquad \mathcal{A} \coloneqq \bigcup_{x_i \in \mathcal{X}} \mathcal{A}_i.$$

An embedding which satisfies a complete set of constraints of this form will receive an AUC score of 1, since every relevant point must be closer to x_i than every irrelevant point.

Now, returning to the more general setting, we do not assume binary relevance scores or complete observations of relevance for all pairs of points. However, we can define the generalized AUC score (GAUC) as simply the average number of correctly ordered pairs (equivalently, satisfied constraints) given a set of relative comparisons:

$$\text{GAUC}(g) = \frac{1}{|\mathcal{A}|} \sum_{(i,j,k,\ell) \in \mathcal{A}} [\![\|g(x_i) - g(x_j)\| < \|g(x_k) - g(x_\ell)\|]\!].$$

Like AUC, GAUC is bounded between 0 and 1, and the two scores coincide exactly in the previously described ranking problem. A corresponding loss function can be defined by reversing the order of the inequality, that is,

$$L_{\text{GAUC}}(g) = \frac{1}{|\mathcal{A}|} \sum_{(i,j,k,\ell) \in \mathcal{A}} [\![\|g(x_i) - g(x_j)\| \geq \|g(x_k) - g(x_\ell)\|]\!].$$

Note that L_{GAUC} takes the form of a sum over indicators, and can be interpreted as the average 0/1-loss over $\mathcal{A}$. This function is clearly not convex in g, and is therefore difficult to optimize. Algorithms 2.2 to 2.4 instead optimize a convex upper bound on L_{GAUC} by replacing indicators by the hinge loss.

As in SVM, this is accomplished by introducing a unit margin and slack variable $\xi_{ijk\ell}$ for each $(i,j,k,\ell) \in \mathcal{A}$, and minimizing $1/|\mathcal{A}| \sum_{\mathcal{A}} \xi_{ijk\ell}$.

Chapter 3

Metric learning to rank

3.1 Introduction

In many machine learning tasks, good performance hinges upon the definition of similarity between objects. Although Euclidean distance on raw features provides a simple and mathematically convenient metric, there is often no reason to assume that it is optimal for the task at hand. Consequently, many researchers have developed algorithms to automatically learn distance metrics in supervised settings.

With few exceptions, these *metric learning* algorithms all follow the same guiding principle: a point's *good* neighbors should lie closer than its *bad* neighbors. The exact definitions of *good* and *bad* vary across problem settings, but typically they derive from some combination of feature proximity and label agreement. In keeping with this principle, metric learning algorithms are often evaluated by testing the accuracy of labels predicted by k-nearest neighbors on held out data.

At a high level, we consider a metric good if, when given a test point q, sorting the training set by increasing distance from q results in good neighbors at the front of the list, and bad neighbors at the end. Viewed in this light, we can cast nearest neighbor prediction as a ranking problem, and the predicted label error rate as a loss function over rankings. Thus, at its core, the metric learning problem is a special case of information retrieval in the *query-by-example* paradigm.

In recent years, many advances have been made in the development of learning algorithms for ranking (Joachims, 2005, Burges et al., 2005, Xu and Li, 2007, Volkovs

and Zemel, 2009). Unlike the classification problems typically addressed by metric learning, ranking problems generally lack a single evaluation criterion. Rather, several evaluation measures have been proposed, each capturing a different notion of "correctness." Because rankings are inherently combinatorial objects, these evaluation measures are often non-differentiable with respect to model parameters, and therefore difficult to optimize by learning algorithms. Despite the combinatorial difficulties of ranking problems, there are now several algorithmic techniques for optimizing various ranking evaluation measures (Joachims, 2005, Chakrabarti et al., 2008, Volkovs and Zemel, 2009).

In this chapter, we seek to bridge the gap between metric learning and ranking. By adapting techniques from information retrieval, we derive a general metric learning algorithm which optimizes for the true quantity of interest: the permutation of data induced by distances in the learned metric.

Conversely, our parameterization of the ranking function by a distance metric is quite natural for many information retrieval applications, including multi-media recommendation.

The present approach, based on structural SVM (Tsochantaridis et al., 2005), readily supports various ranking evaluation measures under a unified algorithmic framework. The interpretation of metric learning as an information retrieval problem allows us to apply loss at the level of rankings, rather than pairwise distances, and enables the use of more general notions of supervision than those used in previous metric learning algorithms, including asymmetric and non-transitive definitions of relevance.

3.1.1 Related work

There has been a great deal of research devoted to the design of algorithms for learning an optimal metric in supervised settings. Typically, these algorithms follow the general scheme of learning a linear transformation of the data such that distances to a pre-determined set of "good neighbors" is minimized, while "bad neighbor" distances are maximized.

Xing et al. (2003) define the good neighbors as all similarly-labeled points, and solve for the metric by semidefinite programming. Distances for similar pairs of points are upper-bounded by a constant, and dissimilar-pair distances are maximized. This

algorithm attempts to map each class into a ball of fixed radius, but does not enforce separation between classes. Weinberger et al. (2006) follow a similar approach, but enforce constraints *locally — i.e.*, label agreement is only enforced among the k nearest neighbors of each point, rather than the entire class — rather than globally. The resulting *large margin nearest neighbor* (LMNN) algorithm has enjoyed widespread success in a variety of real-world applications.

Neighborhood components analysis (NCA) (Goldberger et al., 2005) relaxes the problem by maximizing the expected number of correctly retrieved points under a stochastic neighbor selection rule. While this relaxation makes intuitive sense, the resulting optimization is non-convex, and it cannot identify and optimize the top-k nearest neighbors in the learned space. Globerson and Roweis (2006) optimize a similar stochastic neighbor selection rule while attempting to collapse each class to a single point. This idea enforces more regularity on the output space than NCA and leads to a convex optimization problem, but the assumption that entire classes can be collapsed to distinct points rarely holds in practice.

The core of our method is based on the structural SVM framework (Tsochantaridis et al., 2005). We provide a brief overview in section 3.2, and discuss ranking-specific extensions in section 3.4.

3.1.2 Preliminaries

Let $\mathcal{X} \subset \mathbb{R}^d$ denote a training set (corpus) of n documents. $\mathcal{Y}$ will denote the set of permutations (rankings) of $\mathcal{X}$. For a query q, let $\mathcal{X}_q^+$ and $\mathcal{X}_q^-$ denote the subsets of relevant and irrelevant points in the training set. For a ranking $y \in \mathcal{Y}$ and two points $x_i, x_j \in \mathcal{X}$, we will use $i \prec_y j$ ($i \succ_y j$) to indicate that i is placed before (after) j in y. Finally, let $y(k)$ denote the index of the item at position k in a ranking y.

3.2 Structural SVM review

Structural SVM can be viewed as a generalization of multi-class SVM (Crammer and Singer, 2002), where the set of possible prediction outcomes is generalized from labels to structures, *e.g.*, a parse tree, permutation, sequence alignment, *etc.* (Tsochan-

taridis et al., 2005). The multi-class SVM formulation of Crammer and Singer (2002) forces margins for each training point $q \in \mathcal{X}$ between the true label y_q and all other labels $y \in \mathcal{Y}$:

$$\forall y \neq y_q : \langle w_{y_q}, q \rangle \geq \langle w_y, q \rangle + 1 - \xi,$$

where $\xi \geq 0$ is a slack variable to allow margin violations on the training set. Similarly, structural SVM applies margins between the true *structure* y_q and all other possible structures y:

$$\forall y \in \mathcal{Y} : \langle w, \psi(q, y_q) \rangle \geq \langle w, \psi(q, y) \rangle + \Delta(y_q, y) - \xi. \tag{3.1}$$

Here, $\psi(q, y)$ is a vector-valued joint feature map which characterizes the relationship between an input q and an output structure y. (This notation subsumes the class-specific discriminant vectors of multi-class SVM.) Unlike class labels, two distinct structures (y_q, y) may exhibit similar accuracy, and the margin constraint should reflect this. To support more flexible notions of structural correctness, the margin is set to $\Delta(y_q, y)$: a non-negative loss function defined between structures, which is typically bounded in $[0, 1]$.

For a test query q' in multi-class SVM, the predicted label y is that which maximizes $\langle w_y, q' \rangle$, *i.e.*, the label with the largest margin over other labels. Analogously, structural predictions are made by finding the structure y which maximizes $\langle w, \psi(q', y) \rangle$. The prediction algorithm must be able to efficiently use the learned vector w when computing the output structure y. As we will see in section 3.2.2 and section 3.3, this is easily accomplished in general ranking, and specifically in metric learning.

3.2.1 Optimization

Note that the set $\mathcal{Y}$ of possible output structures is generally quite large (*e.g.*, all possible permutations of the training set), so enforcing all margin constraints in eq. (3.1) may not be feasible in practice. However, cutting planes can be applied to efficiently find a small set $\mathcal{A}$ of active constraints which are sufficient to optimize w within some prescribed tolerance (Tsochantaridis et al., 2005).

The core component of the cutting plane approach is the *separation oracle*, which given a fixed w and input point q, outputs the structure $\overline{y}$ corresponding to the

margin constraint for q which is most violated by w:

$$\overline{y} \leftarrow \operatorname*{argmax}_{y \in \mathcal{Y}} \langle w, \psi(q, y) \rangle + \Delta(y_q, y). \tag{3.2}$$

Intuitively, this computes the structure y with simultaneously large loss $\Delta(y_q, y)$ and margin score $\langle w, \psi(q, y) \rangle$: in short, the weak points of the current model w. Adding margin constraints for these structures y efficiently directs the optimization toward the global optimum by focusing on the constraints which are violated the most by the current model.

In summary, in order to apply structural SVM to a learning problem, three things are required: a definition of the feature map ψ, the loss function Δ, and an efficient algorithm for the separation oracle. These procedures are all of course highly interdependent and domain-specific. In the next section, we will describe the prevalent approach to solving ranking problems in this setting.

3.2.2 Ranking with structural SVM

In the case of ranking, the most commonly used feature map is the *partial order* feature (Joachims, 2005):

$$\psi_{\mathsf{po}}(q, y) := \sum_{\substack{x_i \in \mathcal{X}_q^+ \\ x_j \in \mathcal{X}_q^-}} y_{ij} \left(\frac{\phi(q, x_i) - \phi(q, x_j)}{|\mathcal{X}_q^+| \cdot |X_q^-|} \right), \tag{3.3}$$

where

$$y_{ij} := \begin{cases} +1 & i \prec_y j \\ -1 & i \succ_y j \end{cases},$$

and $\phi(q, x_i)$ is a feature map which characterizes the relation between a query q and point x_i. Intuitively, for each relevant-irrelevant pair (x_i, x_j), the difference vector $\phi(q, x_i) - \phi(q, x_j)$ is added if $i \prec_y j$ and subtracted otherwise. Essentially, ψ_{po} emphasizes directions in feature space which are in some sense correlated with correct rankings. Since ϕ only depends on the query and a single point, rather than the entire list, it is well-suited for incorporating domain-specific knowledge and features.

Separation oracles have been devised for ψ_{po} in conjunction with a wide variety of ranking evaluation measures (Joachims, 2005, Yue et al., 2007, Chakrabarti et al., 2008), and we give a brief overview in section 3.4.

One attractive property of ψ_{po} is that for a fixed w, the ranking y which maximizes $\langle w, \psi_{\mathsf{po}}(q', y)\rangle$ is simply $x_i \in \mathcal{X}$ sorted by descending $\langle w, \phi(q', x_i)\rangle$. As we will show in the next section, this simple prediction rule can be easily adapted to distance-based ranking.

3.3 Metric learning to rank

If the query q lies in the same space as the corpus $\mathcal{X}$, a natural ordering is produced by increasing (squared) distance from q: $\|q - x_i\|^2$. Since our goal is to learn an optimal metric W, distances are computed in the learned space and sorted accordingly: $\|q - x_i\|_W^2$. This computation is characterized in terms of Frobenius inner products as follows:

$$\begin{aligned}\|q - x_i\|_W^2 &= (q - x_i)^{\mathsf{T}} W (q - x_i) \\ &= \operatorname{tr}\left(W (q - x_i)(q - x_i)^{\mathsf{T}}\right) \\ &= \left\langle W, (q - x_i)(q - x_i)^{\mathsf{T}} \right\rangle_{\mathsf{F}},\end{aligned}$$

where the second equality follows by the cyclic property of the trace.

This observation suggests a natural choice of ϕ:

$$\phi_{\mathsf{M}}(q, x_i) := -(q - x_i)(q - x_i)^{\mathsf{T}}. \tag{3.4}$$

The change of sign preserves the ordering used in standard structural SVM. Sorting the corpus by ascending $\|q - x_i\|_W$ is therefore equivalent to sorting by descending $\langle W, \phi_{\mathsf{M}}(q, x_i)\rangle_{\mathsf{F}}$. Similarly, by using ϕ_{M} with ψ_{po}, the ordering y which maximizes the generalized inner product $\langle W, \psi_{\mathsf{po}}(q, y)\rangle_{\mathsf{F}}$ is precisely $\mathcal{X}$ in ascending order of distance from q under the metric defined by W.

Thus, by generalizing the vector products in eqs. (3.1) and (3.2) to Frobenius inner products, we can derive an algorithm to learn a metric optimized for list-wise ranking loss measures.

3.3.1 Algorithm

Ideally, we would like to solve for the optimal metric $\overline{W}$ which maximizes the margins over all possible rankings for each query. However, since $|\mathcal{Y}|$ is super-exponential in the size of the training set, implementing an exact optimization procedure is not possible with current techniques. Instead, we approximate the full optimization by using a cutting-plane algorithm.

Specifically, our algorithm for learning W is adapted from the 1-Slack margin-rescaling cutting-plane algorithm of Joachims et al. (2009). At a high-level, the algorithm alternates between optimizing the model parameters (in our case, W), and updating the constraint set with a new batch of rankings $(\overline{y}_1, \overline{y}_2, \ldots, \overline{y}_n)$ (one ranking for each point). The algorithm terminates once the empirical loss on the new constraint batch is within a prescribed tolerance $\epsilon > 0$ of the loss on the previous set of constraints.

The key difference between the 1-Slack approach and other similar cutting-plane techniques is that, rather than maintaining a slack variable ξ_q for each $q \in \mathcal{X}$, there is a single slack variable ξ which is shared across all constraint batches, which are in turn aggregated by averaging over each point in the training set.

We introduce two modifications to adapt the original algorithm to metric learning. First, W must be constrained to be positive semi-definite in order to define a valid metric. Second, we replace the standard quadratic regularization $\frac{1}{2}\|w\|^2$ (or $\frac{1}{2}\|W\|_{\mathsf{F}}^2$) with $\langle W, I\rangle_{\mathsf{F}} = \operatorname{tr}(W)$. Intuitively, this trades an ℓ_2 penalty on the eigenvalues of W for an ℓ_1 penalty, thereby promoting sparsity and low-rank solutions.

The general optimization procedure is listed as algorithm 3.1. For compactness, we define

$$\delta\psi_{\mathsf{po}}(q, y_q, y) := \psi_{\mathsf{po}}(q, y_q) - \psi_{\mathsf{po}}(q, y).$$

3.3.2 Implementation

To solve the optimization problem in algorithm 3.1, we implemented a projected sub-gradient descent solver in MATLAB[1]. After each sub-gradient step, the updated W is projected back onto the feasible set $\mathbb{S}_+^d$ by spectral decomposition.

[1] Source code can be found at http://www-cse.ucsd.edu/~bmcfee/code/mlr.

Algorithm 3.1 Metric Learning to Rank (MLR).

Input: data $\mathcal{X} \subset \mathbb{R}^d$, target rankings $(y_1, \ldots, y_n)$, $C > 0$, $\epsilon > 0$

Output: $\overline{W} \in \mathbb{S}_+^d$, $\overline{\xi} \geq 0$

1: $\mathcal{A} \leftarrow \emptyset$ // Initialize with no constraints

2: **repeat**

3: Solve for the optimal $\overline{W}$ and $\overline{\xi}$:

$$\left(\overline{W}, \overline{\xi}\right) \leftarrow \quad \operatorname*{argmin}_{W,\xi} \quad \underbrace{\langle W, I \rangle_{\mathsf{F}} + C\xi}_{f(W)}$$

$$\text{s. t. } \forall (\overline{y}_1, \overline{y}_2, \cdots, \overline{y}_n) \in \mathcal{A} :$$

$$\frac{1}{n} \sum_{q \in \mathcal{X}} \langle W, \delta\psi_{\mathsf{po}}(q, y_q, \overline{y}_q) \rangle_{\mathsf{F}} \geq \frac{1}{n} \sum_{q \in \mathcal{X}} \Delta(y_q, \overline{y}_q) - \xi$$

$$W \in \mathbb{S}_+^d$$

$$\xi \geq 0$$

4: **for** each $q \in \mathcal{X}$ **do**

5: $\overline{y}_q \leftarrow \operatorname*{argmax}_{y \in \mathcal{Y}} \Delta(y_q, y) + \left\langle \overline{W}, \psi_{\mathsf{po}}(q, y) \right\rangle_{\mathsf{F}}$

6: **end for**

7: $\mathcal{A} \leftarrow \mathcal{A} \cup \{ (\overline{y}_1, \ldots, \overline{y}_n) \}$ // Update the active set

8: **until**

$$\frac{1}{n} \sum_{q \in \mathcal{X}} \Delta(y_q, \overline{y}_q) - \left\langle \overline{W}, \delta\psi_{\mathsf{po}}(q, y_q, \overline{y}_q) \right\rangle_{\mathsf{F}} \leq \overline{\xi} + \epsilon$$

Although there appears to be a great many feature vectors ($\delta\psi_{\mathsf{po}}$) in use in the algorithm, efficient bookkeeping allows us to reduce the overhead of gradient calculations. Note that ξ can be interpreted as the point-wise maximum of a set $\{ \xi_1, \xi_2, \ldots \}$, where ξ_i corresponds to the margin constraint for the i^{th} batch. Therefore, at any time when $\xi > 0$, a sub-gradient $\nabla f(W)$ of the objective $f(W, \xi)$ can be expressed in terms of any single batch $(\overline{y}_1, \ldots, \overline{y}_n)$ which achieves the current largest margin violation (assuming

one exists):

$$\nabla f(W) = I - \frac{C}{n} \sum_{q \in \mathcal{X}} \delta\psi_{\mathsf{po}}(q, y_q, \overline{y}_q) \in \partial f(W).$$

Note that ψ_{po} only appears in algorithm 3.1 in the form of averages over constraint batches. This indicates that it suffices to maintain only a single $d \times d$ matrix

$$\Psi := \frac{1}{n} \sum_{q \in \mathcal{X}} \delta\psi_{\mathsf{po}}(q, y_q, \overline{y}_q)$$

for each batch, rather than individual matrices for each point. By exploiting the quadratic form of eq. (3.4), each $\psi_{\mathsf{po}}(q, y)$ can be factored as

$$\psi_{\mathsf{po}}(q, y) = X S(q, y) X^{\mathsf{T}},$$

where the columns of X contain the data, and

$$S(q, y) := \sum_{\substack{i \in \mathcal{X}_q^+ \\ j \in \mathcal{X}_q^-}} y_{ij} \left(\frac{E_{qi} - E_{qj}}{|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|} \right), \tag{3.5}$$

$$E_{qx} := -(\mathsf{e}_q - \mathsf{e}_x)(\mathsf{e}_q - \mathsf{e}_x)^{\mathsf{T}} \in \mathbb{S}^n,$$

and e_i is the i^{th} standard basis vector in $\mathbb{R}^n$. By linearity, this factorization can also be carried through to $\delta\psi_{\mathsf{po}}(q, y_q, y)$ and Ψ.

The summation in eq. (3.5) can be computed more directly by counting the occurrences of E_{qx} with positive and negative sign, and collecting the terms. This can be done in linear time by a single pass through y.

By expressing Ψ in factored form, we can delay all matrix multiplications until the final Ψ computation. Because the $S(q, y)$ can be constructed directly without explicitly building the outer-product matrices E_{qi}, we reduce the number of matrix multiplications at each sub-gradient calculation from $\mathcal{O}(n)$ to 2.

3.4 Ranking measures

Here, we give a brief overview of popular information retrieval evaluation criteria, and how to incorporate them into the learning algorithm.

Recall that the separation oracle (eq. (3.2)) seeks a ranking y which maximizes the sum of the discriminant score $\langle W, \psi_{\mathsf{po}}(q, y)\rangle_{\mathsf{F}}$ and the ranking loss $\Delta(y_q, y)$. One property shared by all evaluation criteria under consideration is invariance to permutations within relevant (or irrelevant) sets. As has been previously observed, optimizing over y reduces to finding an optimal interleaving of the relevant and irrelevant sets, each of which has been pre-sorted by the point-wise discriminant score $\langle W, \phi_{\mathsf{M}}(q, x_i)\rangle_{\mathsf{F}}$ (Yue et al., 2007).

Since all measures discussed here take values in $[0, 1]$ (1 being the score for an ideal ranking y_q), we consider loss functions of the form

$$\Delta(y_q, y) := \text{Score}(q, y_q) - \text{Score}(q, y) = 1 - \text{Score}(q, y).$$

3.4.1 AUC

The area under the ROC curve (AUC) is a commonly used measure which characterizes the trade-off between true positives and false positives as a threshold parameter is varied. In our case, the parameter corresponds to the number of items returned (or, predicted as relevant). AUC can equivalently be calculated by counting the portion of incorrectly ordered pairs (*i.e.*, $j \prec_y i$, i relevant and j irrelevant):

$$\text{AUC}(q, y) := \frac{1}{|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|} \sum_{\substack{i \in \mathcal{X}_q^+ \\ j \in \mathcal{X}_q^-}} \frac{1 + y_{ij}}{2}$$

This formulation leads to a simple and efficient separation oracle, described by Joachims (2005).

Note that AUC is position-independent: an incorrect pair-wise ordering at the bottom of the list impacts the score just as much as an error at the top of the list. In effect, AUC is a global measure of list-wise cohesion.

3.4.2 Precision-at-k

Precision-at-k (P@k) is the fraction of relevant results out of the first k returned. P@k is therefore a highly localized evaluation criterion, and captures the quality of rankings for applications where only the first few results matter, *e.g.*, web search.

The separation oracle for P@k exploits two facts: there are only $k+1$ possible values for P@k:

$$\text{P@}k(q,y) \in \left\{ \frac{0}{k}, \frac{1}{k}, \frac{2}{k}, \ldots, \frac{k}{k} \right\},$$

and for any fixed value, the best y is completely determined by the ordering induced by discriminant scores. We can then evaluate all $k+1$ interleavings of the data to find the y which achieves the maximum. See Joachims (2005) for details.

Closely related to P@k is the k-nearest neighbor prediction score. In the binary classification setting, the two are related by

$$k\text{NN}(q,y;k) := \left[\!\left[\text{P@k}(q,y) > \frac{1}{2} \right]\!\right],$$

and the P@k separation oracle can be easily adapted to k-nearest neighbor. However, in the multi-class setting, the interleaving technique fails because the required fraction of relevant points for correct classification depends not only on the relevance or irrelevance of each point, but the labels themselves.

In informal experiments, we noticed no quantitative differences in performance between metrics trained for (binary) kNN and P@k, and we omit kNN from the experiments in section 3.5.

3.4.3 Average Precision

Average precision (AP) (Baeza-Yates and Ribeiro-Neto, 1999) is the precision-at-k score of a ranking y, averaged over all positions k of relevant documents:[2]

$$\text{AP}(q,y) = \frac{1}{|\mathcal{X}_q^+|} \sum_{k=1}^{|\mathcal{X}_q^+|+|\mathcal{X}_q^-|} \text{P@}k(y) \cdot \left[\!\left[y(k) \in \mathcal{X}_q^+ \right]\!\right]. \tag{3.6}$$

Yue et al. (2007) provides a greedy separation oracle for average precision that runs in $\mathcal{O}(|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|)$ time. In appendix A, we derive a somewhat simpler dynamic programming algorithm with equivalent asymptotic runtime, which is used in our implementation.

[2] *Mean average precision* (MAP) is AP averaged across queries.

3.4.4 Mean Reciprocal Rank

Mean reciprocal rank (MRR) is the inverse position of the first relevant document in y, and is therefore well-suited to applications in which only the first result matters.

Like P@k, there is a finite set of possible score values for MRR:

$$\text{MRR}(q, y) \in \left\{ \frac{1}{1}, \frac{1}{2}, \ldots, \frac{1}{1 + |\mathcal{X}_q^-|} \right\}$$

and for a fixed MRR score, the optimal y is completely determined. It is similarly straightforward to search over score values for the maximizer. See Chakrabarti et al. (2008) for a more complete treatment of optimizing MRR.

3.4.5 Normalized Discounted Cumulative Gain

Normalized Discounted Cumulative Gain (NDCG) (Järvelin and Kekäläinen, 2000) is similar to MRR, but rather than rewarding only the first relevant document, all of the top k documents are scored at a decaying gain factor $G(\cdot)$. In the present setting with binary relevance levels, the formulation we adopt is expressed as:

$$\text{NDCG}(q, y; k) := \frac{\sum_{i=1}^{k} G(i) \cdot [\![i \in \mathcal{X}_q^+]\!]}{\sum_{i=1}^{k} G(i)}$$

$$G(i) := \begin{cases} 1 & i = 1 \\ \frac{1}{\log_2(i)} & 2 \leq i \leq k \\ 0 & i > k \end{cases}.$$

Chakrabarti et al. (2008) propose a dynamic programming algorithm for the NDCG separation oracle, which we adapt here.

3.5 Experiments

To evaluate the MLR algorithm, we performed experiments on both small-scale and large-scale data sets, as described in the next two sections. In all experiments, we fixed the accuracy threshold at $\epsilon = 0.01$.

Table 3.1: UCI data sets: dimensionality (d), training and test set sizes, and the number of classes. IsoLet's training set was further split into training and validation sets of size 4991 and 1247.

	d	# Train	# Test	# Classes
Balance	4	500	125	3
Ionosphere	34	281	70	2
WDBC	30	456	113	2
Wine	13	143	35	3
IsoLet	170	6238	1559	26

3.5.1 Classification on UCI data

We first tested the accuracy and dimensionality reduction performance of our algorithm on five data sets from the UCI repository (Asuncion and Newman, 2007): Balance, Ionosphere, WDBC, Wine, and IsoLet. For the first four sets, we generated 50 random 80/20 training and test splits. Each dimension of the data was z-scored by the statistics of the training splits.

For IsoLet, we replicate the experiment of Weinberger et al. (2006) by generating 10 random 80/20 splits of the training set for testing and validation, and then testing on the provided test set. We project by PCA (as computed on the training set) to 170 dimensions, enough to capture 95% of the variance.

Table 3.1 contains a summary of the data sets used here.

We trained metrics on each data set with the five variants of MLR: MLR-AUC, MLR-P@k, MLR-MAP, MLR-MRR, and MLR-NDCG. For comparison purposes, we also trained metrics with Large Margin Nearest Neighbor (LMNN) (Weinberger et al., 2006), Neighborhood Components Analysis (NCA) (Goldberger et al., 2005), and Metric Learning by Collapsing Classes (MLCC) (Globerson and Roweis, 2006).

To evaluate the performance of each algorithm, we tested k-nearest neighbor classification accuracy in the learned metrics. Classification results are presented in

Table 3.2: k-nearest neighbor classification error (%) on learned metrics. Reported error is corresponds to the best choice of C and k.

Algorithm	Balance	Ionosphere	WDBC	Wine	IsoLet
MLR-AUC	7.9	12.3	2.7	1.4	4.5
MLR-P@k	8.2	12.3	2.9	1.5	4.5
MLR-MAP	6.9	12.3	2.6	1.0	5.5
MLR-MRR	8.2	12.1	2.6	1.5	4.5
MLR-NDCG	8.2	11.9	2.9	1.6	4.4
LMNN	8.8	11.7	2.4	1.7	4.7
NCA	4.6	11.7	2.6	2.7	10.8
MLCC	5.5	12.6	2.1	1.1	4.4
Euclidean	10.3	15.3	3.1	3.1	8.1

table 3.2.[3] With the exception of NCA and MLCC on the Balance set, all results on Balance, Ionosphere, WDBC and Wine are within the margin of error. In general, MLR achieves accuracy on par with the best algorithms under comparison, without relying on the input features for selecting target neighbors.

Figure 3.1 illustrates the dimensionality reduction properties of the MLR algorithms. In all cases, MLR achieves significant reductions in dimensionality from the input space, comparable to the best competing algorithms.

3.5.2 eHarmony data

To evaluate MLR on a large data set in an information retrieval context, we trained metrics on matching data provided by eHarmony:[4] an online dating service which matches users by personality traits.

For our experiments, we focused on the following simplification of the data and

[3] LMNN accuracy on IsoLet was reported by Weinberger et al. (2006). Dimensionality results were not reported.

[4] http://www.eharmony.com

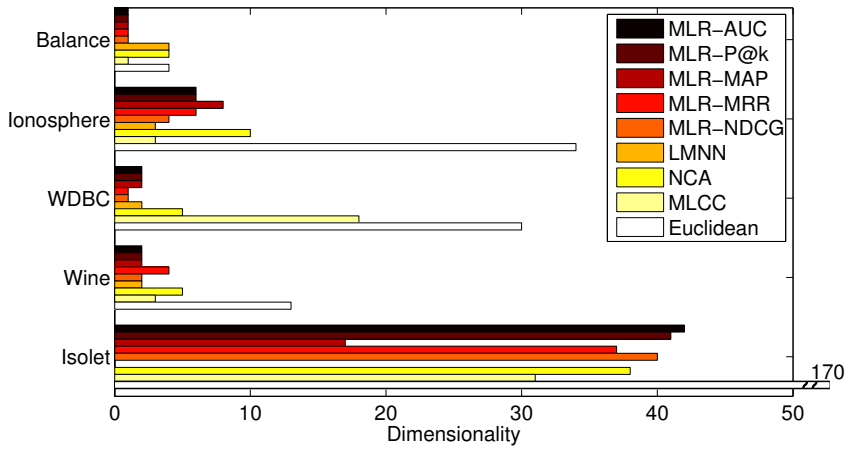


Figure 3.1: Dimensionality reduction for the UCI data sets. Reported dimensionality is the median number of dimensions necessary to capture 95% of the spectral mass of the best-performing W. "Euclidean" corresponds to the native dimensionality of the data.

problem: each matching is presented as a pair of users, with a positive label when the match was successful (*i.e.*, users expressed mutual interest), and negative otherwise. Each user is represented by a vector in $\mathbb{R}^{56}$ which describes the user's personality, interests, *etc.*. We consider two users mutually relevant if they are presented as a successful match, and irrelevant if the match is unsuccessful. Irrelevance is not assumed for unmatched pairs.

Matchings were collected over two consecutive time intervals of equal length, and split into training (interval 1) and testing (interval 2). The training split contains approximately 295000 unique users, not all of which define useful queries: some appear only in positive matchings, while others appear only in negative matchings. Since these users provide no discriminative data, we omit them from the set of *query users*. Note that such users are still informative, and are included in the training set as results to be ranked.

We further reduce the number of training queries to include only users with at least 2 successful and 5 unsuccessful matchings, leaving approximately 22000 training queries. A summary of the data is presented in table 3.3.

We trained metrics with MLR-AUC, MLR-MAP and MLR-MRR. Due to the small number of minimum positive results for each query, we omit MLR-P@k and

Table 3.3: The eHarmony matching data.

	Matchings	Unique users	Queries
Training	506688	294832	22391
Test	439161	247420	36037

Table 3.4: Testing accuracy and training time for MLR and SVM-MAP on eHarmony matching data. Time is reported in CPU-seconds, and $|\mathcal{A}|$ is the number of cutting-plane batches before convergence.

Algorithm	AUC	MAP	MRR	Time	$\lvert\mathcal{A}\rvert$
MLR-AUC	0.612	0.445	0.466	232	7
MLR-MAP	0.624	0.453	0.474	2053	23
MLR-MRR	0.616	0.448	0.469	809	17
SVM-MAP	0.614	0.447	0.467	4968	36
Euclidean	0.522	0.394	0.414	—	—

MLR-NDCG from this experiment. Note that because we are in an information retrieval setting, and not classification, the other metric learning algorithms compared in the previous section do not apply. For comparison, we train models with SVM-MAP (Yue et al., 2007), and feature map $\phi(q, x_i) := (q - x_i)$. When training SVM-MAP, we swept over $C \in \{ 10^{-2}, 10^{-1}, \ldots, 10^{5} \}$.

Table 3.4 shows the accuracy and timing results for MLR and SVM-MAP. The MLR-MAP and MLR-MRR models show slight, but statistically significant improvement over the SVM-MAP model. Note that the MLR algorithms train in significantly less time than SVM-MAP, and require fewer calls to the separation oracle.

3.6 Conclusion

We have presented a metric learning algorithm which optimizes for ranking-based loss functions. By casting the problem as an information retrieval task, we focus attention on what we believe to be the key quantity of interest: the permutation of data induced by distances.

Acknowledgments

The authors thank Steve Checkoway for helpful suggestions. The contents of this chapter originally appeared in the following publication: B. McFee and G.R.G. Lanckriet. Metric learning to rank. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, ICML, pages 775–782, Haifa, Israel, June 2010.

Chapter 4

Faster structural metric learning

4.1 Introduction

Structural metric learning algorithms produce a transformation of a feature space which is optimized to produce structured outputs, such as nearest-neighbor rankings or graphs induced by distance under the learned transformation (chapter 3) (Shaw et al., 2011). These algorithms have a wide variety of applications, including nearest-neighbor classification, data visualization, and query-by-example information retrieval. However, the power of these methods comes at a cost: finding the optimal transformation often amounts to solving a large semi-definite programs which can be prohibitively expensive in large data sets.

In this chapter, we develop an efficient training algorithm for structural metric learning. The proposed method combines the 1-slack structural SVM algorithm (Joachims et al., 2009) with the alternating-direction method of multipliers (ADMM) technique (Boyd et al., 2011), and reduces an expensive semi-definite programming problem to a sequence of small quadratic programs. The resulting algorithm provides substantial efficiency benefits over previous methods.

4.2 Structural metric learning

In a typical metric learning problem, the goal is to learn a linear transformation of some data $\mathcal{X} \subset \mathbb{R}^d$ in order to improve nearest-neighbor classification (Weinberger et al., 2006, Davis et al., 2007). Structural metric learning generalizes this setting to optimize the prediction of *structures* $\mathcal{Y}$ under the learned transformation, rather than the pair-wise distance constraints typical of classification-based metric learning. Formulating a metric learning algorithm (structural or not) directly in terms of the transformation $L \in \mathbb{R}^{k \times d}$ usually results in a difficult, non-convex optimization problem, which is then re-formulated as a convex optimization problem over a positive semi-definite matrix $W := L^\mathsf{T} L \in \mathbb{S}^d_+$.

For the purposes of this chapter, we will focus on the metric learning to rank (MLR) algorithm (chapter 3), although the proposed approach would apply equally well to the structure-preserving metric learning (SPML) algorithm (Shaw et al., 2011). MLR is a metric learning variation of the structural SVM (Tsochantaridis et al., 2005) which optimizes $W \in \mathbb{S}^d_+$ to minimize a ranking loss function $\Delta : \mathcal{Y} \times \mathcal{Y} \to \mathbb{R}_+$ (*e.g.*, decrease in mean average precision) over permutations of data. The optimization problem contains a large (super-exponential) number of constraints, but it can be efficiently approximated by cutting-plane techniques. Specifically, MLR uses the 1-slack method (Joachims et al., 2009) to approximate the following convex optimization problem:

$$\begin{aligned} \min_{W \in \mathbb{S}^d_+} \quad & \langle W, I \rangle_\mathsf{F} + \frac{C}{n} \sum_{q \in \mathcal{X}} \xi_q \\ \text{s.t. } \forall q \in \mathcal{X}, y \in \mathcal{Y}: \quad & \langle W, \psi(q, y_q) - \psi(q, y) \rangle_\mathsf{F} \geq \Delta(y_q, y) - \xi_q \end{aligned} \tag{4.1}$$

Here, $\mathcal{X} \subset \mathbb{R}^d$ is the training set of n points; $\mathcal{Y}$ is the set of all permutations over $\mathcal{X}$; $C > 0$ is a slack trade-off parameter; $\psi : \mathcal{X} \times \mathcal{Y} \to \mathbb{S}^d$ is a feature encoding of an input-output pair (q, y); and $\Delta(y_q, y) \in [0, 1]$ is the margin, *i.e.*, loss incurred for predicting a ranking y rather than the true ranking y_q. The feature map ψ is designed so that $\langle W, \psi(q, y) \rangle_\mathsf{F}$ is large when the ranking of $\mathcal{X}$ induced by distance from q agrees with y, and small otherwise. For a query q with relevant set $\mathcal{X}_q^+ \subseteq \mathcal{X}$ and irrelevant set

$\mathcal{X}_q^- \subseteq \mathcal{X}$, ψ is defined by

$$\psi(q, y) := \sum_{i \in \mathcal{X}_q^+} \sum_{j \in \mathcal{X}_q^-} y_{ij} \frac{\phi(q, x_i) - \phi(q, x_j)}{|\mathcal{X}_q^+| \cdot |\mathcal{X}_q^-|},$$

$$\phi(q, x) := -(q - x)(q - x)^\mathsf{T},$$

$$y_{ij} := \begin{cases} +1 & i \prec_y j \\ -1 & \text{otherwise} \end{cases}.$$

Algorithm 4.1 approximates eq. (4.1) by alternately solving a convex optimization problem (step 3) over a small set $\mathcal{A}$ of active constraints, and updating $\mathcal{A}$ with the constraints most violated by the resulting W (steps 5–10). The process repeats until the most-violated constraint (and hence, all other constraints) are satisfied to within some specified $\epsilon > 0$ of the loss on the active set $\mathcal{A}$.

In structural SVM, the constraint generation step forms the main computational bottleneck, as it typically entails solving a sequence of n dynamic programs, each with linear or quadratic complexity (depending on the choice of loss function Δ), resulting in $\mathcal{O}(n^2)$–$\mathcal{O}(n^3)$ computation for each iteration.[1] By contrast, the SVM optimization (analogous to step 3) is a small quadratic program, and can be solved quite efficiently.

However, in the metric learning problem, the situation can easily be reversed due to the increased complexity of semi-definite programming. In particular, the algorithm proposed in chapter 3 optimizes W in step 3 via projected sub-gradient descent. When the dimensionality d is larger than a few hundred, the complexity of repeatedly projecting back onto the feasible set after each step — $\mathcal{O}(d^3)$ for each spectral decomposition and negative eigenvalue thresholding operation — can easily overshadow the overhead of constraint generation.

We note that the vast majority of metric learning algorithms (structural or otherwise) suffer from this computational bottleneck, although various methods have been proposed to accelerate or circumvent maintaining feasibility, including: rank-1 updates (Shalev-Shwartz et al., 2004, Shen et al., 2009), factored optimization (Weinberger and Saul, 2008), and eigenvalue optimization (Ying and Li, 2012).

[1] Though it is not our focus in this work, we note that the burden of constraint generation can also be eased, for example, by exploiting parallelism or sub-sampling $\mathcal{X}$ in the loop 6–10 (Yu and Joachims, 2008).

Algorithm 4.1 Metric learning to rank (MLR)

Input: Data $\mathcal{X} \subset \mathbb{R}^d$, target rankings $\{y_q\}$, slack parameter $C > 0$, accuracy $\epsilon > 0$

Output: $\overline{W} \in \mathbb{S}_+^d$

1: $\mathcal{A} \leftarrow \{\ \}$

2: **repeat**

3: Solve for $\left(\overline{W}, \overline{\xi}\right)$:

$$\overline{W} \leftarrow \operatorname*{argmin}_{W \in \mathbb{S}_+^d} \underbrace{\langle W, I \rangle_{\mathsf{F}} + C \max_{(\Delta,\Psi)\in\mathcal{A}} h(\Delta - \langle W, \Psi \rangle_{\mathsf{F}})}_{f(W)} \tag{4.2}$$

$$\overline{\xi} \leftarrow \max_{(\Delta,\Psi)\in\mathcal{A}} h\left(\Delta - \left\langle \overline{W}, \Psi \right\rangle_{\mathsf{F}}\right)$$

$$h(x) := x \cdot [\![x > 0]\!]$$

4: $\widehat{\Delta} \leftarrow 0, \quad \widehat{\Psi} \leftarrow 0$ // Generate the new constraint batch

5: **for** $q = 1, 2, \ldots, n$ **do**

6:

$$\overline{y}_q \leftarrow \operatorname*{argmax}_{y\in\mathcal{Y}} \Delta(y_q, y) + \langle \overline{W}, \psi(q, y) \rangle_{\mathsf{F}}$$

$$\widehat{\Delta} \leftarrow \widehat{\Delta} + 1/n \Delta(y_q, \overline{y}_q)$$

$$\widehat{\Psi} \leftarrow \widehat{\Psi} + 1/n \left(\psi(q, y_q) - \psi(q, \overline{y}_q)\right)$$

7: **end for**

8: $\mathcal{A} \leftarrow \mathcal{A} \cup \left\{ \left(\widehat{\Delta}, \widehat{\Psi}\right) \right\}$ // Update the constraint set

9: **until** $\widehat{\Delta} - \left\langle \overline{W}, \widehat{\Psi} \right\rangle_{\mathsf{F}} \leq \overline{\xi} + \epsilon$ // Check for convergence

For the remainder of this chapter, our focus will be on improving the speed of the inner-loop optimization (step 3). In particular, we will show that the semi-definite program can be reduced to a sequence of quadratic programs, which due to the unique structure of the 1-slack formulation, are extremely small — independent of both input dimensionality d and number of training points n — and thus efficiently solvable.

4.3 Alternating direction optimization

Rather than optimizing W via projected sub-gradient descent, we will instead use the alternating direction method of multipliers (ADMM) approach (Boyd et al., 2011). For our application, this corresponds to transforming the optimization problem (algorithm 4.1, step 3) into the following equivalent problem:

$$\begin{aligned} \min_{W,Z} \quad & f(W) + g(Z) \qquad (4.3) \\ \text{s.t.} \quad & W - Z = 0 \end{aligned}$$

$$g(Z) := \begin{cases} 0 & Z \in \mathbb{S}_+^d \\ \infty & Z \notin \mathbb{S}_+^d \end{cases}. \qquad (4.4)$$

By introducing Lagrange multipliers $Y \in \mathbb{S}^d$, the above problem gives rise to the augmented Lagrangian:

$$\mathcal{L}_\rho(W, Z, Y) := f(W) + g(Z) + \langle Y, W - Z \rangle_{\mathsf{F}} + \frac{\rho}{2}\|W - Z\|_{\mathsf{F}}^2,$$

where $\rho > 0$ is a scaling parameter.

The ADMM algorithm optimizes eq. (4.3) by alternating between updates of the primal variables (*directions*) W, Z and the scaled dual variable $U := {}^1\!/\!_\rho Y$ as follows:[2]

$$W^{t+1} \leftarrow \operatorname*{argmin}_W f(W) + \frac{\rho}{2}\|W - (Z^t - U^t)\|_{\mathsf{F}}^2 \qquad (4.5)$$

$$Z^{t+1} \leftarrow \operatorname*{argmin}_Z g(Z) + \frac{\rho}{2}\|W^{t+1} - (Z - U^t)\|_{\mathsf{F}}^2 \qquad (4.6)$$

$$U^{t+1} \leftarrow U^t + W^{t+1} - Z^{t+1}. \qquad (4.7)$$

In the W^{t+1} update, Z^t and U^t are held fixed, and the resulting update amounts to solving a convex optimization problem in W. The Z^{t+1} update simplifies to computing the orthogonal projection

$$Z^{t+1} \leftarrow \Pi_{\mathbb{S}_+^d}\left[W^{t+1} + U^t\right],$$

which is performed by thresholding the negative eigenvalues of $W^{t+1} + U^t$ at 0.

[2] This substitution simplifies notation, and the updates equate to minimizing $\mathcal{L}_\rho$ over W (eq. (4.5)) or Z (eq. (4.6)) for a fixed Y.

The ADMM algorithm amounts to repeatedly solving an unconstrained, strongly convex optimization problem (eq. (4.5)), projecting the solution onto the feasible set (eq. (4.6)), and then updating the residual (eq. (4.7)). Although, for this setting, the theoretical convergence rate of ADMM is no better than the projected sub-gradient method — $\mathcal{O}(\epsilon^{-2})$ steps for ϵ-sub-optimality — it has been empirically observed to yield satisfactory solutions after a small number (*e.g.*, tens) of steps (Boyd et al., 2011).

Qualitatively, the fundamental advantage of the ADMM approach over projected sub-gradient descent is that projections (*i.e.*, Z updates) are computed much less frequently. While the projected sub-gradient method computes projections after every step (and also during the line search to determine step size), the ADMM method only projects after solving the unconstrained problem (W step) to optimality. As we will demonstrate in section 4.4, this fact, combined with early stopping, can result in dramatic reduction in training complexity without degrading accuracy.

4.3.1 Dual optimization

The W update listed as eq. (4.5) is similar to the update in algorithm 4.1 with two modifications: 1) the constraint $W \in \mathbb{S}^d_+$ has been removed, and 2) it is strongly convex, due to the quadratic term from $\mathcal{L}_\rho$. In principle, this could be solved directly in primal form by standard techniques. However, the active set $\mathcal{A}$ is often quite small: in practical problems, $|\mathcal{A}|$ rarely exceeds 100–200 (and is often much smaller), while the number of parameters is $\mathcal{O}(d^2)$ and can easily number in the tens of thousands. This suggests that a dual formulation may lead to a more efficient algorithm.

To simplify the following derivation, let $R^t := Z^t - U^t$ and $m := |\mathcal{A}|$. The W update eq. (4.5) can be stated as the following linearly constrained quadratic program:

$$\begin{aligned} \min_{W,\xi \geq 0} \quad & \langle W, I \rangle_{\mathsf{F}} + C\xi + \frac{\rho}{2} \|W - R^t\|^2_{\mathsf{F}} \\ \text{s.t.}\, \forall (\Delta_i, \Psi_i) \in \mathcal{A}: \quad & \Delta_i - \langle W, \Psi_i \rangle_{\mathsf{F}} - \xi \leq 0. \end{aligned} \tag{4.8}$$

Introducing Lagrange multipliers $\alpha \in \mathbb{R}^m_+$, $\beta \in \mathbb{R}_+$, eq. (4.8) has the following La-

grangian:

$$\mathcal{L}(W, \xi, \alpha, \beta) := \langle W, I \rangle_{\mathsf{F}} + C\xi + \frac{\rho}{2}\|W - R^t\|_{\mathsf{F}}^2 + \sum_{i=1}^{m} \alpha_i \left(\Delta_i - \langle W, \Psi_i \rangle_{\mathsf{F}} - \xi\right) - \beta\xi. \tag{4.9}$$

Differentiating with respect to W yields the following optimality condition:

$$\begin{aligned} \nabla_W \mathcal{L}(W, \xi, \alpha, \beta) = I + \rho(W - R^t) - \sum_{i=1}^{m} \alpha_i \Psi_i = \mathbf{0} \\ \Rightarrow \quad \overline{W} = R^t + \frac{1}{\rho}\left(\sum_{i=1}^{m} \overline{\alpha}_i \Psi_i - I\right). \end{aligned} \tag{4.10}$$

Substituting eq. (4.10) into eq. (4.9) yields the dual program:

$$\begin{aligned} \sup_{\substack{\alpha \in \mathbb{R}_+^m \\ \beta \in \mathbb{R}_+}} \inf_{W, \xi} \mathcal{L}(W, \xi, \alpha, \beta) = \frac{1}{\rho} \max_{\alpha \in \mathbb{R}_+^m} \quad & -\frac{1}{2}\alpha^{\mathsf{T}} H \alpha - b^{\mathsf{T}} \alpha \\ \text{s.t.} \quad & \mathbf{1}^{\mathsf{T}} \alpha \leq C, \end{aligned} \tag{4.11}$$

with the structure kernel $H \in \mathbb{S}_+^m$ and cost vector $b \in \mathbb{R}^m$ defined as:

$$H_{ij} := \langle \Psi_i, \Psi_j \rangle_{\mathsf{F}} \tag{4.12}$$

$$b_i := \langle \rho R^t - I, \Psi_i \rangle_{\mathsf{F}} - \rho\Delta_i. \tag{4.13}$$

(Details of the derivation are deferred to section 4.A.)

Equation (4.11) is a linearly constrained quadratic program in m variables, and can easily be solved by off-the-shelf tools (*e.g.*, `qplcprog` in MATLAB). Note that the dual program is independent of both n and d, resulting in significant improvements in efficiency for large problems. After computing a dual optimum $\overline{\alpha}$, a primal optimum $\overline{W}$ can be recovered via eq. (4.10). The resulting MLR-ADMM algorithm is listed as algorithm 4.2.

Equation (4.11) bears a strong resemblance to the 1-slack structural SVM dual (Joachims et al., 2009). The key distinction lies in the definition of b, which for SVM-struct is simply the margin $-\Delta$, but here includes contributions from the linear regularization and augmented Lagrangian terms. In fact, an entire family of metric learning parameterizations can be shown to take on exactly this form (with slight variations in H

Algorithm 4.2 MLR-ADMM (step 3 of algorithm 4.1)

Input: Constraints $\mathcal{A} = \{ (\Delta, \Psi) \}$, slack parameter $C > 0$

Output: $\overline{W} \in \mathbb{S}_+^d$

$W^0 \leftarrow \mathbf{0}$, $Z^0 \leftarrow \mathbf{0}$, $U^0 \leftarrow \mathbf{0}$

for $t = 0, 1, 2, \ldots, T$ (until convergence) **do**

  $\forall i : b_i \leftarrow \langle \rho(Z^t - U^t) - I, \Psi_i \rangle_{\mathsf{F}} - \rho\Delta_i$

  $\overline{\alpha} \leftarrow \underset{\alpha}{\operatorname{argmax}}$ Equation (4.11)

  $W^{t+1} \leftarrow Z^t - U^t + 1/\rho \left(\sum_{i=1}^m \overline{\alpha}_i \Psi_i - I\right)$

  $Z^{t+1} \leftarrow \Pi_{\mathbb{S}_+^d} \left[W^{t+1} + U^t\right]$

  $U^{t+1} \leftarrow U^t + W^{t+1} - Z^{t+1}$

end for

return $\Pi_{\mathbb{S}_+^d} \left[W^T\right]$

and b), including kernel and multiple-kernel projection (section 4.3.2) and diagonally-constrained (axis-aligned) learning (section 4.B). The common dual form allows a single, generic implementation of algorithm 4.2 to be applied across all variants.

4.3.2 Multiple kernel projection

The MLR algorithm, as originally stated, learns a linear transformation of data in $\mathbb{R}^d$. However, using the techniques described in chapter 2, it has since been extended to allow for non-linear transformations via one or more kernel functions (Galleguillos et al., 2011). Let $\{ K^1, \ldots, K^p \} \subset \mathbb{S}_+^n$ denote p kernel matrices over the training set $\mathcal{X}$, with corresponding kernel functions. The multiple kernel MLR algorithm solves the following optimization problem:

$$\min_{\substack{W^1, \ldots, W^p \in \mathbb{S}_+^n \\ \xi \geq 0}} \quad \sum_{k=1}^p \left\langle W^k, K^k \right\rangle_{\mathsf{F}} + C\xi \tag{4.14}$$

$$\text{s.t.} \; \forall (\Delta_i, \Psi_i^1, \Psi_i^2, \ldots, \Psi_i^p) \in \mathcal{A} : \quad \Delta_i - \sum_{k=1}^p \left\langle W^k, \Psi_i^k \right\rangle_{\mathsf{F}} - \xi \leq 0,$$

where W^k and Ψ^k are transformations and average feature-encodings restricted to the k^{th} kernel K^k. For a test point x in this setting, the learned transformations W^k are

applied to the vectors $K_x^k \in \mathbb{R}^n$ of inner products between x and each training point (Galleguillos et al., 2011).

The ADMM algorithm of the previous section can also be easily extended to learn multiple kernel projections (MKP).[3] For each transformation W^k, we introduce an auxiliary variable Z^k and scaled Lagrange multiplier U^k just as before.[4] The corresponding MKP structure kernel H^{M} and cost vector b^{M} are defined as follows:

$$H_{ij}^{\mathsf{M}} := \sum_{k=1}^{p} \left\langle \Psi_i^k, \Psi_j^k \right\rangle_{\mathsf{F}}$$
$$b_i^{\mathsf{M}} := \left(\sum_{k=1}^{p} \left\langle \rho Z^k - \rho U^k - K^k, \Psi_i^k \right\rangle_{\mathsf{F}} \right) - \rho \Delta_i.$$

The corresponding dual optimization problem is identical to eq. (4.11) (with H and b replaced by H^{M} and b^{M}), and the primal optima are recovered by:

$$\overline{W}^k = Z^k - U^k + \frac{1}{\rho} \left(\sum_{i=1}^{m} \overline{\alpha}_i \Psi_i^k - K^k \right).$$

4.3.3 Implementation details

To set ρ, we initialize with $\rho = 1$ and use the variable scaling technique described by Boyd et al. (2011). Effectively, ρ is adapted so that for a fixed $\eta > 0$, the ratio of residual norms is bounded by

$$\frac{1}{\eta} \leq \frac{\rho \|Z^{t+1} - Z^t\|_{\mathsf{F}}}{\|Z^{t+1} - W^{t+1}\|_{\mathsf{F}}} \leq \eta. \tag{4.15}$$

We set $\eta = 10$, and if either side of eq. (4.15) is not met, ρ is scaled up or down (depending on which inequality is violated) by a factor of 2 before the next iteration. In informal experiments, we did not find the algorithm to be sensitive to these parameters.

To accelerate convergence across iterations of algorithm 4.1, each call to algorithm 4.2 is warm-started with the solutions W, Z, U, and ρ from the previous call. We also implemented optional early stopping by specifying a maximum number of iterations T.

[3] We distinguish *multiple kernel projection* from *multiple kernel learning*, wherein the goal is generally to learn a weighted combination $K' = \sum_k \mu_k K^k$.

[4] The step index superscript t is suppressed in this section to ease presentation.

4.4 Experiments

To evaluate the proposed method, we conducted two sets of experiments. The first set of experiments uses noisy versions of standard UCI data sets to test for efficiency and robustness. The second set of experiments uses realistic, multi-media data to evaluate the effects of early stopping, and the efficiency of diagonally-constrained multiple kernel projection.

4.4.1 UCI data

As a first experiment, we compare the proposed algorithm (MLR-ADMM) to the original projected sub-gradient method (MLR-Proj) on standard data sets from the UCI repository: `Balance` ($n = 625, d = 4$), `Ionosphere` ($n = 351, d = 34$), `WDBC` ($n = 569, d = 30$), and `Wine` ($n = 178, d = 13$). We also include comparisons to two standard algorithms: information-theoretic metric learning (ITML) (Davis et al., 2007) and large-margin nearest neighbor (LMNN) (Weinberger et al., 2006).

To study the effect of dimensionality on the various learning algorithms, we embed each data set into a higher-dimensional space by padding each example x_i with IID uniform binary noise x_σ:

$$x_i^\mathsf{T} \mapsto \left(x_i^\mathsf{T}, x_\sigma^\mathsf{T}\right) \in \mathbb{R}^{d+D}, \qquad x_\sigma \sim_U \{ -1, +1 \}^D .$$

This simulates a common practical scenario where a small number of informative features are hidden amongst a large number of extraneous, noisy features. Ideally, a robust metric learning algorithm should produce transformations which suppress the noisy dimensions and preserve the informative features.

Setup

For each data set, we generated 50 random 60/20/20 training/validation/test splits. Each split was padded with D-dimensional random bit noise, for each $D \in \{ 0, 2^5, 2^6, 2^7, 2^8 \}$. Each split was normalized by z-scoring with the training set statistics.

Performance was measured by 3-nearest-neighbor accuracy using the training set as examples. For ITML, the slack parameter γ was varied over $\{ 1, 10, \ldots, 10^6 \}$, and the best W was selected by validation set accuracy. For LMNN, the push-pull parameter μ was varied over $\{ 0.1, 0.2, \ldots, 0.9 \}$. For MLR, the loss function Δ was fixed to mean average precision (MAP), and C was varied over $\{ 1, 10, \ldots, 10^6 \}$. MLR-ADMM was fixed to $T = 10$ steps.

Results

Figure 4.1 displays the error rates of the various learning algorithms across all data sets and values of D. For $D = 0$, all algorithms are statistically equivalent under a Bonferroni-corrected Wilcoxon test ($\alpha = 0.05$). Not surprisingly, as D increases, the problem becomes more difficult, and test errors increase across the board. However, on all data sets and $D \geq 64$, both MLR algorithms significantly outperform ITML and LMNN.

This can be viewed, in part, as a consequence of regularization; while MLR explicitly prefers low-rank solutions (due to the trace penalty $\langle W, I \rangle_{\mathsf{F}}$), ITML prefers high-rank solutions (by minimizing $\log\det$ divergence from I), and the effects of LMNN's push-pull objective are less clear. Figure 4.2 illustrates the effective rank — *i.e.*, the number of dimensions necessary to capture 95% of the spectral mass of W — averaged across all splits of the `wine`-256 set. As expected, effective rank increases with input dimensionality, not only for ITML and LMNN, but MLR-Proj as well. Interestingly, MLR-ADMM consistently produces metrics of comparable effective rank to the noise-free case.

Figure 4.3 illustrates the reduction in the number of PSD projections from MLR-Proj to MLR-ADMM. In these data sets, n is small enough that constraint generation is less of a bottleneck than the projection step, so the projection count is a reliable proxy for runtime. As a general trend, fig. 4.3 shows at least an order of magnitude reduction in the number of projections.

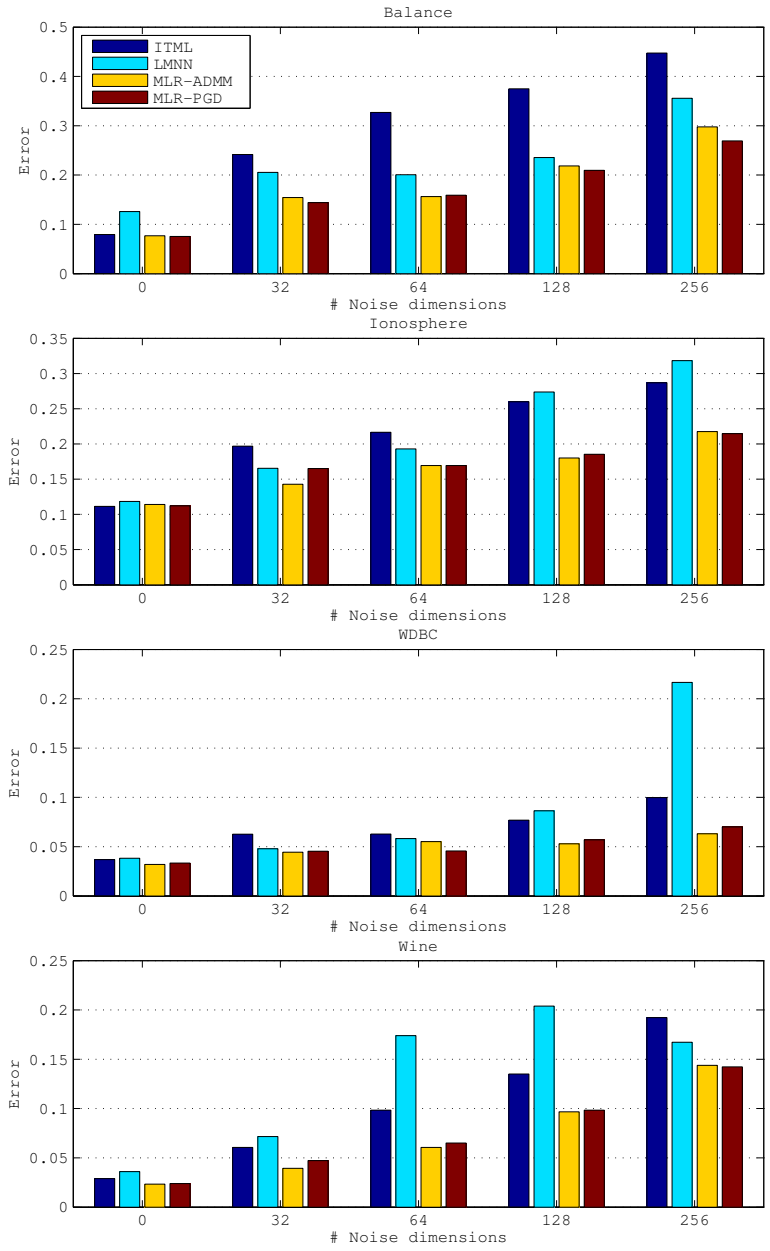


Figure 4.1: Accuracy of ITML, LMNN, MLR-ADMM and MLR-Proj as D noisy dimensions are introduced. Results are averaged across 50 random splits. In all data sets, for $D \geq 64$, both MLR methods significantly outperform ITML and LMNN under a Bonferroni-corrected Wilcoxon signed rank test ($\alpha = 0.05$).

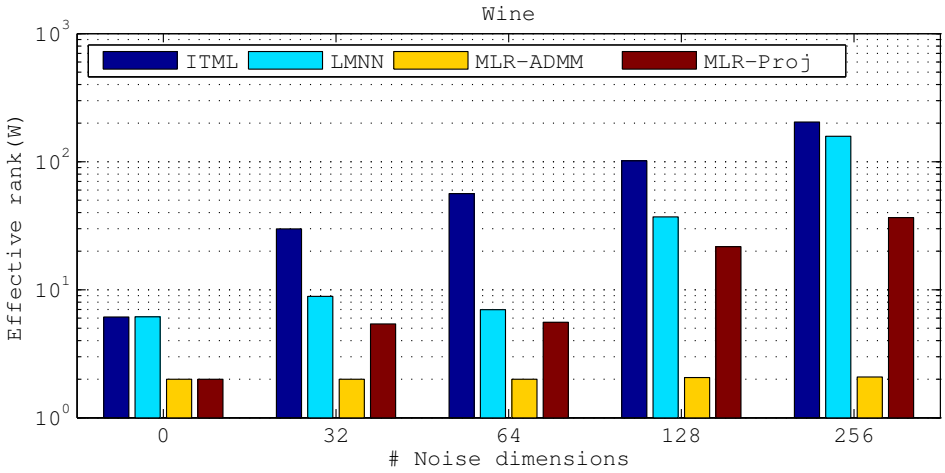


Figure 4.2: Mean effective rank of learned metrics as dimensionality increases. MLR-ADMM produces low-rank solutions across all values of D, and achieves the lowest test-error. For other methods, effective rank scales with D. This effect was consistent across data sets.

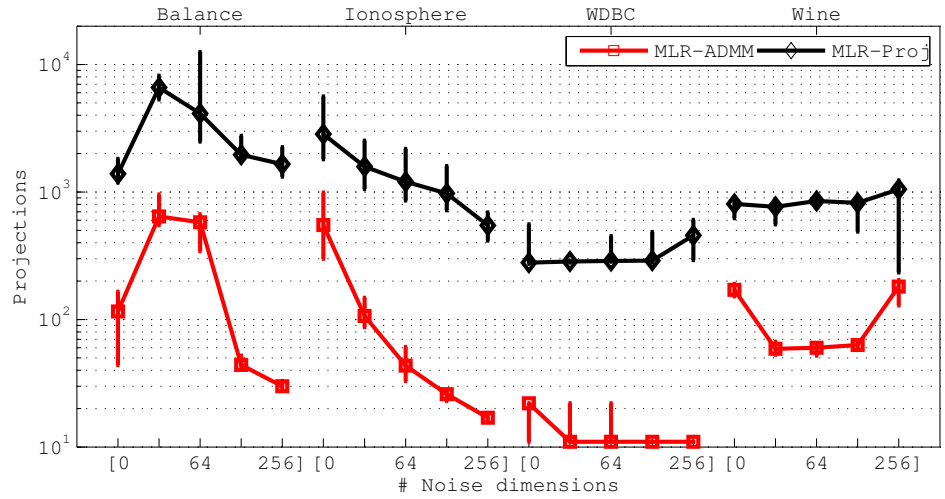


Figure 4.3: Median number of $\mathbb{S}_+^d$ projections for ADMM and projected sub-gradient descent for each data set and noise dimensionality D. Error bars correspond to 25th and 75th percentiles across 50 splits.

4.4.2 Multimedia data

Our second set of experiments involves more realistic, multi-media applications: content-based music similarity, and image segment classification. Our experiments use the following data:

Music similarity (CAL10K) (Tingle et al., 2010) (chapter 5): 5419 songs with audio features represented as vectors in $\mathbb{R}^{1024}$, partitioned into ten random 40/30/30 training/validation/test splits. For each split, each song x_i has a set of relevant training songs $\mathcal{X}_i \subseteq \mathcal{X}_{\text{train}}$, and the goal is to learn a distance function which recovers the relevant songs for a test song. Note that relevance is asymmetric and non-transitive in this data, so pair-wise methods such as ITML, and classification-based methods like LMNN do not apply.

Image segment classification (Graz-02) (Marszałek and Schmid, 2007, McFee et al., 2011): 1410 training segments and 1377 test segments, represented by six kernels, which encode texture, color, scene, and contextual interactions. Each segment belongs to one of three classes, and the goal is to maximize k-nearest neighbor accuracy.

Setup

In the music similarity experiment, we fix Δ to AUC (area under the ROC curve), and vary $C \in \{ 10^4, 10^5, 10^6 \}$. The best value of C (and corresponding metric) is selected by AUC performance on the validation set. To measure the effects of early stopping, we vary $T \in \{ 1, 5, 10, 25, 50, 100 \}$ and track the resulting accuracy, as well as the number of projection operations and calls to the constraint generator (the two key bottlenecks during training). Each split is compressed by PCA to capture 95% training set variance, resulting in dimensionality $d \approx 253 \pm 6$.

In the image segment classification problem, we constrain each W^k to be diagonal (section 4.B), fix Δ to AUC, and vary $C \in \{ 10^1, 10^2, \ldots, 10^6 \}$. Performance is measured by 5-nearest-neighbor classification, and the best value of C (and W^k) is chosen by leave-one-out cross-validation on the training set. Although the diagonal constraint greatly reduces the complexity of projecting onto the feasible set, we are inter-

ested to see if the low memory footprint of MLR-ADMM's dual optimization provides additional speedups over MLR-Proj. All experiments on this data set were conducted on a 1.73GHz Intel Core i7 with 8GB of RAM, and we compare the wall-clock time of training for ADMM and projected sub-gradient descent. We compare training time for $T \in \{ 1, 5, 10, 25 \}$.

Results

The results of the music similarity experiment are presented in fig. 4.4. As in the UCI experiments, MLR-ADMM performs comparably (and, indeed, slightly better) than MLR-Proj, across all values of T. For small values of T, MLR-ADMM requires, on average, an order of magnitude fewer projection operations than MLR-Proj: 275 ± 23.8 for $T = 10$, compared to 2733.7 ± 1443.6 for MLR-Proj. Note that for $T = 1$, the W returned at each step can be highly sub-optimal, and as a result, more cutting planes are required to converge in algorithm 4.1. However, for intermediate values of T, the number of cutting planes does not significantly differ from MLR-Proj, and speedup is directly proportional to the decrease in projections.

Table 4.1 shows the results of the image segment classification experiment. Both MLR-ADMM and MLR-Proj achieve comparable error rates, but MLR-ADMM takes significantly less time. For $T = 10$, ADMM achieves a $20.05\times$ speedup over MLR-Proj, and $30.4\times$ speedup for $T = 25$, despite the fact that the PSD projection bottleneck has been removed due the diagonal constraint. This can be explained by the fact that the dual optimization is both small and independent of the size of the input problem (in this case, $6n = 8460$). This enables the dual solver (MLR-ADMM) to spend the majority of its time operating in a small region of memory (thereby exploiting cache locality), whereas the primal solver (MLR-Proj) performs gradient updates over a large set of variables.

4.5 Conclusion

We have developed an efficient training procedure for structural metric learning. The proposed method exploits the low-dimensional structure of the dual problem, and in

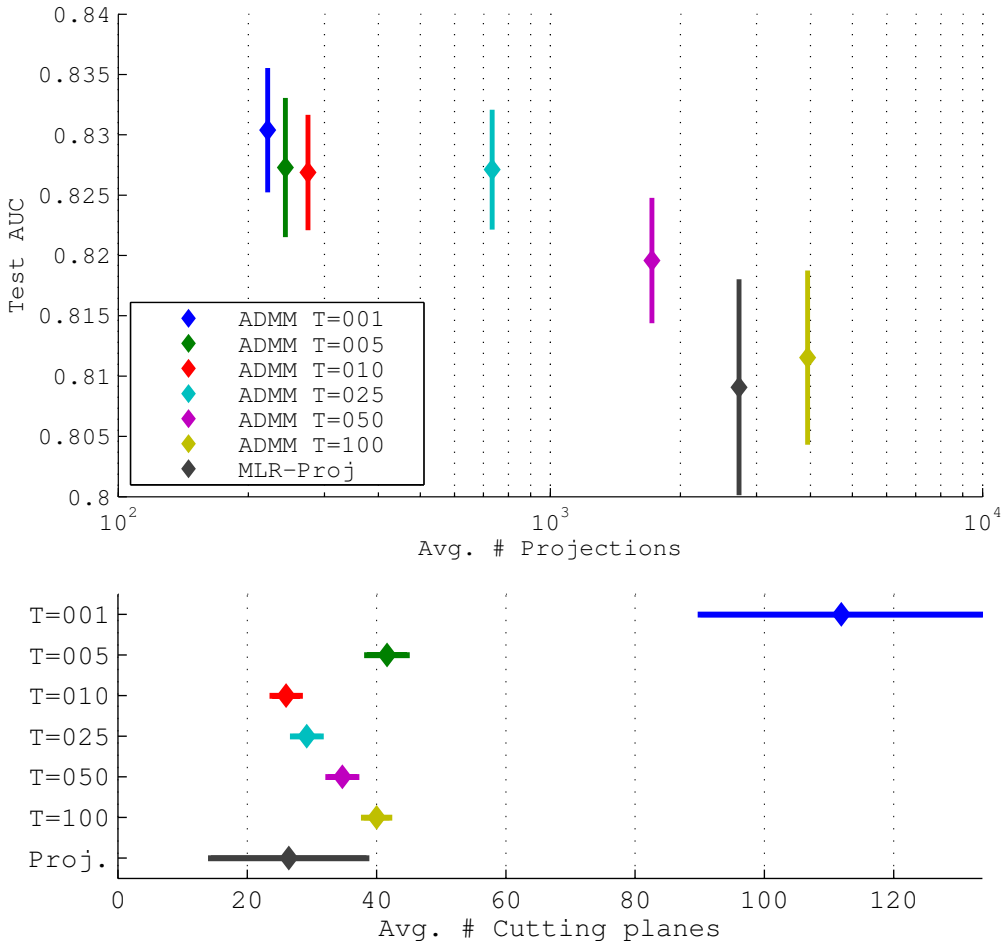


Figure 4.4: Performance of MLR-ADMM and MLR-Proj for music similarity. Top: accuracy versus the number of projections onto $\mathbb{S}_+^d$ as the number of ADMM steps T is increased. Bottom: average number of cutting planes required for each value of T. Results are averaged across ten splits; error bars correspond to ± 1 standard deviation.

Table 4.1: Test-set performance and training time for diagonal multiple kernel projection on Graz-02.

Algorithm	Error	Time (s)
MLR-ADMM $T = 001$	6.1%	105.51
MLR-ADMM $T = 005$	6.5%	27.58
MLR-ADMM $T = 010$	5.9%	22.05
MLR-ADMM $T = 025$	6.0%	14.55
MLR-Proj	5.3%	442.43

most cases, reduces by at least a factor of 10 the number of expensive spectral decomposition operations, resulting in substantial reductions in training time over the projected sub-gradient method.

4.A Derivation of eq. (4.11)

Equation (4.9) depends on W in three terms, which we treat separately here via eq. (4.10). We present the derivation for the linear case here, but the multiple kernel case follows analogously. Substituting eq. (4.10) into each term of eq. (4.9) yields

$$\left\langle \overline{W}, I \right\rangle_{\mathsf{F}} = \left\langle R^t, I \right\rangle_{\mathsf{F}} + \sum_{i=1}^{m} \frac{\overline{\alpha}_i}{\rho} \left\langle \Psi_i, I \right\rangle_{\mathsf{F}} - \frac{1}{\rho} \left\langle I, I \right\rangle_{\mathsf{F}},$$

$$\frac{\rho}{2} \left\| \overline{W} - R^t \right\|_{\mathsf{F}}^2 = \frac{1}{2\rho} \langle I, I \rangle_{\mathsf{F}} - \frac{1}{\rho} \sum_{i=1}^{m} \overline{\alpha}_i \left\langle I, \Psi_i \right\rangle_{\mathsf{F}} + \frac{1}{2\rho} \overline{\alpha}^{\mathsf{T}} H \overline{\alpha},$$

$$\sum_{i=1} \overline{\alpha}_i \left\langle \overline{W}, \Psi_i \right\rangle_{\mathsf{F}} = \sum_{i=1}^{m} \overline{\alpha}_i \left\langle R^t - \frac{1}{\rho} I, \Psi_i \right\rangle_{\mathsf{F}} + \frac{1}{\rho} \overline{\alpha}^{\mathsf{T}} H \overline{\alpha}.$$

Substituting the above into eq. (4.9) yields

$$\sup_{\substack{\alpha\in\mathbb{R}^m_+\\ \beta\in\mathbb{R}_+}} \inf_{W,\xi} \mathcal{L}(W,\xi,\alpha,\beta) = \sup_{\alpha\in\mathbb{R}^m_+} \mathcal{L}\left(\overline{W},\overline{\xi},\alpha,\overline{\beta}\right) \tag{4.16}$$
$$= \max_{\alpha\in\mathbb{R}^m_+} -\frac{1}{2\rho}\alpha^\mathsf{T} H\alpha - \frac{1}{\rho}b^\mathsf{T}\alpha,$$

where H, b are defined as in eqs. (4.12) and (4.13). Finally, differentiating eq. (4.9) with respect to ξ yields the constraint

$$\nabla_\xi \mathcal{L}(W,\xi,\alpha,\beta) = C - \beta - \mathbf{1}^\mathsf{T}\alpha = \mathbf{0}$$
$$\Rightarrow \qquad C - \overline{\beta} = \mathbf{1}^\mathsf{T}\overline{\alpha} \leq C$$

which when combined with eq. (4.16), yields eq. (4.11).

4.B Axis-aligned learning

To learn an axis-aligned transformation (*i.e.*, feature weighting), we can reformulate eq. (4.1) by replacing $W \in \mathbb{S}^d_+$ with vector $w \in \mathbb{R}^d_+$ such that $W = \operatorname{diag}(w)$. Similarly, the regularization term $\langle W, I\rangle_F$ is replaced by the equivalent penalty $\mathbf{1}^\mathsf{T} w$.

The ADMM algorithm proceeds much as before, except that the variables are now vectors $w, z, u \in \mathbb{R}^d$, rather than symmetric matrices. It is straightforward to derive the diagonally-constrained optimization problem and KKT conditions, the result of which is

$$\overline{w} = z - u + \frac{1}{\rho}\left(\sum_{i=1}^m \overline{\alpha}_i \operatorname{diag}(\Psi_i)\right) - \mathbf{1}.$$

The diagonally-constrained structure kernel H^D and cost vector b^D are defined analogously:

$$H^\mathsf{D}_{ij} := \langle \operatorname{diag}(\Psi_i), \operatorname{diag}(\Psi_j)\rangle,$$
$$b^\mathsf{D}_i := \langle \rho r - \mathbf{1}, \operatorname{diag}(\Psi_i)\rangle - \rho\Delta_i,$$

and the dual problem is again equivalent to the form given in eq. (4.11). The updates are similar to eqs. (4.5) and (4.7), and the z-step simplifies to thresholding:

$$z^{t+1} \leftarrow \Pi_{\mathbb{R}^d_+}\left[w^{t+1} + u^t\right]$$

which can be computed in $\mathcal{O}(d)$ time by thresholding at 0.

The diagonal and MKP formulations can be combined, so that each $w^k \in \mathbb{R}^n_+$ defines a weighting over $\mathcal{X}$ in the k^{th} feature space. $\overline{w}^k$ then takes the form:

$$\overline{w}^k = z^k - u^k + \frac{1}{\rho}\left(\sum_{i=1}^{m} \overline{\alpha}_i \operatorname{diag}(\Psi_i^k)\right) - \operatorname{diag}(K^k).$$

H^{MD} and b^{MD} are similarly defined:

$$H_{ij}^{\mathsf{MD}} := \sum_{k=1}^{p} \langle \operatorname{diag}(\Psi_i^k), \operatorname{diag}(\Psi_j^k) \rangle,$$
$$b_i^{\mathsf{MD}} := \sum_{k=1}^{p} \langle \rho r^k - \operatorname{diag}(K^k), \operatorname{diag}(\Psi_i^k) \rangle - \rho \Delta_i.$$

Chapter 5

Similarity from a collaborative filter

5.1 Introduction

An effective notion of similarity forms the basis of many applications involving multimedia data. For example, an online music store can benefit greatly from the development of an accurate method for automatically assessing similarity between two songs, which can in turn facilitate high-quality recommendations to a user by finding songs which are similar to her previous purchases or preferences. More generally, high-quality similarity can benefit any *query-by-example* recommendation system, wherein a user presents an example of an item that she likes, and the system responds with, *e.g.*, a ranked list of recommendations.

The most successful approaches to a wide variety of recommendation tasks — including not just music, but books, movies, *etc.*. — use *collaborative filters* (CF). Systems based on collaborative filters exploit the "wisdom of crowds" to infer similarities between items, and recommend new items to users by representing and comparing these items in terms of the people who use them (Goldberg et al., 1992). Within the domain of music information retrieval, recent studies have shown that CF systems consistently outperform alternative methods for playlist generation (Barrington et al., 2009) and semantic annotation (Kim et al., 2009). However, collaborative filters suffer from the dreaded "cold start" problem: a new item cannot be recommended until it has been purchased, and it is less likely to be purchased if it is never recommended. Thus, only a tiny fraction of songs may be recommended, making it difficult for users to explore and

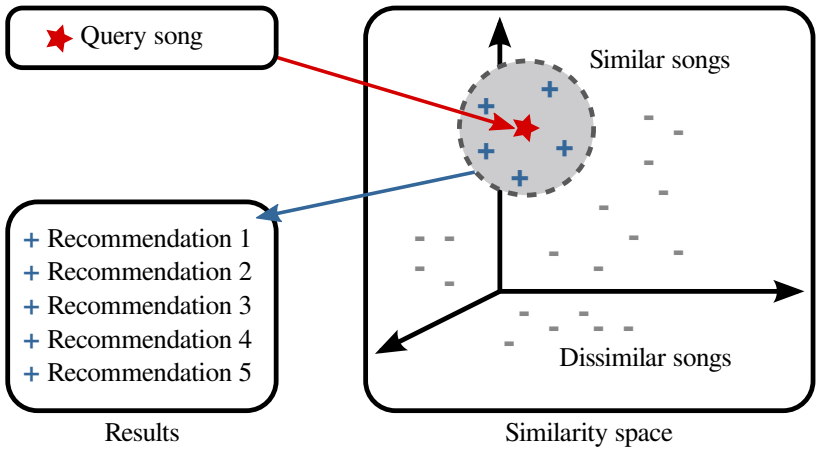


Figure 5.1: Query-by-example recommendation engines allow a user to search for new items by providing an example item. Recommendations are formed by computing the most similar items to the query item from a database of potential recommendations.

discover new music (Celma, 2010).

The cold-start problem has motivated researchers to improve *content-based* recommendation systems. Content-based systems operate on music representations that are extracted automatically from the audio content, eliminating the need for human feedback and annotation when computing similarity. While this approach naturally extends to any item regardless of popularity, the construction of features and definition of *similarity* in these systems are frequently ad-hoc and not explicitly optimized for the specific task.

In this chapter, we propose a method for optimizing content-based audio similarity by learning from a sample of collaborative filter data. Based on this optimized similarity measure, recommendations can then be made where no collaborative filter data is available. The proposed method treats similarity learning as an information retrieval problem, where similarity is learned to optimize the ranked list of results in response to a query example (fig. 5.1). Optimizing similarity for ranking requires more sophisticated machinery than, *e.g.*, genre classification for semantic search. However, the information retrieval approach offers a few key advantages, which we argue are crucial for realistic music applications. First, there are no assumptions of transitivity or symmetry in the proposed method. This allows, for example, that "The Beatles" may be considered a rel-

evant result for "Oasis", but not vice versa. Second, CF data can be collected *passively* from users by mining their listening histories, thereby directly capturing their listening habits. Finally, optimizing similarity for ranking directly attacks the main quantity of interest: the ordered list of retrieved items, rather than coarse abstractions of similarity, such as genre agreement.

5.1.1 Related work

Early studies of musical similarity followed the general strategy of first devising a model of audio content (*e.g.*, spectral clusters (Logan and Salomon, 2001), Gaussian mixture models (Aucouturier and Pachet, 2002), or latent topic assignments (Hoffman et al., 2008)), applying some reasonable distance function (*e.g.*, earth-mover's distance or Kullback-Leibler divergence), and then evaluating the proposed similarity model against some source of ground truth. Logan and Salomon (2001) and Aucouturier and Pachet (2002) evaluated against three notions of similarity between songs: same artist, same genre, and human survey data. Artist or genre agreement entail strongly binary notions of similarity, which due to symmetry and transitivity may be unrealistically coarse in practice. Survey data can encode subtle relationships between items, for example, triplets of the form *"A is more similar to B than to C"* (Aucouturier and Pachet, 2002, Ellis et al., 2002, Berenzweig et al., 2004). However, the expressive power of human survey data comes at a cost: while artist or genre meta-data is relatively inexpensive to collect for a set of songs, similarity survey data may require human feedback on a quadratic (for pairwise ratings) or cubic (for triplets) number of comparisons between songs, and is thus impractical for large data sets.

Later work in musical similarity approaches the problem in the context of supervised learning: given a set of training items (songs), and some knowledge of similarity across those items, the goal is to *learn* a similarity (distance) function that can predict pairwise similarity. Slaney et al. (2008) derive similarity from web-page co-occurrence, and evaluate several supervised and unsupervised algorithms for learning distance metrics. In chapter 2, we developed a metric learning algorithm for triplet comparisons as described above. The method proposed in this chapter follows in this line of work, but is designed to optimize structured ranking loss using the algorithm developed in chapter 3

(not just binary or triplet predictions), and uses a collaborative filter as the source of ground truth.

The idea to learn similarity from a collaborative filter follows from a series of positive results in music applications. Slaney and White (2007) demonstrate that an item-similarity metric derived from rating data matches human perception of similarity better than a content-based method. Similarly, it has been demonstrated that when combined with metric learning, collaborative filter similarity can be as effective as semantic tags for predicting survey data (chapter 2). Kim et al. (2009) demonstrated that collaborative filter similarity vastly out-performs content-based methods for predicting semantic tags. Barrington et al. (2009) conducted a user survey, and concluded that the iTunes Genius playlist algorithm (which is at least partially based on collaborative filters[1]) produces playlists of equal or higher quality than competing methods based on acoustic content or meta-data.

Finally, there has been some previous work addressing the cold-start problem of collaborative filters for music recommendation by integrating audio content. Yoshii et al. (2008) formulate a joint probabilistic model of both audio content and collaborative filter data in order to predict user ratings of songs (using either or both representations), whereas our goal here is to use audio data to predict the similarities derived from a collaborative filter. Our problem setting is most similar to that of Stenzel and Kamps (2005), wherein a CF matrix was derived from playlist data, clustered into latent "pseudo-genres," and classifiers were trained to predict the cluster membership of songs from audio data. Our proposed setting differs in that we derive similarity at the user level (not playlist level), and automatically learn the content-based song similarity that directly optimizes the primary quantity of interest in an information retrieval system: the quality of the rankings it induces.

5.1.2 Contributions

Our primary contribution in this chapter is a framework for improving content-based audio similarity by learning from a sample of collaborative filter data. Toward this end, we first develop a method for deriving item similarity from a sample of col-

[1] http://www.apple.com/pr/library/2008/09/09itunes.html

laborative filter data. We then use the sample similarity to train an optimal distance metric over audio descriptors. More precisely, a distance metric is optimized to produce high-quality rankings of the training sample in a query-by-example setting. The resulting distance metric can then be applied to previously unseen data for which collaborative filter data is unavailable. Experimental results verify that the proposed methods significantly outperform competing methods for content-based music retrieval.

5.2 Learning similarity

The main focus of this chapter is the following information retrieval problem: given a *query* song q, return a ranked list from a database $\mathcal{X}$ of n songs ordered by descending similarity to q. In general, the query may be previously unseen to the system, but $\mathcal{X}$ will remain fixed across all queries. We will assume that each song is represented by a vector in $\mathbb{R}^d$, and similarity is computed by Euclidean distance. Thus, for any query q, a natural ordering of $x \in \mathcal{X}$ is generated by sorting according to increasing distance from q: $\|q - x\|$.

Given some side information describing the similarity relationships between items of $\mathcal{X}$, distance-based ranking can be improved by applying a *metric learning* algorithm. Rather than rely on native Euclidean distance, the learning algorithm produces a matrix $W \in \mathbb{S}_+^d$ which characterizes an optimized distance:

$$\|q - x\|_W = \sqrt{(q - x)^\mathsf{T} W (q - x)}. \tag{5.1}$$

In order to learn W, we will apply the metric learning to rank (MLR) algorithm (chapter 3). At a high level, MLR optimizes the distance metric W on $\mathcal{X}$, *i.e.*, so that W generates optimal rankings of songs in $\mathcal{X}$ when using each song $q \in \mathcal{X}$ as a query. To apply the algorithm, we must provide a set of similar songs $x \in \mathcal{X}$ for each *training* query $q \in \mathcal{X}$. This is achieved by leveraging the side information that is available for items in $\mathcal{X}$. More specifically, we will derive a notion of similarity from collaborative filter data on $\mathcal{X}$. So, the proposed approach optimizes content-based audio similarity by learning from a sample of collaborative filter data.

5.2.1 Collaborative filters

The term *collaborative filter* (CF) is generally used to denote to a wide variety of techniques for modeling the interactions between a set of items and a set of users (Goldberg et al., 1992, Sarwar et al., 2001). Often, these interactions are modeled as a (typically sparse) matrix F where rows represent the users, and columns represent the items. The entry F_{ij} encodes the interaction between user i and item j.

The majority of work in the CF literature deals with F derived from explicit user feedback, *e.g.*, 5-star ratings (Slaney and White, 2007, Yoshii et al., 2008). While rating data can provide highly accurate representations of user-item affinity, it also has drawbacks, especially in the domain of music. First, explicit ratings require active participation on behalf of users. This may be acceptable for long-form content such as films, in which the time required for a user to rate an item is miniscule relative to the time required to consume it. However, for short-form content (*e.g.*, songs), it seems unrealistic to expect a user to rate more than a small fraction of the items consumed. Second, the scale of rating data is often arbitrary, skewed toward the extremes (*e.g.*, 1- and 5-star ratings), and may require careful calibration to use effectively (Slaney and White, 2007).

Alternatively, CF data can also be derived from *implicit* feedback. While somewhat noisier on a per-user basis than explicit feedback, implicit feedback can be derived in much higher volumes by simply counting how often a user interacts with an item (*e.g.*, listens to an artist) (Deshpande and Karypis, 2004, Hu et al., 2008). Implicit feedback differs from rating data, in that it is positive and unbounded, and it does not facilitate explicit negative feedback. As suggested by Hu et al. (2008), binarizing an implicit feedback matrix by thresholding can provide an effective mechanism to infer positive associations.

In a binary CF matrix F, each column F_j can be interpreted as a *bag-of-users* representation of item j. Of central interest in this chapter is the similarity between items (*i.e.*, columns of F). We define the similarity between two items i, j as the Jaccard

index (Jaccard, 1901) of their user sets:

$$S(i,j) := \frac{|F_i \cap F_j|}{|F_i \cup F_j|} = \frac{F_i^\mathsf{T} F_j}{|F_i| + |F_j| - F_i^\mathsf{T} F_j}, \tag{5.2}$$

which counts the number of users shared between items i and j, and normalizes by the total number of users of i and j combined.

Equation (5.2) defines a quantitative metric of similarity between two items. However, for information retrieval applications, we are primarily interested in the most similar (relevant) items for any query. We therefore define the *relevant* set $\mathcal{X}_q^+$ for any item q as the top k most similar items according to eq. (5.2), *i.e.*, those items which a user of the system would be shown first. Although binarizing similarity in this way does simplify the notion of relevance, it still provides a flexible language for encoding relationships between items. Note that after thresholding, transitivity and symmetry are not enforced, so it is possible, *e.g.*, for *The Beatles* to be relevant for *Oasis* but not vice versa. Consequently, we will need a learning algorithm which can support such flexible encodings of relevance.

5.3 Audio representation

In order to compactly summarize audio signals, we represent each song as a histogram over a dictionary of timbral *codewords*. This general strategy has been proven effective in computer vision applications (Fei-Fei and Perona, 2005), as well as audio and music classification (Sundaram and Narayanan, 2008, Seyerlehner et al., 2008, Hoffman et al., 2009). The efficiency and ease of implementation of the codeword histogram approach makes it an attractive choice for audio representation.

As a first step, a *codebook* is constructed by clustering a large collection of feature descriptors (section 5.3.1). Once the codebook has been constructed, each song is summarized by aggregating vector quantization (VQ) representations across all frames in the song, resulting in *codeword histograms* (section 5.3.2). Finally, histograms are represented in a non-linear kernel space to facilitate better learning with MLR (section 5.3.3).

5.3.1 Codebook training

Our general approach to constructing a codebook for vector quantization is to aggregate audio feature descriptors from a large pool of songs into a single bag-of-features, which is then clustered to produce the codebook.

For each song x in the codebook training set $\mathcal{X}_C$ — which may generally be distinct from the MLR training set $\mathcal{X}$ — we compute the first 13 Mel frequency cepstral coefficients (MFCCs) (Rabiner and Juang, 1993) from each half-overlapping 23ms frame. From the time series of MFCC vectors, we compute the first and second instantaneous derivatives, which are concatenated to form a sequence of 39-dimensional dynamic MFCC (ΔMFCC) vectors (Buchanan, 1995). These descriptors are then aggregated across all $x \in \mathcal{X}_C$ to form an unordered bag of features Z.

To correct for changes in scale across different ΔMFCC dimensions, each vector $z \in Z$ is normalized according to the sample mean $\mu \in \mathbb{R}^{39}$ and standard deviation $\sigma \in \mathbb{R}^{39}$ estimated from Z. Each sample z is transformed accordingly:

$$z \mapsto \mathrm{diag}(\sigma)^{-1}(z - \mu). \tag{5.3}$$

The normalized ΔMFCC vectors are then clustered into a set $\mathcal{V}$ of codewords by k-means. (Specifically, an online variant of Hartigan's method (Hartigan, 1975) described in appendix B).

5.3.2 (Top-τ) Vector quantization

Once the codebook $\mathcal{V}$ has been constructed, a song x is represented as a histogram h_x over the codewords in $\mathcal{V}$. This proceeds in three steps:

1. a bag-of-features is computed from x's ΔMFCCs, denoted as $x = \{ x_i \} \subset \mathbb{R}^{39}$;

2. each $x_i \in x$ is normalized according to eq. (5.3);

3. the codeword histogram is constructed by counting the frequency with which each

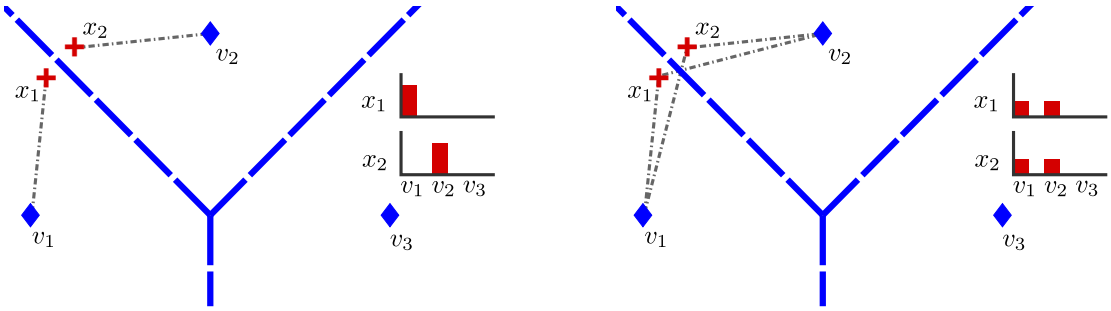


Figure 5.2: Two close data points x_1, x_2 (+) and the Voronoi partition for three VQ codewords v_1, v_2, v_3 (♦). Left: hard VQ ($\tau = 1$) assigns similar data points to dissimilar histograms. Right: assigning each data point to its top $\tau = 2$ codewords reduces noise in codeword histogram representations.

codeword $v \in \mathcal{V}$ quantizes an element of x:[2]

$$[h_x]_v := \frac{1}{|x|} \sum_{x_i \in x} \left[\!\!\left[v = \operatorname*{argmin}_{u \in \mathcal{V}} \|x_i - u\| \right]\!\!\right] . \tag{5.4}$$

Codeword histograms are normalized by the number of frames $|x|$ in the song in order to ensure comparability between songs of different lengths; h_x may therefore be interpreted as a multinomial distribution over codewords.

Equation (5.4) derives from the standard notion of vector quantization (VQ) where each vector (*e.g.*, data point x_i) is replaced by its closest quantizer. However, VQ can become unstable when a vector has multiple, (approximately) equidistant quantizers (fig. 5.2, left), which is more likely to happen as the size of the codebook increases.

To counteract quantization errors, we generalize eq. (5.4) to support *multiple* quantizers for each vector. For a vector x_i, a codebook $\mathcal{V}$, and a *quantization threshold* $\tau \in \{ 1, 2, \ldots, |\mathcal{V}| \}$, we define the quantization set

$$\operatorname*{argmin}_{u \in \mathcal{V}}{}^{\tau} \|x_i - u\| := \{ u \text{ is a } \tau\text{-nearest neighbor of } x_i \} .$$

The *top-*τ codeword histogram for a song x is then constructed as

$$[h_x^\tau]_v := \frac{1}{|x|} \sum_{x_i \in x} \frac{1}{\tau} \left[\!\!\left[v \in \operatorname*{argmin}_{u \in \mathcal{V}}{}^{\tau} \|x_i - u\| \right]\!\!\right] . \tag{5.5}$$

[2]To simplify notation, we denote by $[h_x]_v$ the bin of histogram h_x corresponding to the codeword $v \in \mathcal{V}$. Codewords are assumed to be unique, and the usage should be clear from context.

Intuitively, eq. (5.5) assigns $1/\tau$ mass to each of the τ closest codewords for each $x_i \in x$ (fig. 5.2, right). Note that when $\tau = 1$, eq. (5.5) reduces to eq. (5.4). The normalization by $1/\tau$ ensures that $\sum_v [h_x^\tau]_v = 1$, so that for $\tau > 1$, h_x^τ retains its interpretation as a multinomial distribution over $\mathcal{V}$.

It should be noted that top-τ is by no means the only way to handle over-quantization errors. In particular, the hierarchical Dirichlet process (HDP) method proposed by Hoffman et al. (2008) addresses the quantization error problem (by using a probabilistic encoding), as well as the issue of determining the size of the codebook, and could easily be substituted into our framework. However, as demonstrated in section 5.4, algorithm 3.1 adequately compensates for these effects. For the sake of simplicity and ease of reproducibility, we opted here to use the top-τ method.

5.3.3 Histogram representation and distance

After summarizing each song x by a codeword histogram h_x^τ, these histograms may be interpreted as vectors in $\mathbb{R}^{|\mathcal{V}|}$. Subsequently, for a query song q, retrieval may be performed by ordering $x \in \mathcal{X}$ according to increasing (Euclidean) distance $\|h_q^\tau - h_x^\tau\|$. After optimizing W with algorithm 3.1, the same codeword histogram vectors may be used to perform retrieval with respect to the learned metric $\|h_q^\tau - h_x^\tau\|_W$.

However, treating codeword histograms directly as vectors in a Euclidean space ignores the simplicial structure of multinomial distributions. To better exploit the geometry of codeword histograms, we represent each histogram in a *probability product kernel* (PPK) space (Jebara et al., 2004). Inner products in this space can be computed by evaluating the corresponding *kernel function* k. For PPK space, k is defined as:

$$k(h_q^\tau, h_x^\tau) := \sum_{v \in \mathcal{V}} \sqrt{[h_q^\tau]_v \cdot [h_x^\tau]_v}. \tag{5.6}$$

The PPK inner product in eq. (5.6) is equivalent to the Bhattacharyya coefficient (Bhattacharyya, 1943) between h_q^τ and h_x^τ. Consequently, distance in PPK space induces the same rankings as Hellinger distance between histograms.

Typically in kernel methods, data is represented implicitly in a (typically high-dimensional) Hilbert space via the $n \times n$ matrix of inner products between training points, *i.e.*, the *kernel matrix* (Schölkopf and Smola, 2002). This representation enables

efficient learning, even when the dimensionality of the kernel space is much larger than the number of points (*e.g.*, for histogram-intersection kernels (Barla et al., 2003)) or infinite (*e.g.*, radial basis functions). The MLR algorithm has been extended to support optimization of distances in such spaces by reformulating the optimization in terms of the kernel matrix, and optimizing an $n \times n$ matrix $W \in \mathbb{S}_+^n$ (Galleguillos et al., 2011). While kernel MLR supports optimization in arbitrary inner product spaces, it can be difficult to scale up to large training sets (*i.e.*, large n), and may require making some simplifying approximations to scale up, *e.g.*, restricting W to be diagonal.

However, for the present application, we can exploit the specific structure of the probability product kernel (on histograms) and optimize distances in PPK space with complexity that depends on $|\mathcal{V}|$ rather than n, thereby supporting larger training sets. Note that PPK enables an *explicit* representation of the data according to a simple, coordinate-wise transformation:

$$[h_x^\tau]_v \mapsto \sqrt{[h_x^\tau]_v}, \tag{5.7}$$

which, since $k(h_x^\tau, h_x^\tau) = 1$ for all h_x^τ, can be interpreted as mapping the $|\mathcal{V}|$-dimensional simplex to the $|\mathcal{V}|$-dimensional unit sphere. Training data may therefore be represented as a $|\mathcal{V}| \times n$ data matrix, rather than the $n \times n$ kernel matrix. As a result, we can equivalently apply eq. (5.7) to the data, and learn a $|\mathcal{V}| \times |\mathcal{V}|$ matrix W with algorithm 3.1, which is more efficient than using kernel MLR when $|\mathcal{V}| < n$, as is often the case in our experiments. Moreover, the probability product kernel does not require setting hyper-parameters (*e.g.*, the bandwidth of a radial basis function kernel), and thus simplifies the training procedure.

5.4 Experiments

Our experiments are designed to simulate query-by-example content-based retrieval of songs from a fixed database. Figure 5.3 illustrates the high-level experimental setup: training and evaluation are conducted with respect to collaborative filter similarity (as described in section 5.2.1). In this section, we describe the sources of collaborative filter and audio data, experimental procedure, and competing methods against which we compare.

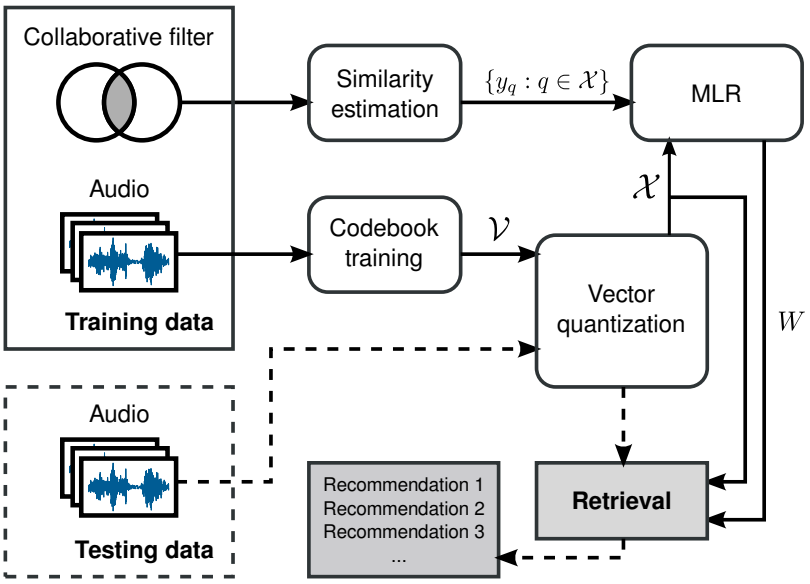


Figure 5.3: Schematic diagram of training and retrieval. Here, "training data" encompasses both the subset of $\mathcal{X}$ used to train the metric W, and the codebook set $\mathcal{X}_C$ used to build the codebook $\mathcal{V}$. While, in our experiments, both sets are disjoint, in general, data used to build the codebook may also be used to train the metric.

5.4.1 Data

Collaborative filter: Last.FM

Our collaborative filter data is provided by Last.fm,[3] and was collected by Celma (2010). The data consists of a users-by-artists matrix F of 359,347 unique users and 186,642 unique, identifiable artists; the entry F_{ij} contains the number of times user i listened to artist j. We binarize the matrix by thresholding at 10, *i.e.*, a user must listen to an artist at least 10 times before we consider the association meaningful.

Audio: CAL10K

For our audio data, we use the CAL10K data set (Tingle et al., 2010). Starting from 10,832 songs by 4,661 unique artists, we first partition the set of artists into those with at least 100 listeners in the binarized CF matrix (2015, the *experiment set*),

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

and those with fewer than 100 listeners (2646, the *codebook set*). We then restrict the CF matrix to just those 2015 artists in the experiment set, with sufficiently many listeners. From this restricted CF matrix, we compute the artist-by-artist similarity matrix according to eq. (5.2).

Artists in the codebook set, with insufficiently many listeners, are held out from the experiments in section 5.4.2, but their songs are used to construct four codebooks as described in section 5.3.1. From each held out artist, we randomly select one song, and extract a 5-second sequence of ΔMFCC vectors (431 half-overlapping 23ms frames at 22050Hz). These samples are collected into a bag-of-features of approximately 1.1 million samples, which is randomly permuted, and clustered via online k-means in a single pass to build four codebooks of sizes $|\mathcal{V}| \in \{ 256, 512, 1024, 2048 \}$, respectively. Cluster centers are initialized to the first (randomly selected) k points. Note that *only* the artists from the codebook set (and thus no artists from the experiment set) are used to construct the codebooks. As a result, the previous four codebooks are fixed throughout the experiments in the following section.

5.4.2 Procedure

For our experiments, we generate 10 random splits of the experiment set of 2015 artists into 40% training, 30% validation and 30% test *artists*.[4] For each split, the set of all training artist songs forms the *training set*, which serves as the database of "known" songs, $\mathcal{X}$. For each split, and for each (training/test/validation) artist, we then define the *relevant artist set* as the top 10 most similar *training*[5] artists. Finally, for any song q by artist i, we define q's *relevant song set*, $\mathcal{X}_q^+$, as all songs by all artists in i's relevant artist set. The songs by all other training artists, not in i's relevant artist set, are collected into $\mathcal{X}_q^-$, the set of irrelevant songs for q. The statistics of the training, validation, and test splits are collected in table 5.1.

For each of the four codebooks, constructed in the previous section, each song

[4] Due to recording artifacts and our definition of similarity, it is crucial to split at the level of artists rather than songs (Whitman et al., 2001).

[5] Also for test and validation artists, we restrict the relevant artist set to the training artists to mimic the realistic setting of retrieving "known" songs from $\mathcal{X}$, given an "unknown" (test/validation) query.

Table 5.1: Statistics of CAL10K data, averaged across ten random training/validation/test splits. *# Relevant* is the average number of relevant songs for each training/validation/test song.

	Training	Validation	Test
# Artists	806	604	605
# Songs	2122.3 ± 36.3	1589.3 ± 38.6	1607.5 ± 64.3
# Relevant	36.9 ± 16.4	36.4 ± 15.4	37.1 ± 16.0

was represented by a histogram over codewords using eq. (5.5), with

$$\tau \in \{ 1, 2, 4, 8 \}.$$

Codeword histograms were then mapped into PPK space by eq. (5.7). For comparison purposes, we also experiment with Euclidean distance and MLR on the raw codeword histograms.

To train the distance metric with algorithm 3.1, we vary

$$C \in \left\{ 10^{-2}, 10^{-1}, \cdots, 10^{9} \right\}.$$

We experiment with three ranking losses Δ for training: area under the ROC curve (AUC), which captures global qualities of the ranking, but penalizes mistakes equally regardless of position in the ranking; normalized discounted cumulative gain (NDCG), which applies larger penalties to mistakes at the beginning of the ranking than at the end, and is therefore more localized than AUC; and mean reciprocal rank (MRR), which is determined by the position of the first relevant result, and is therefore the most localized ranking loss under consideration here. After learning W on the training set, retrieval is evaluated on the validation set, and the parameter setting (C, Δ) which achieves highest AUC on the validation set is then evaluated on the test set.

To evaluate a metric W, the training set $\mathcal{X}$ is ranked according to distance from each test (validation) song q under W, and we record the mean AUC of the rankings over all test (validation) songs.

Prior to training with MLR, codeword histograms are compressed via principal components analysis (PCA) to capture 95% of the variance as estimated on the training

set. While primarily done for computational efficiency, this step is similar to the latent perceptual indexing method described by Sundaram and Narayanan (2008), and may also be interpreted as de-noising the codeword histogram representations. In preliminary experiments, compression of codeword histograms was not observed to significantly affect retrieval accuracy in either the native or PPK spaces (without MLR optimization).

5.4.3 Comparisons

To evaluate the performance of the proposed system, we compare to several alternative methods for content-based query-by-example song retrieval: first, similarity derived from comparing Gaussian mixture models of ΔMFCCs; second, an alternative (unsupervised) weighting of VQ codewords; and third, a high-level, automatic semantic annotation method. We also include a comparison to a manual semantic annotation method (*i.e.*, driven by human experts), which although not content-based, can provide an estimate of an upper bound on what can be achieved by content-based methods. For both manual and automatic semantic annotations, we will also compare to their MLR-optimized counterparts.

Gaussian mixtures

From each song, a Gaussian mixture model (GMM) over its ΔMFCCs was estimated via expectation-maximization (Dempster et al., 1977). Following Turnbull et al. (2008), each song is represented by a GMM consisting of 8 components with diagonal covariance matrices.[6] The training set $\mathcal{X}$ is therefore represented as a collection of GMM distributions $\{ p_x \mid x \in \mathcal{X} \}$. This approach is representative of many previously proposed systems in the music information retrieval literature (Aucouturier and Pachet, 2002, Berenzweig et al., 2004, Jensen et al., 2007), and is intended to serve as a baseline against which we can compare the proposed VQ approach.

At test time, given a query song q, we first estimate its GMM p_q. We would then like to rank each $x \in \mathcal{X}$ by increasing Kullback-Leibler (KL) divergence (Kullback,

[6] In addition to yielding the best performance for the auto-tagger described by Turnbull et al. (2008), 8-component diagonal covariance GMMs yields audio representations of comparable space complexity to the proposed VQ approach.

1968) from p_q:

$$\mathbf{D}(p_q \| p_x) := \int p_q(z) \log \frac{p_q(z)}{p_x(z)} \mathrm{d}z. \tag{5.8}$$

However, we do not have a closed-form expression for KL divergence between GMMs, so we must resort to approximate methods. Several such approximation schemes have been devised in recent years, including variational methods and sampling approaches (Hershey and Olsen, 2007). Here, we opt for the Monte Carlo approximation:

$$\mathbf{D}(p_q \| p_x) \approx \sum_{i=1}^{m} \frac{1}{m} \log \frac{p_q(z_i)}{p_x(z_i)}, \tag{5.9}$$

where $\{ z_i \}_{i=1}^m$ is a collection of m independent samples drawn from p_q. Although the Monte Carlo approximation is considerably slower than closed-form approximations (*e.g.*, variational methods), with enough samples, it often exhibits higher accuracy (Hershey and Olsen, 2007, Jensen et al., 2007). Note that because we are only interested in the rank-ordering of $\mathcal{X}$ given p_q, it is equivalent to order each $p_x \in \mathcal{X}$ by increasing (approximate) cross-entropy:

$$H(p_q, p_x) := \int p_q(z) \log \frac{1}{p_x(z)} \mathrm{d}z \approx \sum_{i=1}^{m} \frac{1}{m} \log \frac{1}{p_x(z_i)}. \tag{5.10}$$

For efficiency purposes, for each query q we fix the sample $\{ z_i \}_{i=1}^m \sim p_q$ across all $x \in \mathcal{X}$. We use $m = 2048$ samples for each query, which was found to yield stable cross-entropy estimates in an informal, preliminary experiment.

TF-IDF

The algorithm described in chapter 3 is a supervised approach to learning an optimal transformation of feature descriptors (in this specific case, VQ histograms). Alternatively, it is common to use the natural statistics of the data in an unsupervised fashion to transform the feature descriptors. As a baseline, we compare to the standard method of combining *term frequency–inverse document frequency* (TF-IDF) (Salton and Buckley, 1987) representations with cosine similarity, which is commonly used with both text (Salton and Buckley, 1987) and codeword representations (Sivic and Zisserman, 2003).

Given a codeword histogram h_q^τ, for each $v \in \mathcal{V}$, $[h_q^\tau]_v$ is mapped to its TF-IDF value by[7]

$$[h_q^\tau]_v \mapsto [h_q^\tau]_v \cdot \text{IDF}_v, \tag{5.11}$$

where IDF_v is computed from the statistics of the training set by[8]

$$\text{IDF}_v := \log \frac{|\mathcal{X}|}{|\{ x \in \mathcal{X} \mid x_v > 0 \}|}. \tag{5.12}$$

Intuitively, IDF_v assigns more weight to codewords v which appear in fewer songs, and reduces the importance of codewords appearing in many songs. The training set $\mathcal{X}$ is accordingly represented by TF-IDF vectors. At test time, each $x \in \mathcal{X}$ is ranked according to decreasing cosine-similarity to the query q:

$$\cos(h_q^\tau, h_x^\tau) = \frac{\left\langle h_q^\tau, h_x^\tau \right\rangle}{\|h_q^\tau\| \cdot \|h_x^\tau\|}. \tag{5.13}$$

Automatic semantic tags

The proposed method relies on low-level descriptors to assess similarity between songs. Alternatively, similarity may be assessed by comparing high-level content descriptors in the form of *semantic tags*. These tags may include words to describe genre, instrumentation, emotion, *etc.*. Because semantic annotations may not be available for novel query songs, we restrict attention to algorithms which automatically predict tags given only audio content.

In our experiments, we adapt the auto-tagging method proposed by Turnbull et al. (2008). This method summarizes each song by a *semantic multinomial distribution* (SMD) over a vocabulary of 149 tag words. Each tag t is characterized by a GMM p_t over ΔMFCC vectors, each of which was trained previously on the CAL500 data set (Turnbull et al., 2007). A song q is summarized by a multinomial distribution s_q, where the t^{th} entry is computed by the geometric mean of the likelihood of q's ΔMFCC vectors q_i under p_t:

$$[s_q]_t \propto \left(\prod_{q_i \in q} p_t(q_i) \right)^{1/|q|}. \tag{5.14}$$

[7] Since codeword histograms are pre-normalized, there is no need to re-compute the term frequency in eq. (5.11).

[8] To avoid division by 0, we define $\text{IDF}_v := 0$ for any codeword v which does not appear in the training set.

(Each SMD s_q is normalized to sum to 1.) The training set $\mathcal{X}$ is thus described as a collection of SMDs $\{ s_x \mid x \in \mathcal{X} \}$. At test time, $\mathcal{X}$ is ranked according to increasing distance from the test query under the probability product kernel[9] as described in section 5.3.3. This representation is also amenable to optimization by MLR, and we will compare to retrieval performance after optimizing PPK representations of SMDs with MLR.

Human tags

Our final comparison uses semantic annotations manually produced by humans, and may be interpreted as an upper bound on the performance of automated content analysis. Each song in CAL10K includes a partially observed, binary annotation vector over a vocabulary of 1053 tags from the Music Genome Project.[10] The annotation vectors are *weak* in the sense that a 1 indicates that the tag applies, while a 0 indicates only that the tag *may not* apply.

In our experiments, we observed the best performance by using cosine similarity as the retrieval function, although we also tested TF-IDF and Euclidean distances. As in the auto-tag case, we will also compare to tag vectors after optimization by MLR. When training with MLR, annotation vectors were compressed via PCA to capture 95% of the training set variance.

5.5 Results

Vector quantization

In a first series of experiments, we evaluate various approaches and configurations based on VQ codeword histograms. Figure 5.4 lists the AUC achieved by four different approaches (*Native*, *TF-IDF*, *MLR*, *PPK-MLR*), based on VQ codeword histograms, for each of four codebook sizes and each of four quantization thresholds. We

[9] We also experimented with χ^2-distance, ℓ_1, Euclidean, and (symmetrized) KL divergence, but PPK distance was always statistically equivalent to the best-performing distance.

[10] http://www.pandora.com/mgp.shtml

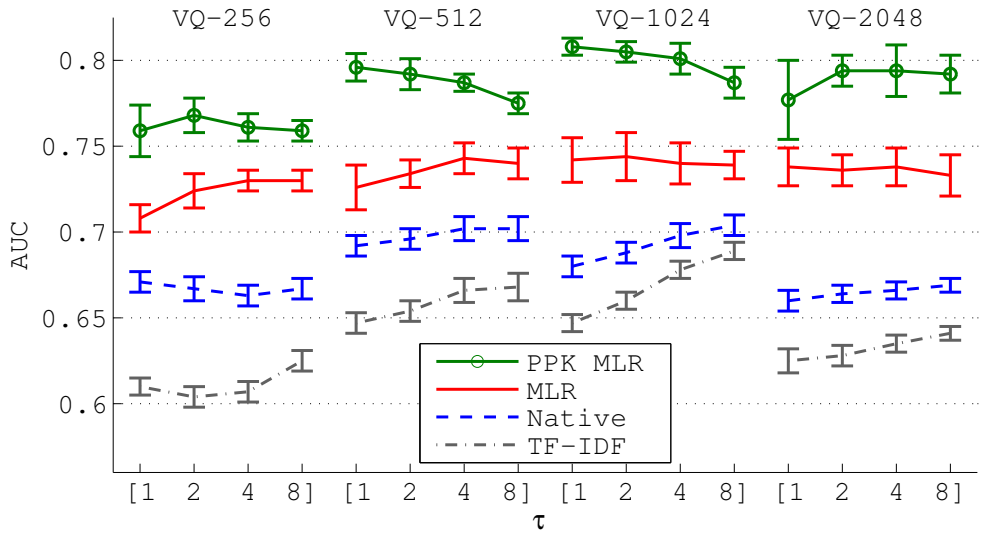


Figure 5.4: Retrieval accuracy with vector quantized ΔMFCC representations. Each grouping corresponds to a different codebook size $|\mathcal{V}| \in \{ 256, 512, 1024, 2048 \}$. Each point within a group corresponds to a different quantization threshold $\tau \in \{ 1, 2, 4, 8 \}$. Error bars correspond to one standard deviation across trials.

observe that using Euclidean distance on raw codeword histograms[11] (*Native*) yields significantly higher performance for codebooks of intermediate size (512 or 1024) than for small (256) or large (2048) codebooks. For the 1024 codebook, increasing τ results in significant gains in performance, but it does not exceed the performance for the 512 codebook. The decrease in accuracy for $|\mathcal{V}| = 2048$ suggests that performance is indeed sensitive to overly large codebooks.

After learning an optimal distance metric with MLR on raw histograms (*i.e.*, not PPK representations) (*MLR*), we observe two interesting effects. First, MLR optimization always yields significantly better performance than the native Euclidean distance. Second, performance is much less sensitive to the choice of codebook size and quantization threshold: all settings of τ for codebooks of size at least $|\mathcal{V}| \geq 512$ achieve statistically equivalent performance.

Finally, we observe the highest performance by combining the PPK representa-

[11] For clarity, we omit the performance curves for native Euclidean distance on PPK representations, as they do not differ significantly from the *Native* curves shown.

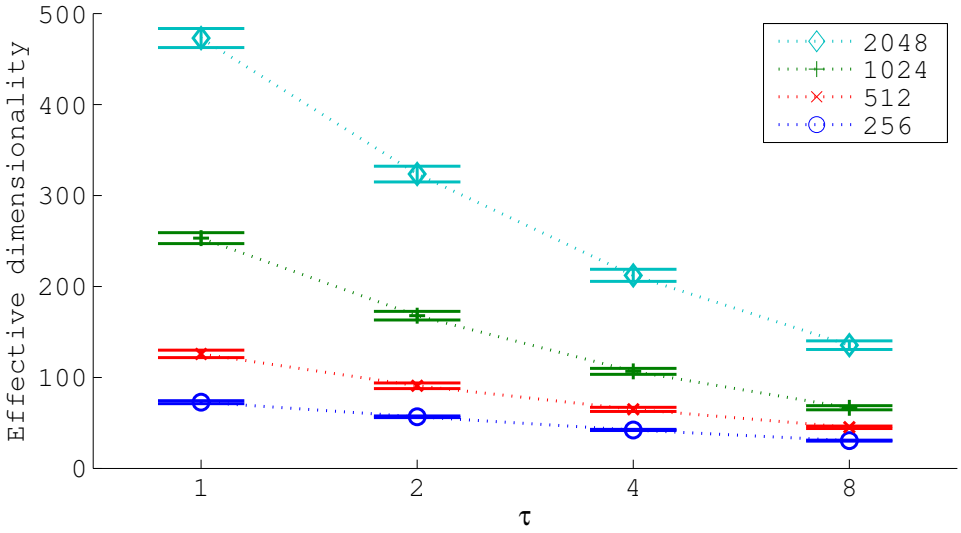


Figure 5.5: The *effective dimensionality* of codeword histograms in PPK space, *i.e.*, the number of principal components necessary to capture 95% of the training set's variance, as a function of the quantization threshold τ. (The results reported in the figure are the average effective dimension $\pm$ one standard deviation across trials.)

tion with MLR optimization (*PPK-MLR*). For $|\mathcal{V}| = 1024, \tau = 1$, the mean AUC score improves from 0.680 ± 0.006 (Native) to 0.808 ± 0.005 (PPK-MLR). The effects of codebook size and quantization threshold are diminished by MLR optimization, although they are slightly more pronounced than in the previous case without PPK. We may then ask: does top-τ VQ provide any benefit?

Figure 5.5 lists the effective dimensionality — the number of principal components necessary to capture 95% of the training set's variance — of codeword histograms in PPK space as a function of quantization threshold τ. Although for the best-performing codebook size $|\mathcal{V}| = 1024$, each of $\tau \in \{ 1, 2, 4 \}$ achieves statistically equivalent performance, the effective dimensionality varies from 253.1 ± 6.0 ($\tau = 1$) to 106.6 ± 3.3 ($\tau = 4$). Thus, top-τ VQ can be applied to dramatically reduce the dimensionality of VQ representations, which in turn reduces the number of parameters learned by MLR, and therefore improves the efficiency of learning and retrieval, without significantly degrading performance.

Figure 5.6: A t-SNE visualization of the optimized similarity space produced by PPK+MLR on one training/test split of the data ($|\mathcal{V}| = 1024$, $\tau = 1$). Close-ups on three peripheral regions reveal *hip-hop* (upper-right), *metal* (lower-left), and *classical* (lower-right) genres.

Qualitative results

Figure 5.6 illustrates an example optimized similarity space produced by MLR on PPK histogram representations, as visualized in two dimensions by t-SNE (van der Maaten and Hinton, 2008). Even though the algorithm is never exposed to any explicit semantic information, the optimized space does exhibit regions which seem to capture intuitive notions of genre, such as *hip-hop*, *metal*, and *classical*.

Table 5.2 illustrates a few example queries and their top-5 closest results under the Euclidean and MLR-optimized metric. The native space seems to capture similarities due to energy and instrumentation, but does not necessarily match CF similarity. The optimized space captures aspects of the audio data which correspond to CF similarity, and produces playlists with more relevant results.

Comparison

Figure 5.4 lists the accuracy achieved by using TF-IDF weighting on codeword

Table 5.2: Example playlists generated by 5-nearest (training) neighbors of three different query (test) songs (left) using Euclidean distance on raw codeword histograms (center) and MLR-optimized PPK distances (right). Relevant results are indicated by ►.

Test query	VQ (Native)	VQ (PPK+MLR)
Ornette Coleman - Africa is the Mirror of All Colors	Judas Priest - You've Got Another Thing Comin'	Wynton Marsalis - Caravan
	Def Leppard - Rock of Ages	►Dizzy Gillespie - Dizzy's Blues
	KC & The Sunshine Band - Give it Up	►Michael Brecker - Two Blocks from the Edge
	Wynton Marsalis - Caravan	►Eric Dolphy - Miss Ann (live)
	Ringo Starr - It Don't Come Easy	Ramsey Lewis - Here Comes Santa Claus
Fats Waller - Winter Weather	►Dizzy Gillespie - She's Funny that Way	Chet Atkins - In the Mood
	Enrique Morente - Solea	►Charlie Parker - What Is This Thing Called Love?
	Chet Atkins - In the Mood	►Bud Powell - Oblivion
	Rachmaninov - Piano Concerto #4 in Gmin	►Bob Wills & His Texas Playboys - Lyla Lou
	Eluvium - Radio Ballet	►Bob Wills & His Texas Playboys - Sittin' On Top Of The World
The Ramones - Go Mental	Def Leppard - Promises	►The Buzzcocks - Harmony In My Head
	►The Buzzcocks - Harmony In My Head	Motley Crue - Same Ol' Situation
	Los Lonely Boys - Roses	►The Offspring - Gotta Get Away
	Wolfmother - Colossal	►The Misfits - Skulls
	Judas Priest - Diamonds and Rust (live)	►AC/DC - Who Made Who (live)

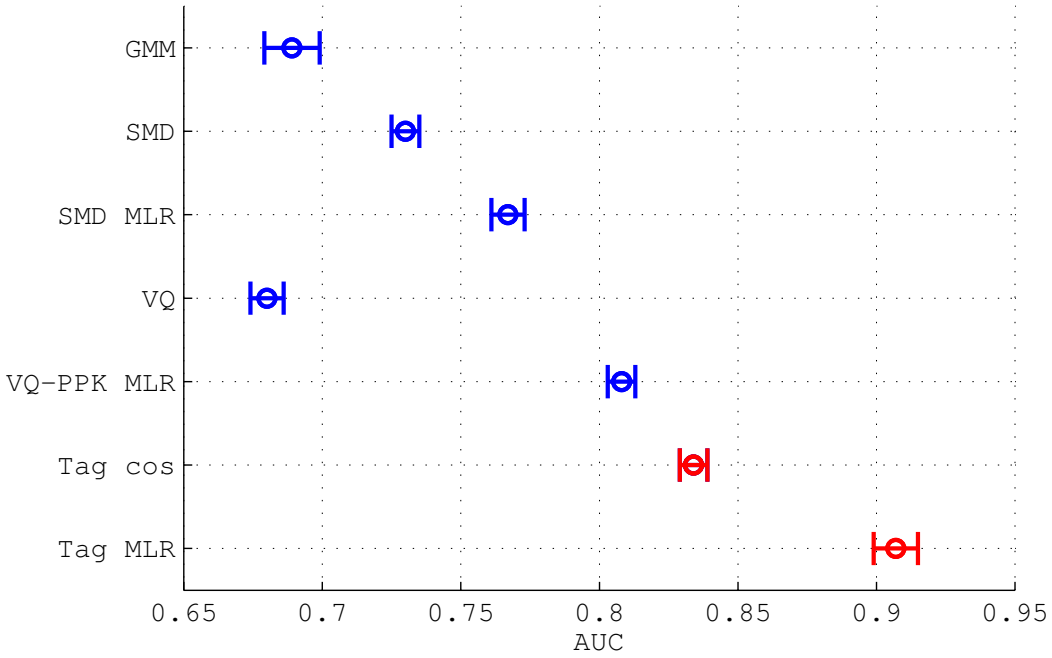


Figure 5.7: Comparison of VQ-based retrieval accuracy to competing methods. *VQ* corresponds to a codebook of size $V = 1024$ with quantization threshold $\tau = 1$. *Tag*-based methods (red) use human annotations, and are not automatically derived from audio content. Error bars correspond to one standard deviation across trials.

histograms. For all VQ configurations (*i.e.*, for each codebook size and quantization threshold) TF-IDF significantly degrades performance compared to MLR-based methods, which indicates that inverse document frequency may not be as an accurate predictor of salience in codeword histograms as in natural language (Salton and Buckley, 1987).

Figure 5.7 shows the performance of all other methods against which we compare. First, we observe that *raw* SMD representations provide more accurate retrieval than both the GMM approach and *raw* VQ codeword histograms (*i.e.*, prior to optimization by MLR). This may be expected, as previous studies have demonstrated superior query-by-example retrieval performance when using semantic representations of multimedia data (Rasiwasia et al., 2007, Barrington et al., 2007).

Moreover, SMD and VQ can be optimized by MLR to achieve significantly

higher performance than raw SMD and VQ, respectively. The semantic representations in SMD compress the original audio content to a small set of descriptive terms, at a higher level of abstraction. In raw form, this representation provides a more robust set of features, which improves recommendation performance compared to matching low-level content features that are often noisier. On the other hand, semantic representations are inherently limited by the choice of vocabulary and may prematurely discard important discriminative information (*e.g.*, subtle distinctions within sub-genres). This renders them less attractive as starting point for a metric learning algorithm like MLR, compared to less-compressed (but possibly noisier) representations, like VQ. Indeed, the latter may retain more information for MLR to learn an appropriate similarity function. This is confirmed by our experiments: MLR improves VQ significantly more than it does for SMD. As a result, MLR-VQ outperforms all other content-based methods in our experiments.

Finally, we provide an estimate of an upper bound on what can be achieved by automatic, content-based methods, by evaluating the retrieval performance when using manual annotations (*Tag* in fig. 5.7): 0.834 ± 0.005 with cosine similarity, and 0.907 ± 0.008 with MLR-optimized similarity. The improvement in accuracy for human tags, when using MLR, indicates that even hand-crafted annotations can be improved by learning an optimal distance over tag vectors. By contrast, TF-IDF on human tag vectors decreases performance to 0.771 ± 0.004, indicating that IDF does not accurately model (binary) tag salience. The gap in performance between content-based methods and manual annotations suggests that there is still room for improvement. Closing this gap may require incorporating more complex features to capture rhythmic and structural properties of music which are discarded by the simple timbral descriptors used here.

5.6 Conclusion

In this chapter, we have proposed a method for improving content-based audio similarity by learning from a sample of collaborative filter data. Collaborative filters form the basis of state-of-the-art recommendation systems, but cannot directly form recommendations or answer queries for items which have not yet been consumed or rated.

By optimizing content-based similarity from a collaborative filter, we provide a simple mechanism for alleviating the cold-start problem and extending music recommendation to novel or less known songs.

By using implicit feedback in the form of user listening history, we can efficiently collect high-quality training data without active user participation, and as a result, train on larger collections of music than would be practical with explicit feedback or survey data. Our notion of similarity derives from user activity in a bottom-up fashion, and obviates the need for coarse simplifications such as genre or artist agreement.

Our proposed top-τ VQ audio representation enables efficient and compact description of the acoustic content of music data. Combining this audio representation with an optimized distance metric yields similarity calculations which are both efficient to compute and substantially more accurate than competing content-based methods. The proposed metric learning framework is robust with respect to the choice of codebook size and VQ threshold τ, and yields stable performance over a broad range of VQ configurations.

While in this chapter, our focus remains on music recommendation applications, the proposed methods are quite general, and may apply to a wide variety of applications involving content-based similarity, such as nearest-neighbor classification of audio signals.

Acknowledgments

The contents of this chapter originally appeared in the following publication: B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning content similarity for music recommendation. *IEEE Transactions on Audio, Speech, and Language Processing*, 2012a. To appear.

Chapter 6

Large-scale similarity search

6.1 Introduction

Nearest neighbor computations lie at the heart of many content-based music information retrieval problems, such as playlist generation (Logan, 2004, Cai et al., 2007), classification and annotation (Slaney et al., 2008, Kim et al., 2009) and recommendation (chapter 5). Typically, each item (*e.g.*, song, clip, or artist) is represented as a point in some high-dimensional space, *e.g.*, $\mathbb{R}^d$ equipped with Euclidean distance or Gaussian mixture models equipped with Kullback-Leibler divergence.

For large music databases, nearest neighbor techniques face an obvious limitation: computing the distance from a query point to each element of the database becomes prohibitively expensive. However, for many tasks, *approximate* nearest neighbors may suffice. This observation has motivated the development of general-purpose data structures which exploit metric structure to locate neighbors of a query in sub-linear time (Bentley, 1975, Faloutsos and Lin, 1995, Datar et al., 2004).

In this chapter, we investigate the efficiency and accuracy of several modern variants of KD-trees (Bentley, 1975) for answering nearest neighbor queries for musical content. As we will demonstrate, these *spatial trees* are simple to construct, and can provide substantial improvements in retrieval time while maintaining satisfactory performance.

6.2 Related work

Content-based similarity search has received a considerable amount of attention in recent years, but due to the obvious data collection barriers, relatively little of it has focused on retrieval in large-scale collections.

Cai et al. (2007) developed an efficient query-by-example audio retrieval system by applying locality sensitive hashing (LSH) (Datar et al., 2004) to a vector space model of audio content. Although LSH provides strong theoretical guarantees on retrieval performance in sub-linear time, realizing those guarantees in practice can be challenging. Several parameters must be carefully tuned — the number of bins in each hash, the number of hashes, the ratio of *near* and *far* distances, and collision probabilities — and the resulting index structure can become quite large due to the multiple hashing of each data point. The implementation of Cai et al. (2007) scales to upwards of 10^5 audio clips, but since their focus was on playlist generation, they did not report the accuracy of nearest neighbor recall.

Schnitzer et al. (2009) developed a filter-and-refine system to quickly approximate the Kullback-Leibler (KL) divergence between timbre models. Each song was summarized by a multivariate Gaussian distribution over MFCC vectors, and mapped into a low-dimensional Euclidean vector space via the FastMap algorithm (Faloutsos and Lin, 1995), so that Euclidean distance approximates the symmetrized KL divergence between song models. To retrieve nearest neighbors for a query song, the approximate distances are computed from the query to each point in the database by a linear scan (the *filter* step). The closest points are then *refined* by computing the full KL divergence to the query. This approach exploits the fact that low-dimensional Euclidean distances are much cheaper to compute than KL-divergence, and depending on the size of the filter set, can produce highly accurate results. However, since the filter step computes distance to the entire database, it requires $\mathcal{O}(n)$ work, and performance may degrade if the database is too large to fit in memory.

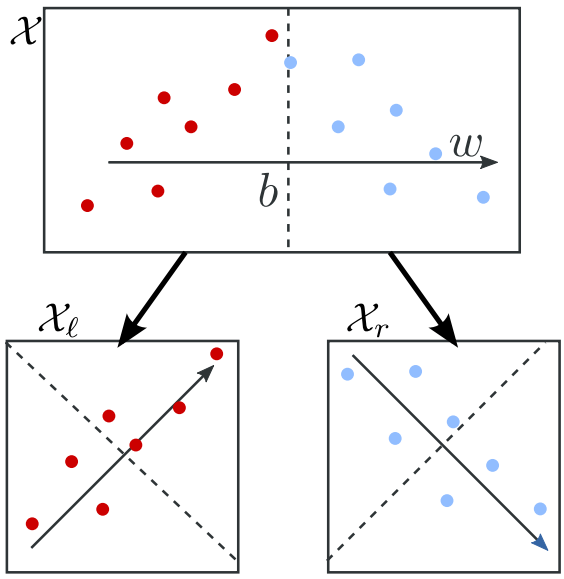


Figure 6.1: Spatial partition trees recursively split a data set $\mathcal{X} \subset \mathbb{R}^d$ by projecting onto a direction $w \in \mathbb{R}^d$ and splitting at the median b (dashed line), forming two disjoint subsets $\mathcal{X}_\ell$ and $\mathcal{X}_r$.

6.3 Spatial trees

Spatial trees are a family of data structures which recursively bisect a data set $\mathcal{X} \subset \mathbb{R}^d$ of n points in order to facilitate efficient (approximate) nearest neighbor retrieval (Bentley, 1975, Uhlmann, 1991). The recursive partitioning of $\mathcal{X}$ results in a binary tree, where each node t corresponds to a subset of the data $\mathcal{X}_t \subseteq \mathcal{X}$ (fig. 6.1). At the root of the tree lies the entire set $\mathcal{X}$, and each node t defines a subset of its parent.

A generic algorithm to construct partition trees is listed as algorithm 6.1. The set $\mathcal{X} \subset \mathbb{R}^d$ is projected onto a direction $w_t \in \mathbb{R}^d$, and split at the median b_t into subsets $\mathcal{X}_\ell$ and $\mathcal{X}_r$: splitting at the median ensures that the tree remains balanced. This process is then repeated recursively on each subset, until a specified tree depth δ is reached.

Spatial trees offer several appealing properties. They are simple to implement, and require minimal parameter-tuning: specifically, only the maximum tree depth δ, and the rule for generating split directions. Moreover, they are efficient to construct and use for retrieval. While originally developed for use in metric spaces, the framework has been recently extended to support general Bregman divergences (including, *e.g.*, KL-

Algorithm 6.1 Spatial partition tree

Input: data $\mathcal{X} \subset \mathbb{R}^d$, maximum tree depth δ

Output: balanced binary tree t over $\mathcal{X}$

PARTITION$(\mathcal{X}, \delta)$

1: **if** $\delta = 0$ **then**
2: **return** $\mathcal{X}$ (leaf set) // base case
3: **else**
4: $w_t \leftarrow \text{split}(\mathcal{X})$ // find a split direction
5: $b_t \leftarrow \text{median}\left(\left\{\, w_t^\mathsf{T} x \;\middle|\; x \in \mathcal{X} \,\right\}\right)$ // compute splitting threshold
6: $\mathcal{X}_\ell \leftarrow \left\{\, x \;\middle|\; w_t^\mathsf{T} x \leq b_t,\, x \in \mathcal{X} \,\right\}$ // partition $\mathcal{X}$
7: $\mathcal{X}_r \leftarrow \left\{\, x \;\middle|\; w_t^\mathsf{T} x > b_t,\, x \in \mathcal{X} \,\right\}$
8: $t_\ell \leftarrow$ PARTITION$(\mathcal{X}_\ell, \delta - 1)$ // recursively build sub-trees
9: $t_r \leftarrow$ PARTITION$(\mathcal{X}_r, \delta - 1)$
10: **return** $t := (w_t, b_t, t_\ell, t_r)$
11: **end if**

divergence) (Cayton, 2008). However, for the remainder of this chapter, we will focus on building trees for vector space models ($\mathbb{R}^d$ with Euclidean distance).

In order for algorithm 6.1 to be fully specified, we must provide a function $\text{split}(\mathcal{X})$ which determines the split direction w. Several splitting rules have been proposed in the literature, and our experiments will cover the four described by Verma et al. (2009): maximum variance KD, principal direction (PCA), 2-means, and random projection.

6.3.1 Maximum variance KD-tree

The standard KD-tree (k-dimensional tree) chooses w by cycling through the standard basis vectors e_i ($i \in \{\, 1, 2, \ldots, d \,\}$), so that at level j in the tree, the split direction is $w := \mathsf{e}_{i+1}$ with $i{:=}j \mod d$ (Bentley, 1975). The standard KD-tree can be effective for low-dimensional data, but it is known to perform poorly in high dimensions (Reiss et al., 2001, Verma et al., 2009). Note also that if $n < 2^d$, there will not be enough data to split along every coordinate, so some (possibly informative) features

may never be used by the data structure.

A common fix to this problem is to choose w as the coordinate which maximally spreads the data (Verma et al., 2009):

$$\mathrm{split}_{\mathrm{KD}}(\mathcal{X}) := \operatorname*{argmax}_{\mathsf{e}_i} \sum_{x \in \mathcal{X}} \left(\mathsf{e}_i^{\mathsf{T}} (x - \mu)\right)^2, \tag{6.1}$$

where μ is the sample mean vector of $\mathcal{X}$. Intuitively, this split rule picks the coordinate which provides the greatest reduction in variance (increase in concentration).

The maximum variance coordinate can be computed with a single pass over $\mathcal{X}$ by maintaining a running estimate of the mean vector and coordinate-wise variance, so the complexity of computing $\mathrm{split}_{\mathrm{KD}}(\mathcal{X})$ is $\mathcal{O}(dn)$.

6.3.2 PCA-tree

The KD split rule (eq. (6.1)) is limited to axis-parallel directions w. The principal direction (or principal components analysis, PCA) rule generalizes this to choose the direction $w \in \mathbb{R}^d$ which maximizes the variance, *i.e.*, the leading eigenvector v of the sample covariance matrix $\widehat{\Sigma}$:

$$\mathrm{split}_{\mathrm{PCA}}(\mathcal{X}) := \operatorname*{argmax}_{v} v^{\mathsf{T}} \widehat{\Sigma} v \quad \text{s. t. } \|v\|_2 = 1. \tag{6.2}$$

By using the full covariance matrix to choose the split direction, the PCA rule may be more effective than KD-tree at reducing the variance at each split in the tree.

$\widehat{\Sigma}$ can be estimated from a single pass over $\mathcal{X}$, so (assuming $n > d$) the time complexity of $\mathrm{split}_{\mathrm{PCA}}$ is $\mathcal{O}(d^2 n)$.

6.3.3 2-means

Unlike the KD and PCA rules, which try to maximally reduce variance with each split, the 2-means rule produces splits which attempt preserve cluster structure. This is accomplished by running the k-means algorithm on $\mathcal{X}$ with $k = 2$, and defining w to be the direction spanned by the cluster centroids $c_1, c_2 \in \mathbb{R}^d$:

$$\mathrm{split}_{\mathrm{2M}}(\mathcal{X}) := c_1 - c_2. \tag{6.3}$$

While this general strategy performs well in practice (Liu et al., 2005), it can be costly to compute a full k-means solution. In our experiments, we instead use an online k-means variant which runs in $\mathcal{O}(dn)$ time (appendix B).

6.3.4 Random projection

The final splitting rule we will consider is to simply take a direction uniformly at random from the unit sphere $\mathcal{S}^{d-1}$:

$$\text{split}_{\text{RP}}(\mathcal{X}) \sim_U \mathcal{S}^{d-1}, \tag{6.4}$$

which can equivalently be computed by normalizing a sample from the multivariate Gaussian distribution $\mathcal{N}(0, I_d)$. The random projection rule is simple to compute and adapts to the intrinsic dimensionality of the data $\mathcal{X}$ (Dasgupta and Freund, 2008).

In practice, the performance of random projection trees can be improved by independently sampling m directions $w_i \sim \mathcal{S}^{d-1}$, and returning the w_i which maximizes the decrease in data diameter after splitting (Verma et al., 2009). Since a full diameter computation would take $\mathcal{O}(dn^2)$ time, we instead return the direction which maximizes the *projected diameter*:

$$\underset{w_i}{\text{argmax}} \max_{x_1, x_2 \in \mathcal{X}} w_i^\mathsf{T} x_1 - w_i^\mathsf{T} x_2. \tag{6.5}$$

This can be computed in a single pass over $\mathcal{X}$ by tracking the maximum and minimum of $w_i^\mathsf{T} x$ in parallel for all w_i, so the time complexity of split_{RP} is $\mathcal{O}(mdn)$. Typically, $m \leq d$, so split_{RP} is comparable in complexity to $\text{split}_{\text{PCA}}$.

6.3.5 Spill trees

The main drawback of partition trees is that points near the decision boundary become isolated from their neighbors across the partition. Because data concentrates near the mean after (random) projection (Dasgupta and Freund, 2008), hard partitioning can have detrimental effects on nearest neighbor recall for a large percentage of queries.

Spill trees remedy this problem by allowing overlap between the left and right sub-trees (Liu et al., 2005). If a point lies close to the median, then it will be added

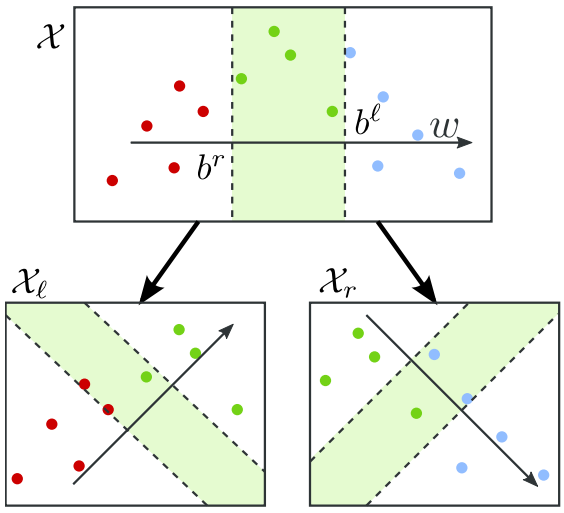


Figure 6.2: Spill trees recursively split data like partition trees, but the subsets are allowed to overlap. Points in the shaded region are propagated to both sub-trees.

to both sub-trees, thus reducing the chance that it becomes isolated from its neighbors (fig. 6.2). This is accomplished by maintaining two decision boundaries b_t^ℓ and b_t^r. If $w_t^\mathsf{T} x > b_t^r$, then x is added to the right tree, and if $w_t^\mathsf{T} x \leq b_t^\ell$, it is added to the left. The gap between b_t^ℓ and b_t^r controls the amount of data which *spills* across the split.

The algorithm to construct a spill tree is listed as algorithm 6.2. The algorithm requires a spill threshold $\tau \in [0, 1/2)$: rather than splitting at the median (so that a set of n items is split into subsets of size roughly $n/2$), the data is split at the $(1/2 + \tau)$-quantile, so that each subset has size roughly $n(1/2 + \tau)$. Note that when $\tau = 0$, the thresholds coincide ($b_t^\ell = b_t^r$), and the algorithm simplifies to algorithm 6.1. Partition trees, therefore, correspond to the special case of $\tau = 0$.

6.3.6 Retrieval algorithm and analysis

Once a spill tree has been constructed, approximate nearest neighbors can be recovered efficiently by the *defeatist search* method (Liu et al., 2005), which restricts the search to only the leaf sets which contain the query. For a novel query $q \in \mathbb{R}^d$ (*i.e.*, a previously unseen point), these sets can be found by algorithm 6.3.

The total time required to retrieve k neighbors for a novel query q can be computed as follows. First, note that for a spill tree with threshold τ, each split reduces the

Algorithm 6.2 Spill tree

Input: data $\mathcal{X} \subset \mathbb{R}^d$, depth δ, threshold $\tau \in [0, 1/2)$

Output: τ-spill tree t over $\mathcal{X}$

SPILL$(\mathcal{X}, \delta, \tau)$

if $\delta = 0$ **then**
  return $\mathcal{X}$ (leaf set) // base case
else
  $w_t \leftarrow \text{split}(\mathcal{X})$ // find a split direction
  $b_t^\ell \leftarrow \text{quantile}\left(1/2 + \tau, \left\{ w_t^\mathsf{T} x \;\middle|\; x \in \mathcal{X} \right\}\right)$ // compute thresholds
  $b_t^r \leftarrow \text{quantile}\left(1/2 - \tau, \left\{ w_t^\mathsf{T} x \;\middle|\; x \in \mathcal{X} \right\}\right)$
  $\mathcal{X}_\ell \leftarrow \left\{ x \;\middle|\; w_t^\mathsf{T} x \le b_t^\ell,\, x \in \mathcal{X} \right\}$ // split $\mathcal{X}$ with spilling
  $\mathcal{X}_r \leftarrow \left\{ x \;\middle|\; w_t^\mathsf{T} x > b_t^r,\, x \in \mathcal{X} \right\}$
  $t_\ell \leftarrow$ SPILL$(\mathcal{X}_\ell, \delta - 1, \tau)$ // recursively build sub-trees
  $t_r \leftarrow$ SPILL$(\mathcal{X}_r, \delta - 1, \tau)$
  return $t := (w_t, b_t^\ell, b_t^r, t_\ell, t_r)$
end if

size of the set by a factor of $(1/2 + \tau)$, so the leaf sets of a depth-δ tree are exponentially small in δ: $n(1/2 + \tau)^\delta$. Note that $\delta \le \log n$, and is typically chosen so that the leaf set size lies in some reasonable range (*e.g.*, between 100 and 1000 items).

Next, observe that in general, algorithm 6.3 may map the query q to some h distinct leaves, so the total size of the retrieval set is at most $n' = hn(1/2+\tau)^\delta$ (although it may be considerably smaller if the sets overlap). For h leaves, there are at most h paths of length δ to the root of the tree, and each step requires $\mathcal{O}(d)$ work to compute $w_t^\mathsf{T} q$, so the total time taken by algorithm 6.3 is

$$\mathcal{T}_{\text{RETRIEVE}} \in \mathcal{O}\big(h\left(d\delta + n(1/2 + \tau)^\delta\right)\big).$$

Finally, once the retrieval set has been constructed, the k closest points can be found in time $\mathcal{O}(dn' \log k)$ by using a k-bounded priority queue (Cormen et al., 2009). The total time to retrieve k approximate nearest neighbors for the query q is therefore

$$\mathcal{T}_{k\text{NN}} \in \mathcal{O}\big(hd\left(\delta + n(1/2 + \tau)^\delta \log k\right)\big).$$

Algorithm 6.3 Spill tree retrieval

Input: query q, tree t

Output: Retrieval set $\mathcal{X}_q$

RETRIEVE(q, t)

if t is a leaf **then**
  return $\mathcal{X}_t$ // all items contained in the leaf
else
  $\mathcal{X}_q \leftarrow \emptyset$
  if $w_t^\mathsf{T} q \leq b_t^\ell$ **then**
    $\mathcal{X}_q \leftarrow \mathcal{X}_q \cup$ RETRIEVE(q, t_ℓ) // does q belong to left sub-tree?
  end if
  if $w_t^\mathsf{T} q > b_t^r$ **then**
    $\mathcal{X}_q \leftarrow \mathcal{X}_q \cup$ RETRIEVE(q, t_r) // does q belong to right sub-tree?
  end if
  return $\mathcal{X}_q$
end if

Intuitively, for larger values of τ, more data is spread throughout the tree, so the leaf sets become larger and retrieval becomes slower. Similarly, larger values of τ will result in larger values of h as queries will map to more leaves. However, as we will show experimentally, this effect is generally mild even for relatively large values of τ.

In the special case of partition trees ($\tau = 0$), each query maps to exactly $h = 1$ leaf, so the retrieval time simplifies to $\mathcal{O}(d(\delta + n/2^\delta \log k))$.

6.4 Experiments

Our song data was taken from the Million Song Dataset (MSD) (Bertin-Mahieux et al., 2011). Before describing the tree evaluation experiments, we will briefly summarize the process of constructing the underlying acoustic feature representation.

6.4.1 Audio representation

The audio content representation was developed on the 1% Million Song Subset (MSS), and follows the basic pipeline described in chapter 5. From each MSS song, we extracted the time series of Echo Nest timbre descriptors (ENTs). This results in a sample of approximately 8.5 million 12-dimensional ENTs, which were normalized by z-scoring according to the estimated mean and variance of the sample, randomly permuted, and then clustered by online k-means (appendix B) to yield 512 acoustic codewords. Each song was summarized by quantizing each of its (normalized) ENTs and counting the frequency of each codeword, resulting in a 512-dimensional histogram vector. Each codeword histogram was mapped into a probability product kernel (PPK) space (Jebara et al., 2004) by square-rooting its entries, which has been demonstrated to be effective on similar audio representations (chapter 5). Finally, we appended the song's tempo, loudness, and key confidence, resulting in a vector $v_i \in \mathbb{R}^{515}$ for each song x_i.

Next, we trained an optimized similarity metric over audio descriptors. First, we computed target similarity for each pair of MSS artists by the Jaccard index between their user sets in a sample of Last.fm[1] collaborative filter data (Celma, 2010, chapter 3). Tracks by artists with fewer than 30 listeners were discarded. Next, all remaining artists were partitioned into a training (80%) and validation (20%) set, and for each artist, we computed its top 10 most similar training artists.

Having constructed a training and validation set, the distance metric was optimized by applying the metric learning to rank (MLR) algorithm (algorithm 3.1) on the training set of 4455 songs, and tuning parameters $C \in \{ 10^5, 10^6, \ldots, 10^9 \}$ and

$$\Delta \in \{ \text{AUC}, \text{MRR}, \text{MAP}, \text{Prec@10} \}$$

to maximize AUC score on the validation set of 1110 songs. Finally, the resulting metric W was factored by PCA (retaining 95% spectral mass) to yield a linear projection $L \in \mathbb{R}^{222 \times 515}$.

The projection matrix L was then applied to each v_i in MSD. As a result, each MSD song was mapped into $\mathbb{R}^{222}$ such that Euclidean distance is optimized by MLR to

[1] http://last.fm

retrieve songs by similar artists.

6.4.2 Representation evaluation

To verify that the optimized vector quantization (VQ) song representation carries musically relevant information, we performed a small-scale experiment to evaluate its predictive power for semantic annotation. We randomly selected one song from each of 4643 distinct artists. (Artists were restricted to be disjoint from MSS to avoid contamination.) Each song was represented by the optimized 222-dimensional VQ representation, and as ground truth annotations, we applied the corresponding artist's terms from the *top-300 terms* provided with MSD, so that each song x_i has a binary annotation vector $y_i \in \{ 0, 1 \}^{300}$. For a baseline comparison, we adapt the representation used by Schnitzer et al. (2009), and for each song, we fit a full-covariance Gaussian distribution over its ENT features.

The set was then randomly split 10 times into 80%-training and 20%-test sets. Following the procedure described by Kim et al. (2009), each test song was annotated by thresholding the average annotation vector of its k nearest training neighbors as determined by Euclidean distance on VQ representations, and by KL-divergence on Gaussians. Varying the decision threshold yields a trade-off between precision and recall. In our experiments, the threshold was varied between 0.1 and 0.9.

Figure 6.3 displays the precision-recall curves averaged across all 300 terms and training/test splits for several values of k. At small values of k, the VQ representation achieves significantly higher performance than the Gaussian representation. We note that this evaluation is by no means conclusive, and is merely meant to demonstrate that the underlying space is musically relevant.

6.4.3 Tree evaluation

To test the accuracy of the different spatial tree algorithms, we partitioned the MSD data into 890205 training songs $\mathcal{X}$ and 109795 test songs $\mathcal{X}'$. Using the optimized VQ representations on $\mathcal{X}$, we constructed trees with each of the four splitting rules (PCA, KD, 2-means, and random projection), varying both the maximum depth $\delta \in$

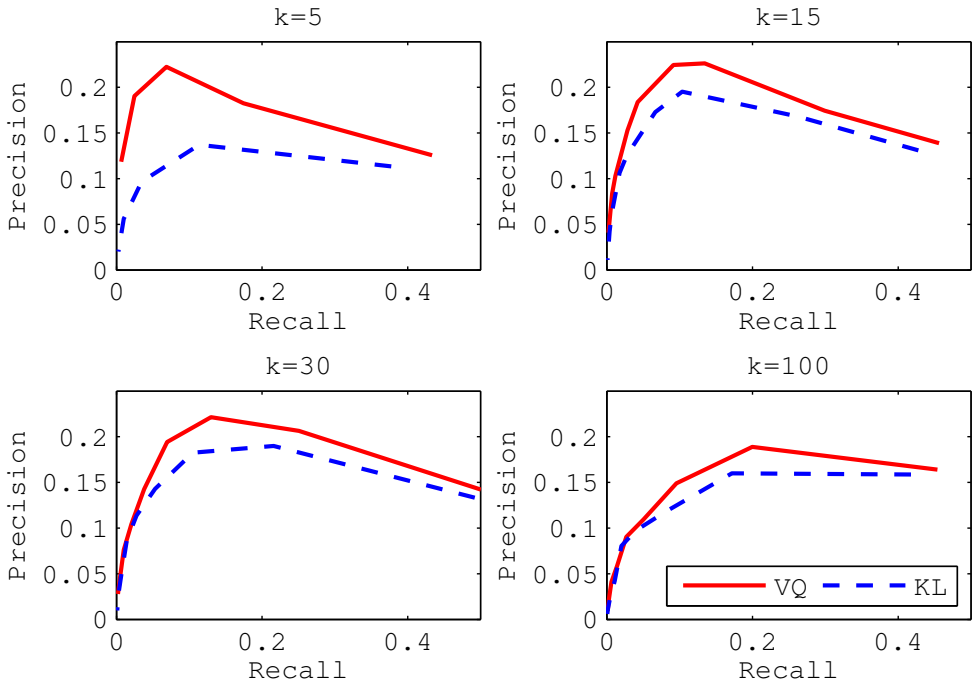


Figure 6.3: Mean precision-recall for k-nearest neighbor annotation with VQ and Gaussian (KL) representations.

$\{ 5, 6, \ldots, 13 \}$ and spill threshold $\tau \in \{ 0, 0.01, 0.05, 0.10 \}$. At $\delta = 13$, this results in leaf sets of size 109 with $\tau = 0$, and 1163 for $\tau = 0.10$. For random projection trees, we sample $m = 64$ dimensions at each call to split_{RP}.

For each test song $q \in \mathcal{X}'$, and tree t, we compute the retrieval set with algorithm 6.3. The *recall* for q is the fraction of the true nearest-neighbors $k\text{NN}(q)$ contained in the retrieval set:

$$R(q,t) := \frac{1}{k} \left| \textsc{Retrieve}(q,t) \cap k\text{NN}(q) \right|. \tag{6.6}$$

Note that since true nearest neighbors are always closer than any other points, they are always ranked first, so precision and recall are equivalent here.

To evaluate the system, $k = 100$ exact nearest neighbors $k\text{NN}(q)$ were found from $\mathcal{X}$ for each query $q \in \mathcal{X}'$ by a full linear search over $\mathcal{X}$.

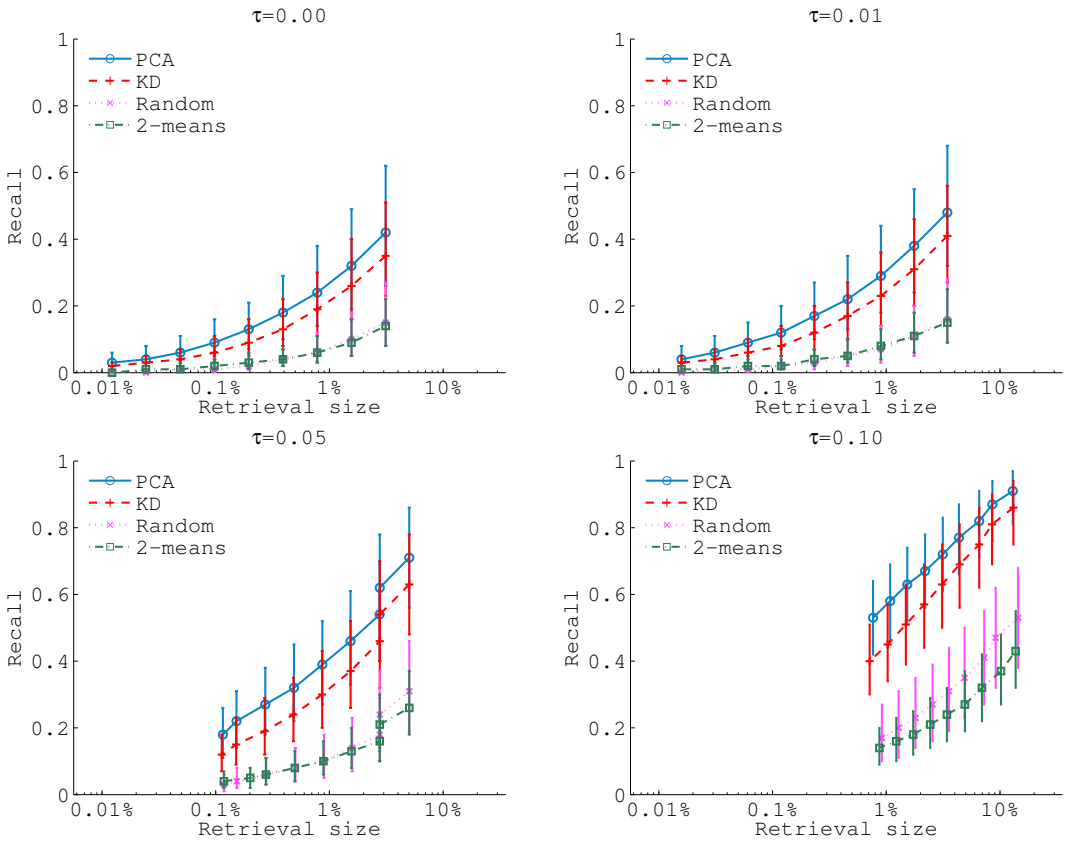


Figure 6.4: Median 100-nearest-neighbor recall for each splitting rule, spill threshold τ, and tree depth $\delta \in \{ 5, 6, \ldots, 13 \}$. Recall points correspond to different values of δ, and are plotted at the median size of the retrieval set. Error bars correspond to 25th and 75th percentiles of recall for all test queries.

6.4.4 Retrieval results

Figure 6.4 lists the nearest-neighbor recall performance for all tree configurations. As should be expected, for all splitting rules and spill thresholds, recall performance degrades as the maximum depth of the tree increases.

Across all spill thresholds τ and tree depths δ, the relative ordering of performance of the different split rules is essentially constant: $\text{split}_{\text{PCA}}$ performs slightly better than split_{KD}, and both dramatically outperform split_{RP} and split_{2M}. This indicates that for the feature representation under consideration here (optimized codeword histograms), variance reduction seems to be the most effective strategy for preserving

nearest neighbors in spatial trees.

For small values of τ, recall performance is generally poor for all split rules. However, as τ increases, recall performance increases across the board. The improvements are most dramatic for $\mathrm{split}_{\mathrm{PCA}}$. With $\tau = 0$, and $\delta = 7$, the PCA partition tree has leaf sets of size 6955 (0.8% of $\mathcal{X}$), and achieves median recall of 0.24. With $\tau = 0.10$ and $\delta = 13$, the PCA spill tree achieves median recall of 0.53 with a comparable median retrieval set size of 6819 (0.7% of $\mathcal{X}$): in short, recall is nearly doubled with no appreciable computational overhead. So, by looking at less than 1% of the database, the PCA spill tree is able to recover more than half of the 100 true nearest neighbors for novel test songs. This contrasts with the filter-and-refine approach (Schnitzer et al., 2009), which requires a full scan of the entire database.

6.4.5 Timing results

Finally, we evaluated the retrieval time necessary to answer k-nearest neighbor queries with spill trees. We assume that all songs have already been inserted into the tree, since this is the typical case for long-term usage. As a result, the retrieval algorithm can be accelerated by maintaining indices mapping songs to leaf sets (and vice versa).

We evaluated the retrieval time for PCA spill trees of depth $\delta = 13$ and threshold $\tau \in \{ 0.05, 0.10 \}$, since they exhibit practically useful retrieval accuracy. We randomly selected 1000 test songs and inserted them into the tree prior to evaluation. For each test song, we compute the time necessary to retrieve the k nearest training neighbors from the spill tree (ignoring test songs), for $k \in \{ 10, 50, 100 \}$. Finally, for comparison purposes, we measured the time to compute the true k nearest neighbors by a linear search over the entire training set.

Our implementation is written in Python/NumPy, [2] and loads the entire data set into memory. The test machine has two 1.6GHz Intel Xeon CPUs and 4GB of RAM. Timing results were collected through the *cProfile* utility.

Figure 6.5 lists the average retrieval time for each algorithm. The times are relatively constant with respect to k: a full linear scan typically takes approximately 2.4 seconds, while the $\tau = 0.10$ spill tree takes less than 0.14 seconds, and the $\tau =$

[2] http://numpy.scipy.org

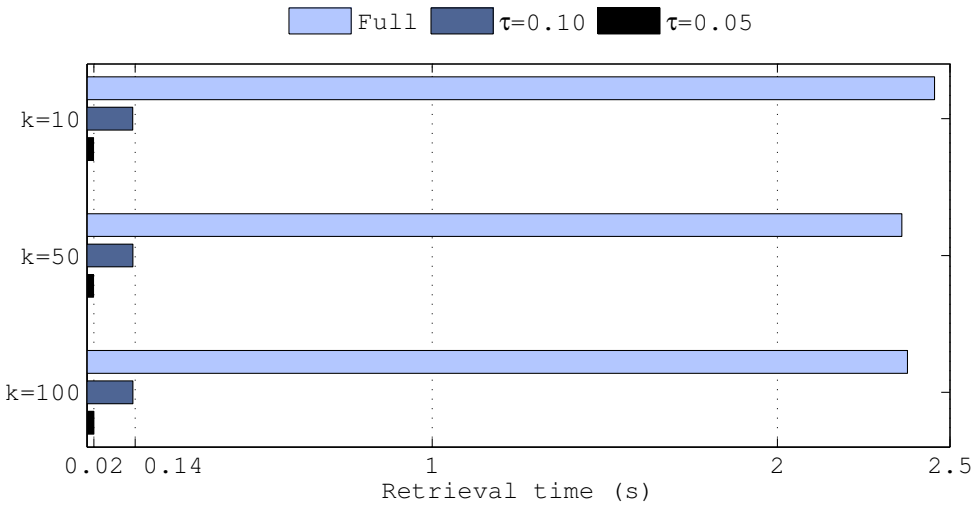


Figure 6.5: Average time to retrieve k (approximate) nearest neighbors with a full scan versus PCA spill trees.

0.05 tree takes less than 0.02 seconds. In relative terms, setting $\tau = 0.10$ yields a speedup factor of 17.8, and $\tau = 0.05$ yields a speedup of 119.5 over the full scan. The difference in speedup from $\tau = 0.10$ to $\tau = 0.05$ can be explained by the fact that smaller overlapping regions result in smaller (and fewer) leaf sets for each query. In practice, this speed-accuracy trade-off can be optimized for the particular task at hand: applications requiring only a few neighbors which may be consumed rapidly (*e.g.*, sequential playlist generation) may benefit from small values of τ, whereas applications requiring more neighbors (*e.g.*, browsing recommendations for discovery) may benefit from larger τ.

6.5 Conclusion

We have demonstrated that spatial trees can effectively accelerate approximate nearest neighbor retrieval. In particular, for VQ audio representations, the combination of spill trees with and PCA splits yields a favorable trade-off between accuracy and complexity of k-nearest neighbor retrieval.

Acknowledgments

The contents of this chapter originally appeared in the following publication: B. McFee and G.R.G. Lanckriet. Large-scale music similarity search with spatial trees. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 55–60, 2011a.

Chapter 7

Modeling playlists

7.1 Introduction

Music listeners typically do not listen to a single song in isolation. Rather, listening sessions tend to persist over a sequence of songs: a *playlist*. The increasing quantity of readily available, digital music content has motivated the development of algorithms and services to automate search, recommendation, and discovery in large music databases. However, playlist generation is fundamental to how users interact with music delivery services, and is generally distinct from related topics, such as similarity (chapter 5) and semantic search (Turnbull et al., 2008).

Although many automatic playlist generation algorithms have been proposed over the years, there is currently no standard evaluation procedure. As a result, it is difficult to quantitatively compare different algorithms and objectively gauge progress being made by the research community.

Previously, the predominant approach to playlist algorithm evaluation has been to conduct human opinion surveys, which can be expensive, time-consuming, and difficult to reproduce. Alternatively, current automated evaluation schemes either reduce the problem to a (discriminative) information retrieval setting, or rely on simplifying assumptions that may not hold in practice.

In this chapter, we propose a simple, scalable, and objective evaluation procedure for playlist algorithms that avoids the pitfalls of previous approaches. Our approach is guided by the observation that playlist generation is not (only) an information retrieval

problem, but a *language modeling* problem. Using this insight, we develop a general class of data-driven playlist models built upon hyper-graph random walks.

7.2 A brief history of playlist evaluation

Although many algorithms for playlist generation have been proposed, evaluation procedures have received relatively little specific attention. Here, we briefly summarize previously proposed evaluation strategies, which can broadly be grouped into three categories: *human evaluation*, *semantic cohesion*, and *sequence prediction*. This section is not intended as a comprehensive survey of playlist *algorithms*, for which we refer the interested reader to (Fields, 2011, chapter 2).

7.2.1 Human evaluation

Since the eventual goal of playlist algorithms is to improve user experience, the ideal method of algorithm evaluation is to directly measure human response. Numerous studies have been conducted in which test subjects rate the quality of playlists generated by one or more algorithms. Pauws and Eggen (2002) asked users to provide a query song with a particular context-of-use in mind (*e.g.*, *lively music*), which was used as a seed to generate a playlist. The user evaluated the resulting playlist on a scale of 1–10, and how many tracks in the playlist fit the user's intended use context. From these survey responses, the authors were able to derive various statistics to demonstrate that their proposed algorithm significantly outperforms randomly generated playlists. Similarly, Barrington et al. (2009) conducted experiments in which users were presented with two playlists (generated by obscured, competing systems) and asked to indicate which one was (subjectively) better, and why.

While direct human evaluation studies can provide evidence that one algorithm measurably outperforms another, they also have obvious practical limitations. They can be laborious, difficult to reproduce, and may require large numbers of test subjects and example playlists to achieve statistically meaningful results and overcome the effects of subjectivity.

7.2.2 Semantic cohesion

The general impracticality of large-scale user studies has motivated the development of automated evaluation techniques. The most common approaches compute some easily measurable quantity from each song in a generated playlist (*e.g.*, artist, album, or genre), which is used to determine the *cohesion* of the playlist. Cohesion may be defined by frequency counts of meta-data co-occurrence (*e.g.*, songs by the same artist) (Logan, 2002, 2004) or entropy of the distribution of genres within the playlist (Knees et al., 2006, Dopler et al., 2008). In this framework, it is typically assumed that each song can be mapped to a unique semantic tag (*e.g.*, *blues*). This assumption is often unrealistic, as songs generally map to multiple tags. Assigning each song to exactly one semantic description may therefore discard a great deal of information, and obscure the semantic content of the playlist. A more general form of semantic summarization was developed by Fields et al. (2010), and used to derive a distance measure between latent topic models of playlists. However, it is not immediately clear how such a distance metric would facilitate algorithm evaluation.

Issues of semantic ambiguity aside, a more fundamental flaw lies in the assumption that cohesion accurately characterizes playlist quality. In reality, this assumption is rarely justified, and evidence suggests that users often prefer highly diverse playlists (Slaney and White, 2006).

7.2.3 Sequence prediction

A more direct approach to automatic evaluation arises from formulating playlist generation as a prediction problem: given some contextual query (*e.g.*, a user's preferences, or a partial observation of songs in a playlist), the algorithm must predict which song to play next. The algorithm is then evaluated on the grounds of its prediction, under some notion of correctness. For example, Platt et al. (2002) observe a subset of songs in an existing playlist (the *query*), and the algorithm predicts a ranking of all songs. The quality of the algorithm is then determined by the position within the predicted ranking of the remaining, unobserved songs from the playlist. Maillet et al. (2009) similarly predict a ranking over songs from a contextual query — in this case, the preceding song

or pair of songs — and evaluate by comparing the ranking to one derived from a large collection of existing playlists.

Essentially, both of the above approaches transform playlist evaluation into an information retrieval (IR) problem: songs observed to co-occur with the query are *relevant*, and all other songs as *irrelevant*. As noted by Platt et al. (2002), this notion of relevance may be exceedingly pessimistic in practice due to sparsity of observations. Even in reasonably small music databases (say, a personal collection on the order of thousands of songs), the probability of observing any given pair of songs in a playlist becomes vanishingly small, and therefore, the overwhelming majority of song predictions are considered incorrect. In this framework, a prediction may disagree with observed co-occurrences, but still be equally pleasing to a user of the system, and therefore be unfairly penalized.

The IR approach — and more generally, any discriminative approach — is only applicable when one can obtain negative examples, *i.e.*, *bad* playlists. In reality, negative examples are difficult to define, let alone obtain, as users typically only share playlists that they like.[1] This suggests that discriminative evaluation may not be the most natural fit for playlist generation.

7.3 A natural language approach

In contrast to discriminative approaches to playlist evaluation, we advocate the *generative* perspective when modeling playlist composition. Rather than attempting to objectively score playlists as *good* or *bad*, which generally depends on user taste and unobservable contextual factors, we instead focus on modeling the distribution of naturally occurring playlists.

Formally, let $\mathcal{X} = \{ x_1, x_2, \ldots, x_n \}$ denote a library of songs. We define a *playlist* as an ordered finite sequence of elements of $\mathcal{X}$. Any procedure which constructs such ordered sequences is a *playlist algorithm* (or *playlister*). In general, we consider randomized algorithms, which can be used to generate multiple unique playlists from a

[1] A notable exception is the work of Bosteels et al. (2009), in which explicit negative feedback was inferred from skip behavior of Last.fm users. As noted by the authors, skip behavior can be notoriously difficult to interpret.

single query. Each playlister, be it randomized or deterministic, induces a probability distribution over song sequences, and may therefore be treated as a probabilistic generative model.

This leads to our central question: how should generative models of song sequences be evaluated? Here, we take inspiration from the literature of statistical natural language processing (Manning and Schütze, 1999), in which statistical models are fit to a sample of strings in the language (*e.g.*, grammatically valid sentences in English). A language model determines a probability distribution $\mathbf{P}$ over strings, which can be evaluated objectively by how well $\mathbf{P}$ matches the true distribution $\mathbf{P}_*$. Since $\mathbf{P}_*$ is unknown, this evaluation is approximated by drawing a sample $\mathcal{S} \sim \mathbf{P}_*$ of naturally occurring strings, and then computing the *likelihood* of the sample under the model $\mathbf{P}$.

Returning to the context of playlist generation, in place of vocabulary words, we have songs; rather than sentences, we have playlists. The universe of human-generated playlists therefore constitutes a *natural language*, and playlisters are models of the language of playlists. While this observation is not itself novel — it appears to be folklore among music researchers — its implications for algorithm evaluation have not yet been fully realized. We note that recent work by Zheleva et al. (2010) evaluated playlisters in terms of perplexity (exponentiated log-likelihood) of the genre distribution in a playlist, rather than the song selection itself.

7.3.1 Evaluation procedure

To evaluate a playlister A, we require the following:

1. a library of n songs $\mathcal{X}$,
2. a sample of playlists $\mathcal{S} \subseteq \mathcal{X}^\star$,[2] and
3. the likelihood $\mathbf{P}_A[s]$ of any playlist $s \in \mathcal{X}^\star$.

While the last requirement may seem like a tall order, we will demonstrate that for large classes of playlisters, the computation can be quite simple.

[2] $\mathcal{X}^\star$ denotes the Kleene-$\star$: all sequences of any length of elements drawn from $\mathcal{X}$.

A playlister A can be evaluated by computing the average log-likelihood of the sample $\mathcal{S}$:

$$\mathcal{L}(\mathcal{S}|\ A) := \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \log \mathbf{P}_A[s]. \tag{7.1}$$

The average log-likelihood, on an absolute scale, is not directly interpretable — although it's proportional to the cross-entropy between $\mathbf{P}_A$ and $\mathbf{P}_*$ — but it is useful for performing relative comparisons between two playlisters. Given a competing playlister A', we can say that A is a better model of the data than A' if $\mathcal{L}(\mathcal{S}|\ A) > \mathcal{L}(\mathcal{S}|\ A')$.

There is a subtle, but important distinction between the proposed approach and previous approaches to playlist evaluation. Rather than evaluate the perceived quality of a generated, *synthetic* playlist, we instead evaluate the *algorithm* in terms of how likely it is to produce *naturally occurring* playlists.

7.4 Playlist dialects

Numerous subtleties and difficulties arise when working with user-generated playlist data. For example, the data is often noisy, and the playlist creator's intent may be obscure. In extreme cases, users may compose playlists by randomly selecting songs from their libraries. More generally, different playlists may have different intended uses (*e.g.*, *road trip* or *party mix*), thematic elements (*break up* or *romantic*), or simply contain songs only of specific genres.

To better understand the structure of playlists, we advocate a more subtle approach. Rather than viewing naturally occurring playlists as a single language, we propose to model playlists as a collection of *dialects*, each of which may exhibit its own particular structure. Toward this end, we develop dialect-specific playlist models, and evaluate on a large corpus of annotated, user-generated playlists.

The proposed approach raises several natural questions:

- Is it beneficial to individually model playlist dialects?
- Are some dialects easier to model than others?
- Which features are important for each dialect?

Answering these questions will hopefully provide valuable insight into the underlying mechanics of playlist generation.

7.5 Hyper-graph random walks

Over the last decade, several researchers have proposed playlist generation algorithms based upon random walks (Logan, 2002, Ragno et al., 2005). A random walk playlist model consists of a weighted graph $G = (\mathcal{X}, E, w)$, where the vertices $\mathcal{X}$ represent the library of songs, and the edges E and weights w encode pairwise affinities between songs. A playlist is then generated by following a random trajectory through the graph, where transitions $x_t \rightsquigarrow x_{t+1}$ are sampled according to the weights on edges incident to x_t.

While random walk models are simple and often quite natural, they do carry certain limitations. It is often unclear how to define the weights, especially when multiple sources of pairwise affinity are available. Moreover, relying on pairwise interactions can severely limit the expressive power of these models (if each song has few neighbors), or scalability and precision (if each song has many neighbors), thus limiting their practical applicability.

To overcome these limitations, we propose a new class of playlist algorithms which allow for more flexible affinities between songs and sets of songs.

7.5.1 The user model

To motivate our playlist generation algorithm, we propose a simple model of user behavior. Rather than selecting songs directly from the entire collection $\mathcal{X}$, we assume that the user first narrows her selection to a subset $e \subseteq \mathcal{X}$ (*e.g.*, *blues* songs), from which the initial song is chosen uniformly at random. Similarly, for each subsequent transition $x_t \rightsquigarrow x_{t+1}$, the user selects a feature which describes the current song x_t, and then selects x_{t+1} uniformly from the subset of songs which also share that feature. The process repeats until the playlist is complete.

This user model is exactly characterized by a random walk on a *hyper-graph*. Hyper-graphs generalize undirected graphs by allowing an edge $e \in E$ to be an arbitrary

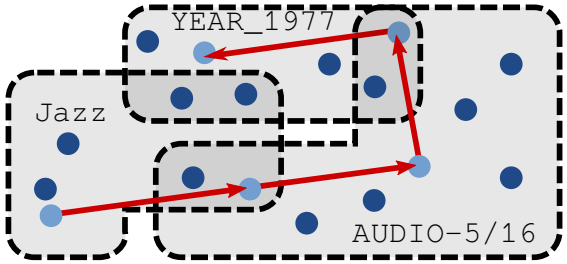


Figure 7.1: An example random walk on a song hypergraph: vertices represent songs, and edges are subsets of songs. Each transition $x_t \rightsquigarrow x_{t+1}$ must lie within an edge.

subset of the vertices, rather than a pair (fig. 7.1). For example, a hypergraph edge may be as general as *blues songs*, or as specific as *funk songs from 1977*. Edge weights can be used to encode the importance of a subset: for example, a model of *Blues* playlists would assign a high weight to an edge containing *blues* songs.

This model has several practically beneficial properties. First, it is efficient and scalable, in that the only information necessary to describe a song is its membership in the edge sets. Similarly, it naturally supports extension to new songs without having to significantly alter the model parameters (edge weights). Second, the model can easily integrate disparate feature sources, such as audio descriptors, lyrics, tags, *etc.*, as long as they can be encoded as subsets. Moreover, the model degrades gracefully if a song only has partial representation (*e.g.*, audio but no lyrics or tags). Finally, the model is *transparent*, in that each transition can be explained to the user simply in terms of the underlying edge taken between songs. As we will see in section 7.6, these edges often have natural semantic descriptions.

7.5.2 The playlist model

To formalize our model, let $H = (\mathcal{X}, E, w)$ denote a hypergraph over vertices (songs) $\mathcal{X}$, edges $E \subseteq 2^{\mathcal{X}}$, and non-negative weights $w \in \mathbb{R}_+^{|E|}$. We assume that the song library $\mathcal{X}$ and edge set E are given, and our goal is to optimize the edge weights w. We denote by $x_t^e := [\![x_t \in e]\!]$ the indicator that the song x_t is contained in the edge e.

Because the selection of the next song x_{t+1} depends only on the previous song x_t and edge weights w, the model is a first-order Markov process. The likelihood of

a playlist $s = (x_0 \rightsquigarrow x_1 \rightsquigarrow \cdots \rightsquigarrow x_T)$ thus factors into likelihood of the initial song, and each subsequent transition:

$$\mathbf{P}[x_0 \rightsquigarrow x_1 \rightsquigarrow \cdots \rightsquigarrow x_T |\, w] = \mathbf{P}[x_0 |\, w] \prod_{t=0}^{T-1} \mathbf{P}[x_{t+1} |\, x_t, w].$$

Given the edge weights w, the distribution over the initial song x_0 can be characterized by marginalizing over edges:

$$\mathbf{P}[x_0 |\, w] := \sum_{e \in E} \mathbf{P}[x_0 |\, e] \mathbf{P}[e |\, w] = \sum_{e \in E} \frac{x_t^e}{|e|} \frac{w_e}{\sum_{f \in E} w_f}.$$

Similarly, the probability of a transition $x_t \rightsquigarrow x_{t+1}$ is defined by marginalizing over edges incident to x_t:

$$\begin{aligned} \mathbf{P}[x_{t+1} |\, x_t, w] &:= \sum_{e \in E} \mathbf{P}[x_{t+1} |\, e, x_t] \cdot \mathbf{P}[e |\, x_t, w] \\ &= \sum_{e \in E} \frac{[\![x_{t+1} \neq x_t]\!] \cdot x_{t+1}^e}{|e| - 1} \cdot \frac{x_t^e w_e}{\sum_{f \in E} x_t^f w_f}. \end{aligned}$$

Finally, to promote sparsity among the edge weights and resolve scale-invariance in the model, we assume an IID exponential prior on edge weights w_e with rate $\lambda > 0$:

$$\mathbf{P}[w_e] := \lambda \cdot \exp\left(-\lambda w_e\right) \cdot [\![w_e \in \mathbb{R}_+]\!].$$

7.5.3 Learning the weights

Given a training sample of playlists $\mathcal{S} \subset \mathcal{X}^\star$, we would like to find the maximum a posteriori (MAP) estimate of w:

$$\overline{w} \leftarrow \operatorname*{argmax}_{w \in \mathbb{R}_+^{|E|}} \log \mathbf{P}[w |\, \mathcal{S}] = \operatorname*{argmax}_{w \in \mathbb{R}_+^{|E|}} \sum_{s \in \mathcal{S}} \log \mathbf{P}[s |\, w] + \sum_{e \in E} \log \mathbf{P}[w_e]. \tag{7.2}$$

The MAP objective (eq. (7.2)) is not concave, and it is generally difficult to find a global optimum. Our implementation uses the L-BFGS-B algorithm (Byrd et al., 1995) to solve for $\overline{w}$, and converges quite rapidly to a stationary point. Training typically takes a matter of seconds, even for the large playlist collections and edge sets described in section 7.6.

7.6 Data collection

Previous work on playlist modeling used the Art of the Mix[3] (AotM) collection of Ellis et al. (2002). The existing AotM dataset was collected in 2002, and consists of roughly 29K playlists over 218K songs, provided as lists of plain-text song and artist names. In this work, we expand and enrich this dataset into a new collection, which we denote as AotM-2011. This section describes our data collection, pre-processing, and feature extraction methodology.

7.6.1 Playlists: Art of the Mix 2011

To expand the AotM playlist collection, we crawled the site for all playlists, starting from the first indexed playlist (1998-01-22) up to the most recent at the time of collection (2011-06-17), resulting in 101343 unique playlists. Each playlist contains not only track and artist names, but a timestamp and categorical label (*e.g.*, *Road Trip* or *Reggae*).

In order to effectively model the playlist data, the plain-text song and artist names must be resolved into a common namespace. We use the Million Song Dataset (MSD) as the underlying database (Bertin-Mahieux et al., 2011). Rather than rely on the Echo Nest text-search API to resolve song identifiers, we instead implemented a full-text index of MSD song and artist names in Python with the Whoosh[4] library. This allowed both high throughput and fine-grained control over accent-folding and spelling correction. Each (*artist, song*) pair in the raw playlist data was used as a query to the index, and resolved to the corresponding MSD song identifier (if one was found). In total, 98359 songs were matched to unique identifiers.

Because not every song in a playlist could be correctly resolved, each playlist was broken into contiguous segments of two or more matched song identifiers. Finally, playlist segments were grouped according to category. Table 7.1 lists each of the 25 most popular categories by size.

[3] http://www.artofthemix.org
[4] http://packages.python.org/Whoosh/

7.6.2 Edge features

To fully specify the playlist model, we must define the edges of the hypergraph. Because edges can be arbitrary subsets of songs, the model is able to seamlessly integrate disparate feature modalities. We use the following collection of edge features, which can be derived from MSD and its add-ons.

Audio To encode low-level acoustic similarity, we first mapped each song i to a vector $x_i \in \mathbb{R}^{222}$ using the optimized vector quantized Echo Nest Timbre (ENT) descriptors constructed in chapter 6. Audio descriptors were clustered via online k-means, and cluster assignments were used to produce k disjoint subsets. This process was repeated for $k \in \{16, 64, 256\}$ to produce multiple overlapping edges of varying degrees of granularity. All 98K songs receive audio representations.

Collaborative filter To capture high-level similarities due to user listening patterns, we construct edges from the taste profile data used in the MSD Challenge (McFee et al., 2012b). We used the Bayesian Personalized Ranking (BPR) algorithm (Rendle et al., 2009, Gantner et al., 2011) to factor the users-by-songs (1M-by-380K) feedback matrix into latent feature vectors $x_i \in \mathbb{R}^{32}$. The BPR regularization parameters were set to $\lambda_1 = \lambda_2 = 10^{-4}$. Edges were constructed by cluster assignments following the procedure described above for audio features. 62272 songs (63%) coincide with the taste profile data.

Era The era in which songs are released can play an important role in playlist composition (Cunningham et al., 2006, Lee, 2011). To model this, we use the MSD metadata to represent each song by its year and half-overlapping decades. For example, the song *Parliament - Flash Light* maps to edges `YEAR-1977`, `DECADE-1970` and `DECADE-1975`. 77884 songs (79%) were mapped to era descriptors.

Familiarity Previous studies have noted the importance of song- or artist-familiarity in playlist composition (Cunningham et al., 2006). We used the artist familiarity data provided with MSD, which maps each song to the range $[0, 1]$ (0 being unfamiliar, 1 being very familiar). Edges were constructed by estimating the 25th and 75th

percentiles of familiarity, and mapping each song to `LOW`, `MEDIUM`, or `HIGH` familiarity.

Lyrics Previous studies have shown the importance of lyrics in playlist composition (Lee, 2011). To compute lyrical similarity, we applied online latent Dirichlet allocation (LDA) (Hoffman et al., 2010) with $k = 32$ to the musiXmatch lyrics database.[5] We then constructed three sets of 32 edges (one edge per topic): the first matches each song to its most probable topic, the second matches each song to its top three topics, and the third set to its top five topics. 53351 songs (56%) were found in the musiXmatch data.

Social tags Using the Last.fm[6] tags for MSD, we matched each song to its ten most frequent tags. Each tag induces an edge, comprised of the songs assigned to that tag.[7] 80396 songs (82%) matched to tag edges.

Uniform shuffle Because the features described above cannot model all possible transitions, we include the `uniform` edge that contains the entire set of songs. A transition through the uniform edge can be interpreted as a random restart of the playlist. The uniform shuffle also provides a standard baseline for comparison purposes.

Feature conjunctions Some of the features described above may be quite weak individually, but when combined, may become highly descriptive. For example, the tag `rock` and era `YEAR-1955` are both quite vague, but the conjunction of these two descriptors — `rock-&-YEAR-1955` — retains semantic interpretability, and is much more precise. We therefore augment the above collection of edges with all pair-wise intersections of features. Note that this induces general cross-modal feature conjunctions, such as `Lyrics topic #4-&-Audio cluster #17`, resulting in an extremely rich set of song descriptors.

[5] http://labrosa.ee.columbia.edu/millionsong/musixmatch
[6] http://last.fm/
[7] A similar tag-hypergraph model was proposed by Wang et al. (2009).

7.7 Experiments

To evaluate the proposed method, we randomly partitioned each of the top-25 categories listed in table 7.1 into ten 75/25 train/test splits. For each split, the train (test) sets are collected across categories to form a global train (test) set *ALL*, which is used to train a global model. After fitting a model to each training set, we compute the average (length-normalized) log-likelihood of the test set $\mathcal{S}'$:

$$\mathcal{L}(\mathcal{S}'|\, w) := \frac{1}{|\mathcal{S}'|} \sum_{s \in \mathcal{S}'} \frac{1}{|s|} \log \mathbf{P}[s|\, w].$$

For comparison purposes, we report performance in terms of the relative gain over the uniform shuffle model w_{u} (all weight assigned to the uniform edge):

$$G(w) := 1 - \frac{\mathcal{L}(\mathcal{S}'|\, w)}{\mathcal{L}(\mathcal{S}'|\, w_{\mathsf{u}})}.$$

To simplify the model and reduce over-fitting effects, we pruned all edges containing fewer than 384 (98359/256) songs. Similarly, we pruned redundant conjunction edges that overlapped by more than 50% with either of their constituent edges. Table 7.2 lists the number of edges retained after pruning. On average, each song maps to 76.46 ± 57.79 edges, with a maximum of 218. In all experiments, we fix the prior parameter $\lambda = 1$.

7.7.1 Experiment 1: Does dialect matter?

In our first set of experiments, we compare the global model to category-specific models. Figures 7.2 and 7.3 illustrate the relative gain over uniform across all categories for four different model configurations: tags, tags with pairwise conjunctions, all features, and all features with conjunctions.

Several interesting trends can be observed from these figures. First, in all but two cases — *Narrative* and *Rock* under the all features with conjunctions model — the category-specific models perform at least as well as the global model, often substantially so. As should be expected, the effect is most pronounced for genre-specific categories that naturally align with semantic tags (*e.g.*, *Hip Hop* or *Punk*).

Note that the larger categories overlap more with *ALL*, leaving less room for improvement over the global model. Not surprisingly, the *Mixed* category appears to be

difficult to model with similarity-based features. Similarly, several other categories are quite broad (*Theme*, *Narrative*, *Rock*), or may be inherently difficult (*Alternating DJ*, *Mixed*).

Also of note are the differences across model configurations. In most cases, feature conjunctions provide a modest improvement, both in the global and category-specific models. Due to the large parameter space, some over-fitting effects can be observed in the smallest categories (*Folk*, *Reggae*, *Blues*). Interestingly, several categories benefit substantially from the inclusion of all features compared to only tags (*e.g.*, *Hip Hop*, *Punk*, *Jazz*).

7.7.2 Experiment 2: Do transitions matter?

Given the flexibility of the model, it is natural to question the importance of modeling playlist continuity: could a model which ignores transition effects perform as well as the random walk model? To test this, we split each playlist $s = (x_0 \rightsquigarrow x_1 \rightsquigarrow \cdots \rightsquigarrow x_T)$ into singletons $s_0 = (x_0)$, $\cdots$, $s_T = (x_T)$. With the modified corpus, the model treats each song in a playlist as an independent draw from the initial distribution $\mathbf{P}[x_0 |\ w]$. Consequently, a model trained on this corpus can fit global trends across playlists within a category, but cannot enforce local continuity.

Figure 7.4 illustrates the relative gain for each category under the stationary distribution with all features and conjunctions. The results are qualitatively similar for alternate model configurations. Compared to fig. 7.3 (bottom), the results are substantially worse for most categories. In many cases, the stationary model performs worse than the uniform shuffle. This reflects the importance of transition effects when modeling playlists, even when the corpus is confined to genre-specific categories.

7.7.3 Experiment 3: Which features matter?

As illustrated in fig. 7.3, certain categories seem to benefit substantially from the inclusion of non-tag features. To investigate this effect, fig. 7.5 illustrates the aggregated weight for each feature type under each of the category models. Note that weight is aggregated across feature conjunctions, so the weight for edge `DECADE_1955-&-Rock`

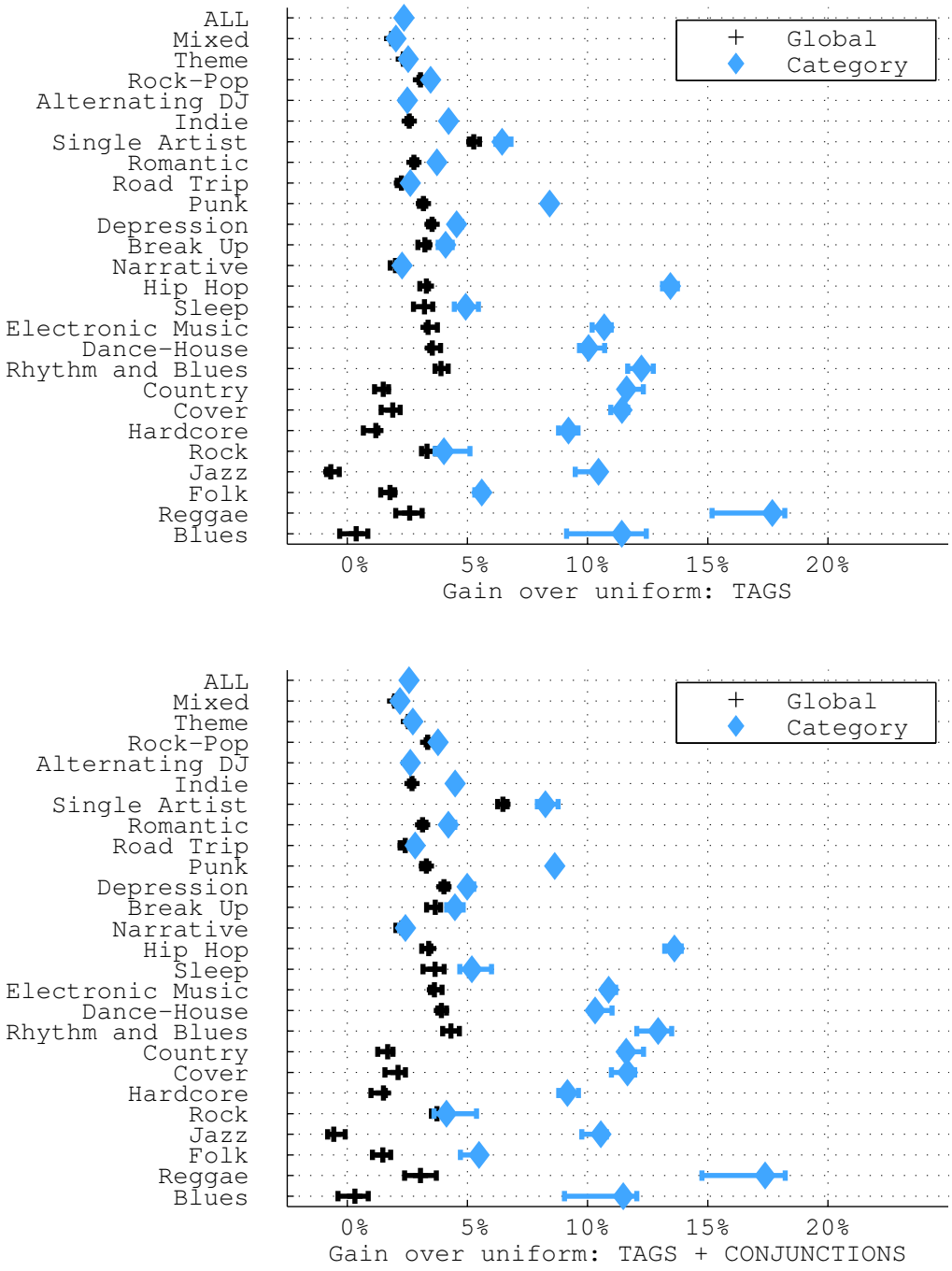


Figure 7.2: The median gain in log-likelihood over the baseline model using tags and conjunctions of tags, aggregated over ten random splits of the data. Error bars span the 0.25–0.75 quantiles. Category-specific models generally outperform global models.

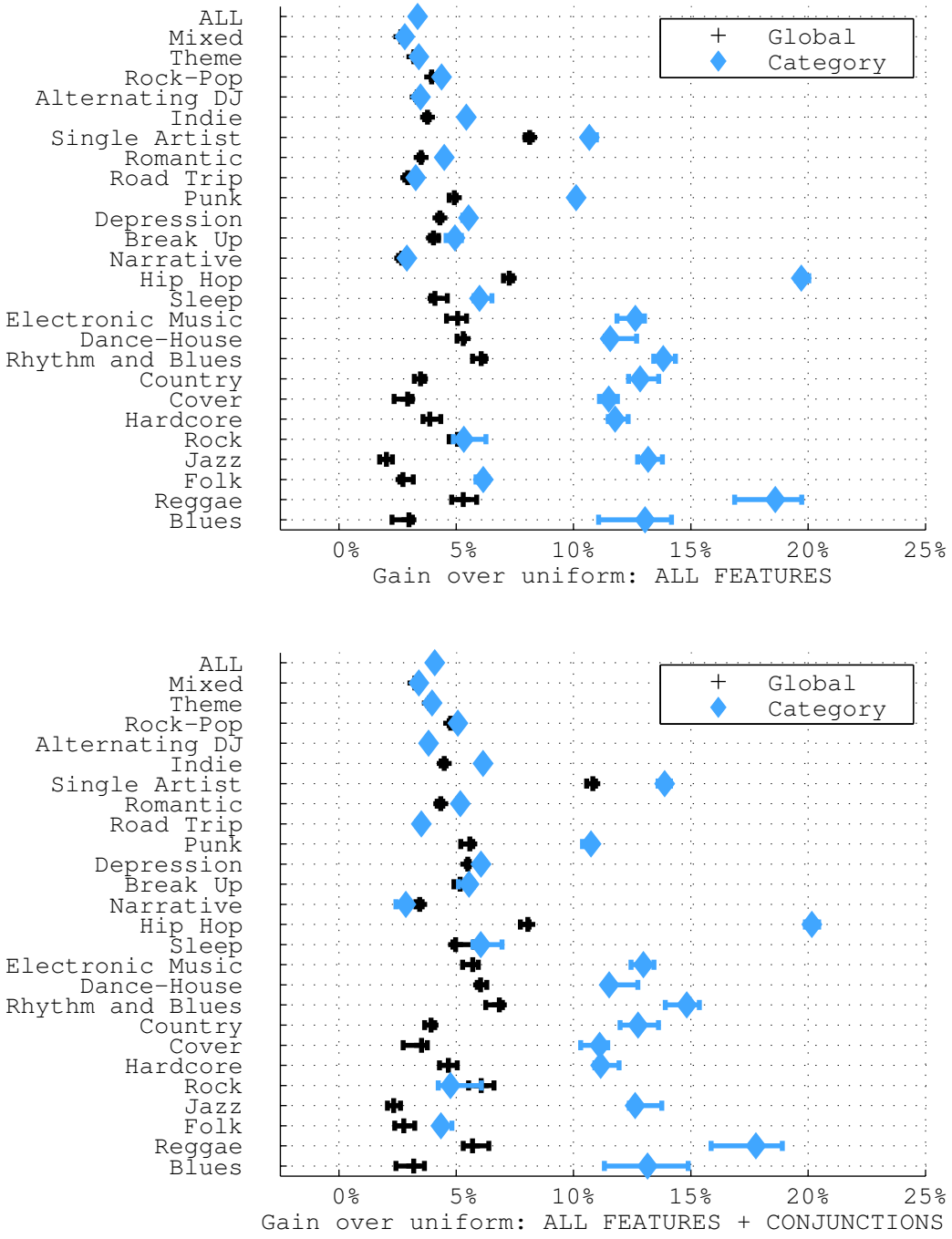


Figure 7.3: The median gain in log-likelihood over the baseline model using all features.

counts both for *Era* and *Tag*.

Tags receive the majority of edge weight (64% on average) across all categories.

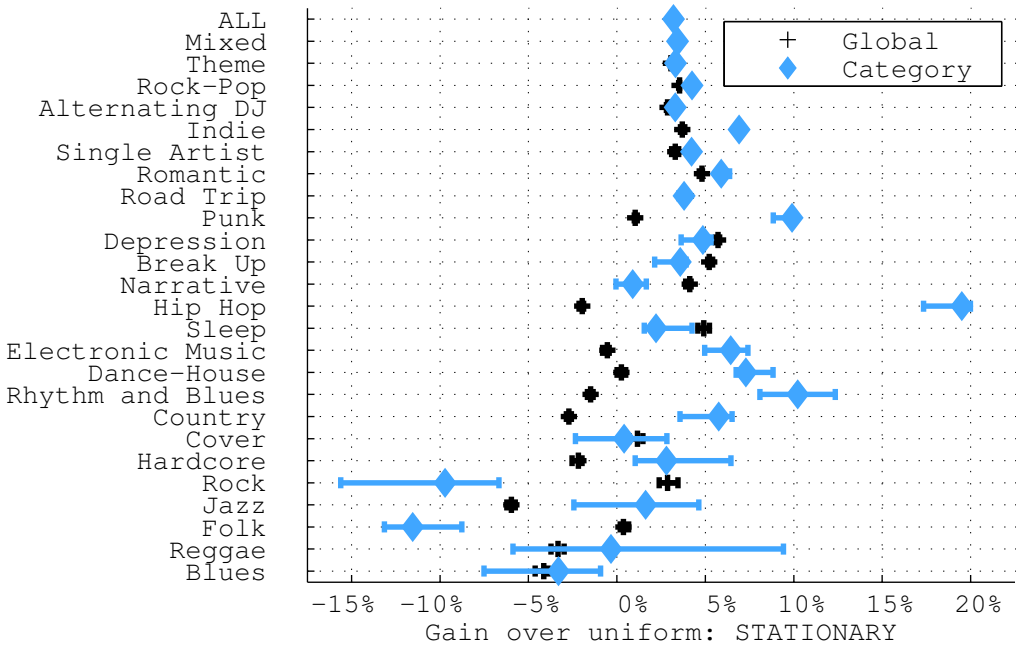


Figure 7.4: Log-likelihood gain over uniform with the stationary model (all features and conjunctions). Ignoring temporal structure significantly degrades performance.

Audio features appear to be most useful in *Hip Hop*, *Jazz* and *Blues* (43%–44%, compared to 26% average). This is not surprising, given that these styles feature relatively distinctive instrumentation and production qualities. Lyrical features receive the most weight in categories with prominent or salient lyrical content (*Folk*, *Cover*, *Narrative*, *Hardcore*, *Break Up*) and low weight in categories with little or highly variable lyrical content (*Electronic Music*, *Dance-House*, *Jazz*). Era and familiarity receive moderate weight (on average, 22% and 15% respectively), but the majority (20% and 14%) is due to conjunctions.

7.7.4 Example playlists

Table 7.3 illustrates a few example playlists generated by the category-specific feature conjunction model. For generative purposes, the uniform edge was removed after training. The generated playlists demonstrate both consistency within a single playlist and variety across playlists. Each transition in the playlist is explained by the

corresponding (incoming) edge, which provides transparency to the user: for example, *Cole Porter - You're the Top* follows *Django Rheinhardt - Brazil* because both songs belong to the conjunction edge `AUDIO-3/16-&-jazz`, and share both high- and low-level similarity.

7.8 Conclusion

We have demonstrated that playlist model performance can be improved by treating specific categories of playlists individually. While the simple models proposed here work well in some situations, they are far from complete, and suggest many directions for future work. The first-order Markov assumption is clearly a simplification, given that users often create playlists with long-term interactions and global thematic properties. Similarly, the uniform distribution over songs within an edge set allows for an efficient and scalable implementation, but allowing non-uniform distributions could also be an avenue for future improvement.

Acknowledgments

The authors thank Ben Fields and Matt Hoffman for many helpful discussions. The contents of this chapter, in part, originally appeared in the following publication: B. McFee and G.R.G. Lanckriet. The natural language of playlists. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 537–541, 2011b.

Table 7.1: The distribution of the top 25 playlist categories in AotM-2011. Each playlist consists of one or more segments of at least two contiguous MSD songs. 948 songs do not appear within the top 25 categories, but are included in the model.

Category	Playlists	Segments	Songs
Mixed	41798	101163	64766
Theme	12813	31609	35862
Rock-Pop	4935	13661	20364
Alternating DJ	4334	10493	18083
Indie	4528	10333	13678
Single Artist	3717	9044	17715
Romantic	2523	6269	8873
Road Trip	1846	4817	8935
Punk	1167	3139	4936
Depression	1128	2625	4794
Break Up	1031	2512	4692
Narrative	964	2328	5475
Hip Hop	1070	1958	2505
Sleep	675	1487	2957
Electronic Music	611	1131	2290
Dance-House	526	1117	2375
Rhythm and Blues	432	1109	2255
Country	398	908	1756
Cover	447	833	1384
Hardcore	268	633	1602
Rock	215	565	1866
Jazz	295	512	1089
Folk	241	463	1137
Reggae	183	403	831
Blues	165	373	892
Top-25	86310	209485	97411

Table 7.2: Summary of edges after pruning.

Feature	# Edges
Audio	204
Era	56
Lyrics	82
Uniform	1
Collaborative filter	93
Familiarity	3
Tags	201
All features	640
Feature conjunctions	6390

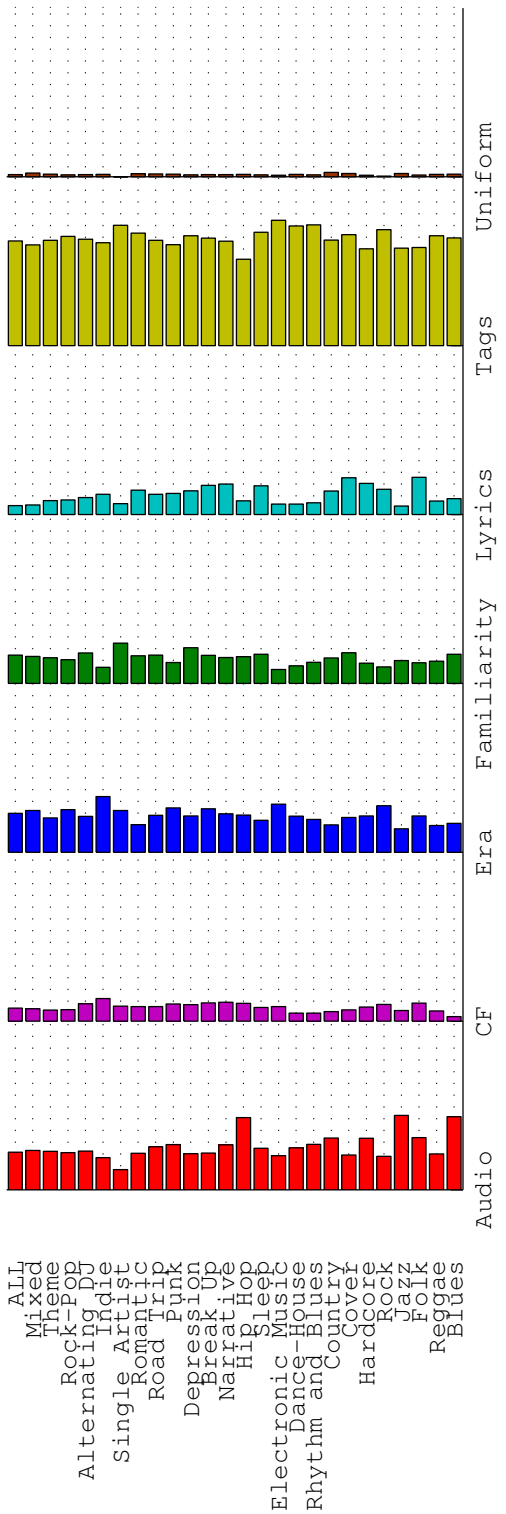


Figure 7.5: Distribution of learned edge weights for each playlist category. Weight is aggregated across feature conjunctions.

Table 7.3: Example playlists generated by various dialect models. The *incoming edge* is shared by the previous song (if one exists), and is used to select the current song.

Incoming edge	Playlist
Hip Hop	
AUDIO-149/256	*Eminem - The Conspiracy (Freestyle)*
AUDIO-149/256	*Busta Rhymes - Bounce*
DECADE-2000-&-rap	*Lil' Kim (feat. Sisqo) - How Many Licks?*
old school	*A Tribe Called Quest - Butter*
DECADE_1985-&-Hip-Hop	*Beastie Boys - Get It Together*
AUDIO-12/16	*Big Daddy Kane - Raw [Edit]*
Electronic Music	
AUDIO-11/16-&-downtempo	*Everything But The Girl - Blame*
DECADE_1990-&-trip-hop	*Massive Attack - Spying Glass*
AUDIO-11/16-&-electronica	*Björk - Hunter*
DECADE_2000-&-AUDIO-23/64	*Four Tet - First Thing*
electronica-&-experimental	*Squarepusher - Port Rhombus*
electronica-&-experimental	*The Chemical Brothers - Left Right*
Rhythm and Blues	
70s-&-soul	*Lyn Collins - Think*
AUDIO-14/16-&-funk	*Isaac Hayes - No Name Bar*
DECADE_1965-&-soul	*Michael Jackson - My Girl*
AUDIO-6/16-&-soul	*The Platters - Red Sails In The Sunset*
FAMILIARITY_MED-&-60s	*The Impressions - People Get Ready*
soul-&-oldies	*James & Bobby Purify - I'm Your Puppet*
Jazz	
AUDIO-14/16-&-jazz	*Peter Cincotti - St Louis Blues*
jazz	*Tony Bennett - The Very Thought Of You*
vocal jazz	*Louis Prima - Pennies From Heaven*
jazz-&-instrumental	*Django Reinhardt - Brazil*
AUDIO-3/16-&-jazz	*Cole Porter - You're The Top*
jazz	*Doris Day - My Blue Heaven*

Appendix A

Optimizing Average Precision

A.1 Cutting plane optimization of AP

Recall from eq. (3.6) the definition of a ranking's *average precision* (AP). Average precision is a commonly used performance metric in information retrieval, due to its ability to emphasize the early positions of a ranking without relying on a hard threshold (like in precision-at-k) (Baeza-Yates and Ribeiro-Neto, 1999). Yue et al. (2007) derived a greedy cutting plane algorithm to optimize AP loss within the structural SVM framework. In this appendix, we derive an alternative algorithm based on dynamic programming, which exploits the recursive substructure of AP and the partial order feature ψ_{po}.

To implement this, we must describe an efficient algorithm to find the *most violated constraint* for each q, that is, the ranking y which has simultaneously large discriminant value $\langle w, \psi(q, y)\rangle$ and large loss $\Delta(y_q, y)$. Explicitly, for each q, we seek

$$\begin{aligned}
\overline{y} &\leftarrow \operatorname*{argmax}_y \langle w, \psi(q,y) - \psi(q,y_q)\rangle + \Delta(y_q, y) && \text{(A.1)}\\
&= \operatorname*{argmax}_y \langle w, \psi(q,y)\rangle + \Delta(y_q, y)\\
&= \operatorname*{argmax}_y \langle w, \psi(q,y)\rangle + \text{AP}(q, y_q) - \text{AP}(q, y)\\
&= \operatorname*{argmax}_y \langle w, \psi(q,y)\rangle - \text{AP}(q, y). && \text{(A.2)}
\end{aligned}$$

In the next section, we derive a dynamic programming algorithm to solve for the most

violated constraint.

A.2 Most violated constraint DP

The first thing to note, as observed by Yue et al. (2007), is that AP is invariant to permutations of relevant (and irrelevant) documents, but the discriminant score is not. Therefore, if the positions of relevant documents are fixed within a ranking, eq. (A.2) is optmized by sorting the relevant (and irrelevant) documents in descending order according to to the point-wise discriminant score $\langle w, \phi(q, x_i)\rangle$. Given this observation, to solve eq. (A.2), it suffices to first sort $\mathcal{X}_q^+$ and $\mathcal{X}_q^-$ by the discriminant score, and then solve for an optimal interleaving of the two ordered sets.

Our technique to construct the optimal interleaving exploits two additional observations:

Proposition A.1. *Let y be a partial ranking of the first $a-1$ relevant documents $\mathcal{X}_q^+$ and all irrelevant documents $\mathcal{X}_q^-$. Creating y' by inserting the a^{th} relevant document at position b (where b is strictly greater than the index of the last relevant document in y, see fig. A.1) yields the following relations:*

$$P@k(q,y') = \begin{cases} P@k(q,y) & k < b \\ P@k(q,y) + \frac{1}{k} & b \leq k < a + |\mathcal{X}_q^-| \\ \frac{a}{a+|\mathcal{X}_q^-|} & k = a + |\mathcal{X}_q^-| \end{cases} \tag{A.3}$$

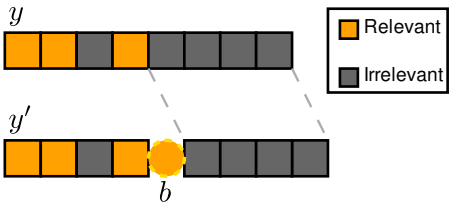
and

$$AP(q,y') = \frac{1}{a}\left((a-1)\,AP(q,y) \;+\; \frac{a}{b}\right) = \left(\frac{a-1}{a}\right) AP(q,y) + \frac{1}{b} \tag{A.4}$$

The first term in eq. (A.4) re-scales the AP score of y to account for the increased length of y', and the second term adds $\text{P@}b = {}^{a}\!/_{b}$ to the average precision for the new relevant document.

Proposition A.2. *Let y and y' be as in proposition A.1. Then the following relation holds:*

$$\psi(q,y') = \left(\frac{a-1}{a}\right)\psi(q,y) + \sum_{j \in \mathcal{X}_q^-} y'_{aj}\left(\frac{\phi(q,x_a) - \phi(q,x_j)}{a \cdot |\mathcal{X}_q^-|}\right). \tag{A.5}$$



(a)

Ranking	AP
y	$\frac{1}{3}\left(1+1+\frac{3}{4}\right) = 11/12$
y'	$\frac{1}{4}\left(1+1+\frac{3}{4}+\frac{4}{5}\right) = 71/80$

(b)

Figure A.1: (a) A ranking y' is created from y by inserting a new relevant document at position $b = 5$. In this example, $a = 4$ and $|\mathcal{X}_q^-| = 5$. (b) The difference in AP scores for the two rankings follows eq. (A.4).

The first term of eq. (A.5) rescales $\psi(q, y)$ to account for the inclusion of a new (relevant) document, and the second term counts all of the vector differences due to new document. Note that for a fixed position b, y'_{aj} is completely determined:

$$y'_{aj} = \begin{cases} +1 & j \geq (b-1) - (a-1) = b - a \\ -1 & j < b - a \end{cases}, \tag{A.6}$$

assuming that the j index respects the descending discriminant ordering of $\mathcal{X}_q^-$.

Equipped with these observations, we can now derive a dynamic programming algorithm to solve eq. (A.2).

A.2.1 DP algorithm

Proposition A.1 and A.2 provide a convenient decomposition of eq. (A.2) into two terms, one of which depending all but the last relevant document, and the other depending on the position of the last relevant document. Importantly, the first term defines the solution to a sub-problem involving all but the last document, allowing us to solve for the total score recursively, and the second is *independent* of the solution to the sub-problem.

To maximize eq. (A.2) over y, we restrict attention to rankings which place the last relevant document at position b, and then maximize over all choices of b. Let

$$V(a, b) := \max_{y_b^a} \langle w, \psi(q, y_b^a) \rangle - \text{AP}(q, y_b^a), \tag{A.7}$$

where y_b^a is restricted to rankings of only the first a relevant documents, and the a^{th} relevant document is located at position b. The solution of eq. (A.2) can be found by optimizing over all b when all $a = |\mathcal{X}_q^+|$ documents have been inserted:

$$\max_{b \leq |\mathcal{X}_q^+| + |\mathcal{X}_q^-|} V(|\mathcal{X}_q^+|, b).$$

Applying proposition A.1 and proposition A.2, we can rewrite eq. (A.7) as

$$\begin{aligned} V(a,b) &= \max_{y_b^a} \langle w, \psi(q, y_b^a) \rangle - \text{AP}(q, y_b^a) \\ &= \max_{b'<b} \left(\frac{a-1}{a} \right) \langle w, \psi(q, y_{b'}^{a-1}) \rangle + \left\langle w, \sum_{j \in \mathcal{X}_q^-} (y_b^a)_{aj} \frac{\phi(q, x_a) - \phi(q, x_j)}{a \cdot |\mathcal{X}_q^-|} \right\rangle \\ &\qquad - \left(\frac{a-1}{a} \right) \text{AP}(q, y_{b'}^{a-1}) - \frac{1}{b} \\ &= \left(\frac{a-1}{a} \max_{b'<b} V(a-1, b') \right) + \left\langle w, \sum_{j \in \mathcal{X}_q^-} (y_b^a)_{aj} \frac{\phi(q, x_a) - \phi(q, x_j)}{a \cdot |\mathcal{X}_q^-|} \right\rangle - \frac{1}{b}. \end{aligned} \tag{A.8}$$

For the base case of $a = 0$, there are no relevant documents, so $V(0, b) = 0$ for all b. For $a < b$, there is no feasible ranking, so $V(a, b) = -\infty$. The recursive formulation of eq. (A.8) completely characterizes the dynamic programming algorithm to solve for the most violated constraint, which is listed as algorithm A.1.

A.2.2 Complexity

To compute the optimal $\overline{y}$ using algorithm A.1 requires filling in a table of size[1]

$$O\left(|\mathcal{X}_q^+| \times \left(|\mathcal{X}_q^+| + |\mathcal{X}_q^-|\right)\right) = O\left(|\mathcal{X}_q^+| \times |\mathcal{X}_q^-|\right).$$

Filling in each cell of the table thus requires maximizing over $O\left(|\mathcal{X}_q^-|\right)$ entries in the previous row.

However, the per-cell cost can be amortized to $\mathcal{O}\left(|\mathcal{X}_q^+| + |\mathcal{X}_q^-|\right)$ per row — $\mathcal{O}(1)$ per cell — by working in ascending order of b over the range $[a, |\mathcal{X}_q^-| + |\mathcal{X}_q^+|]$, and re-using the $\max$ computed in the previous cell. Similarly, the second term in eq. (A.8)

[1] In practice, it is generally assumed that $|\mathcal{X}_q^+| \in \mathcal{O}(|\mathcal{X}_q^-|)$, and $\mathcal{X}_q^+$ is often small enough to be treated as constant.

Algorithm A.1 Cutting plane algorithm for average precision

Input: parameter vector w, query q, positive and negative results $\mathcal{X}_q^+$ and $\mathcal{X}_q^-$

Output: $\overline{y} := \operatorname{argmax}_{y \in \mathcal{Y}} \langle w, \psi(q, y) \rangle + \Delta_{\text{AP}}(y_q, y)$

Initialize $(|\mathcal{X}_q^+| + 1) \times (|\mathcal{X}_q^+| + |\mathcal{X}_q^-|)$ value array V and pointer array Y

$\forall b : \; V[0, b] \leftarrow 0$

$\forall a, b \text{ s. t. } 0 < a < b : \; V[a, b] \leftarrow -\infty$

for $a = 1, 2, \ldots, |\mathcal{X}_q^+|$ **do**

  for $b = a, a + 1, \ldots, |\mathcal{X}_q^+| + |\mathcal{X}_q^-|$ **do**

    $V[a, b] \leftarrow$ eq. (A.8)

    $Y[a, b] \leftarrow \operatorname{argmax}_{b' < b}$ eq. (A.8)

  end for

end for

return ranking $\overline{y}$ generated by following $Y\left[|\mathcal{X}_q^+|, \overline{b}\right]$ where

$$\overline{b} := \operatorname*{argmax}_{b'} V\left[|\mathcal{X}_q^+|, b'\right].$$

can be amortized by adjusting the score from b to $b + 1$: only a single y_{aj} changes from $V(a, b)$ to $V(a, b+1)$, so the second term can be updated in $\mathcal{O}(1)$ time as well. Therefore, the total cost of the algorithm is $\mathcal{O}\left(|\mathcal{X}_q^+|^2 + |\mathcal{X}_q^+| \times |\mathcal{X}_q^-|\right) = \mathcal{O}\left(|X_q^+| \times |\mathcal{X}_q^-|\right)$.

Appendix B

Online k-Means

B.1 Introduction

Clustering a data set $\mathcal{X} \subset \mathbb{R}^d$ via k-means is a central step to many of the methods discussed in this dissertation, including codebook learning for vector quantization (chapter 5), generating splitting rules for spatial tree data structures (chapter 6), and clustering songs for playlist generation (chapter 7). In each case, the set to be clustered contains a large number of points, ranging from 10^5 (song clustering) to 10^7 (codebook learning).

In these large-scale settings, batch clustering algorithms can be prohibitively expensive, both in terms of time and space complexity. This appendix describes an online k-means algorithm derived from Hartigan's method (Hartigan, 1975, Telgarsky and Vattani, 2010) to generate cluster centers with minimal space complexity $\mathcal{O}(kd)$. The method is originally due to Telgarsky (2010), and is included here for completeness.

B.2 Hartigan's method and online k-means

All k-means algorithms seek to minimize the sum-squared error objective over the k cluster centroids $\{ \mu_i \}_{i=1}^k$:

$$f(\mu_1, \mu_2, \ldots, \mu_k) := \sum_{x_i \in \mathcal{X}} \|x_i - \mu(x_i)\|^2 \tag{B.1}$$

$$\mu(x) := \operatorname*{argmin}_{\mu_i} \|x - \mu_i\|^2.$$

Most commonly, the centroids are optimized via Lloyd's method (MacQueen, 1967, Lloyd, 1982), which repeatedly constructs k clusters

$$C_j^{t+1} \leftarrow \left\{ x_i \;\middle|\; j = \operatorname*{argmin}_{\iota} \|x_i - \mu_\iota^t\|^2 \right\},$$

updates each centroid

$$\mu_j^{t+1} \leftarrow \mu(C_j^{t+1})$$

$$\mu(C) := \frac{1}{|C|} \sum_{x \in C} x,$$

repeating for $t = 1, 2, \ldots,$ until convergence.

The method of Hartigan (1975) is similar, but differs in that updates are performed one point at a time, rather than in batches. At step t, a point $x_t \in \mathcal{X}$ is selected and reassigned so as to decrease eq. (B.1). Let C_t denote the cluster currently containing x_t. As shown by Telgarsky and Vattani (2010), the change in eq. (B.1) due to moving x_t from C_t to some cluster C_j can be computed as:

$$\Delta f(x_t, C_j) := \frac{|C_j|}{|C_j| + 1} \|\mu(C_j) - x_t\|^2 - \frac{|C_t|}{|C_t| - 1} \|\mu(C_t) - x_t\|^2, \tag{B.2}$$

and x_t is moved to the C_j that maximally decreases cost: $\operatorname{argmin}_j \Delta f(x_t, C_j)$.

This point-wise reassignment strategy lends itself naturally to an online clustering algorithm (Telgarsky, 2010). When a new point x_t arrives, it can be arbitrarily assigned to a cluster C_t (equivalently, to a centroid μ_t, and then reassigned to the cluster (centroid) which maximally decreases cost. Functionally, this is equivalent to assigning x_t to the cluster C_j which minimizes the first term of eq. (B.2) while ignoring the second term (which is constant over the argmin). Note that all that is needed for this operation are the centroids μ_j and cluster sizes $n_j := |C_j|$; after updating these parameters, the point x_t may be discarded. The resulting algorithm is listed as algorithm B.1.

Algorithm B.1 Online Hartigan k-means

Input: $k \in \mathbb{Z}_+$, data sequence $x_1, x_2, \cdots \in \mathbb{R}^d$

Output: k centroids $\{ \mu_1, \mu_2, \ldots, \mu_k \} \subset \mathbb{R}^d$

1: Initialize each $\mu_i \leftarrow \mathbf{0}$ and counters $n_i \leftarrow 0$.

2: **for** $t = 1, 2, \ldots$ **do**

3: Find the closest centroid to x_t:

$$\iota \leftarrow \operatorname*{argmin}_{1 \le i \le k} \frac{n_i}{n_i + 1} \|\mu_i - x_t\|^2$$

4: Update the centroid μ_ι and count n_ι:

$$\mu_\iota \leftarrow \frac{n_\iota}{n_\iota + 1} \mu_\iota + \frac{1}{n_\iota + 1} x_t$$
$$n_\iota \leftarrow n_\iota + 1$$

5: **end for**

6: **return** $\{ \mu_1, \mu_2, \ldots, \mu_k \}$

Bibliography

Sameer Agarwal, Joshua Wills, Lawrence Cayton, G.R.G. Lanckriet, David Kriegman, and Serge Belongie. Generalized non-metric multi-dimensional scaling. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, AISTATS, 2007.

A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972.

Chris Anderson. *The Long Tail: Why the Future of Business Is Selling Less of More*. Hyperion, July 2006.

Anelia Angelova. Data pruning. Master's thesis, California Institute of Technology, 2004.

Anelia Angelova, Yaser Abu-Mostafa, and Pietro Perona. Pruning training sets for learning of object categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, pages 494–501, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2 (4):343–370, 1988. ISSN 0885-6125.

A. Asuncion and D.J. Newman. UCI machine learning repository, 2007. URL http://www.ics.uci.edu/~mlearn/MLRepository.html.

Jean-Julien Aucouturier and François Pachet. Music similarity measures: What's the use? In *Proceedings of the 3rd International Conference on Music Information Retrieval*, ISMIR, pages 157–163, 2002.

Francis R. Bach. Consistency of the group lasso and multiple kernel learning. *Journal of Machine Learning Research*, 9:1179–1225, 2008. ISSN 1532-4435.

Ricardo A. Baeza-Yates and Berthier Ribeiro-Neto. *Modern Information Retrieval*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.

A. Barla, F. Odone, and A. Verri. Histogram intersection kernel for image classification. In *International Conference on Image Processing*, volume 3 of *ICIP*, pages III – 513–16 vol.2, 2003.

L. Barrington, A. Chan, D. Turnbull, and G. Lanckriet. Audio information retrieval using semantic similarity. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 2 of *ICASSP*, pages II–725 –II–728, april 2007.

Luke Barrington, Reid Oda, and G.R.G. Lanckriet. Smarter than genius? Human evaluation of music recommender systems. In *Proceedings of the 10th International Conference on Music Information Retrieval*, 2009.

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 Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems*, NIPS, Cambridge, MA, 2004. MIT Press.

J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, Sep. 1975. ISSN 0001-0782.

Adam Berenzweig, Beth Logan, Daniel P.W. Ellis, and Brian Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

Bonnie Berger and Peter W. Shor. Approximation alogorithms for the maximum acyclic subgraph problem. In *SODA '90: Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 236–243, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics. ISBN 0-89871-251-3.

Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, 2011.

A. Bhattacharyya. On a measure of divergence between two statistical populations defined by probability distributions. *Bulletin of the Calcutta Mathematical Society*, 35: 99–109, 1943.

Mikhail Bilenko, Sugato Basu, and Raymond J. Mooney. Integrating constraints and metric learning in semi-supervised clustering. In *Proceedings of the 21st International Conference on Machine Learning*, ICML, pages 81–88, 2004.

Ingwer Borg and Patrick J.F. Groenen. *Modern Multidimensional Scaling: Theory and Applications*. Springer-Verlag, second edition, 2005.

K. Bosteels, E. Pampalk, and E.E. Kerre. Evaluating and analysing dynamic playlist generation heuristics using radio logs and fuzzy set theory. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009.

S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and trends in machine learning*, 3(1):1–122, 2011.

Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

C.R. Buchanan. Semantic-based audio recognition and retrieval. Master's thesis, School of Informatics, University of Edinburgh, Edinburgh, U.K., 1995.

Chris Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Greg Hullender. Learning to rank using gradient descent. In *Proceedings of the 22nd International Conference on Machine Learning*, ICML, pages 89–96, New York, NY, USA, 2005. ACM.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, Sep. 1995.

Rui Cai, Chao Zhang, Lei Zhang, and Wei-Ying Ma. Scalable music recommendation by search. In *International Conference on Multimedia*, pages 1065–1074, 2007.

Lawrence Cayton. Fast nearest neighbor retrieval for bregman divergences. In *Proceedings of the 25th International Conference on Machine Learning*, ICML, pages 112–119, 2008.

O. Celma. *Music Recommendation and Discovery in the Long Tail*. Springer, 2010.

Soumen Chakrabarti, Rajiv Khanna, Uma Sawant, and Chiru Bhattacharyya. Structured learning for non-smooth ranking losses. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge Discovery and Data mining*, KDD, pages 88–96, New York, NY, USA, 2008. ACM.

T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 3rd edition, 2009.

Corinna Cortes, Mehryar Mohri, and Afshin Rostamizadeh. Learning non-linear combinations of kernels. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 396–404, 2009.

Trevor F. Cox and Michael A.A. Cox. *Multidimensional Scaling*. Chapman and Hall, 1994.

Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2002. ISSN 1532-4435.

S.J. Cunningham, D. Bainbridge, and A. Falconer. "More of an art than a science": Supporting the creation of playlists and mixes. In *Proceedings of the 7th International Conference on Music Information Retrieval*, ISMIR, 2006.

R.B. Dannenberg, W.P. Birmingham, G. Tzanetakis, C. Meek, N. Hu, and B. Pardo. The MUSART testbed for query-by-humming evaluation. *Computer Music Journal*, 28(2):34–48, 2004.

Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *ACM Symposium on Theory of Computing*, STOC, pages 537–546, 2008.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, SCG '04, pages 253–262, New York, NY, USA, 2004. ACM. ISBN 1-58113-885-7.

Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th International Conference on Machine learning*, ICML, pages 209–216, New York, NY, USA, 2007. ACM.

Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. In *Readings in speech recognition*, pages 65–74. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. ISBN 1-55860-124-4.

A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum Likelihood from Incomplete Data via the EM Algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977. ISSN 00359246.

Mukund Deshpande and George Karypis. Item-based top-n recommendation algorithms. *ACM Transactions on Information Systems*, 22:143–177, January 2004. ISSN 1046-8188.

M. Dopler, M. Schedl, T. Pohle, and P. Knees. Accessing music collections via representative cluster prototypes in a hierarchical organization scheme. In *Proceedings of the 9th International Conference on Music Information Retrieval*, ISMIR, 2008.

G. Eisenberg, J.M. Batke, and T. Sikora. Beatbank-an mpeg-7 compliant query by tapping system. In *Audio Engineering Society Convention*, 2004.

D. Ellis, B. Whitman, A. Berenzweig, and S. Lawrence. The quest for ground truth in musical artist similarity. In *Proceedings of the 2nd International Symposium on Music Information Retrieval*, ISMIR, pages 170–177, October 2002.

Christos Faloutsos and King-Ip Lin. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD, pages 163–174, New York, NY, USA, 1995. ACM. ISBN 0-89791-731-6.

L. Fei-Fei and P. Perona. A bayesian hierarchical model for learning natural scene categories. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2005.

B. Fields. *Contextualize Your Listening: The Playlist as Recommendation Engine*. PhD thesis, Goldsmiths, University of London, April 2011.

B. Fields, C. Rhodes, and M. d'Inverno. Using song social tags and topic models to describe and compare playlists. In *Workshop on Music Recommendation and Discovery*, WOMRAD, 2010.

Arthur Flexer, Dominik Schnitzer, Martin Gasser, and Gerhard Widmer. Playlist generation using start and end songs. In *Proceedings of the 9th International Conference on Music Information Retrieval*, ISMIR, 2008.

C. Galleguillos, B. McFee, S. Belongie, and G.R.G. Lanckriet. Multi-class object localization by combining local contextual interactions. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, pages 113–120, 2010.

C. Galleguillos, B. McFee, S. Belongie, and G.R.G. Lanckriet. From region similarity to category discovery. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2011.

Z. Gantner, S. Rendle, C. Freudenthaler, and L. Schmidt-Thieme. MyMediaLite: A free recommender system library. In *Proceedings of the third ACM conference on Recommender systems*, RecSys, 2011.

Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979. ISBN 0716710447.

J. M. Geusebroek, G. J. Burghouts, and A. W. M. Smeulders. The Amsterdam library of object images. *International Journal of Computer Vision*, 61(1):103–112, 2005.

Amir Globerson and Sam Roweis. Metric learning by collapsing classes. In Yair Weiss, Bernhard Schölkopf, and John Platt, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 451–458, Cambridge, MA, 2006. MIT Press.

Amir Globerson and Sam Roweis. Visualizing pairwise similarity via semidefinite embedding. In *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics*, AISTATS, 2007.

David Goldberg, David Nichols, Brian M. Oki, and Douglas Terry. Using collaborative filtering to weave an information tapestry. *Commun. ACM*, 35:61–70, December 1992. ISSN 0001-0782.

Jacob Goldberger, Sam Roweis, Geoffrey Hinton, and Ruslan Salakhutdinov. Neighborhood components analysis. In Lawrence K. Saul, Yair Weiss, and Léon Bottou, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 513–520, Cambridge, MA, 2005. MIT Press.

John A. Hartigan. *Clustering Algorithms*. John Wiley & Sons, Inc., New York, NY, USA, 1975.

J.R. Hershey and P.A. Olsen. Approximating the kullback leibler divergence between gaussian mixture models. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, volume 4 of *ICASSP*, pages IV–317 –IV–320, 2007.

M. Hoffman, D. Blei, and P. Cook. Content-Based Musical Similarity Computation Using the Hierarchical Dirichlet Process. In *Proceedings of the 9th International Conference on Music Information Retrieval*, ISMIR, pages 349–354, 2008.

M. Hoffman, D. Blei, and P. Cook. Easy as CBA: A simple probabilistic model for tagging music. In *Proceedings of the 10th International Conference on Music Information Retrieval*, ISMIR, 2009.

M. Hoffman, D. Blei, and F. Bach. Online learning for latent dirichlet allocation. In *Advances in Neural Information Processing Systems*, NIPS. 2010.

Yifan Hu, Y. Koren, and C. Volinsky. Collaborative filtering for implicit feedback datasets. In *Eighth IEEE International Conference on Data Mining*, ICDM, pages 263 –272, 2008.

Paul Jaccard. Étude comparative de la distribution florale dans une portion des Alpes et des Jura. *Bulletin del la Société Vaudoise des Sciences Naturelles*, 37:547–579, 1901.

Saketha Nath Jagarlapudi, Dinesh G, Raman S, Chiranjib Bhattacharyya, Aharon Ben-Tal, and Ramakrishnan K.R. On the algorithmics and applications of a mixed-norm based kernel learning formulation. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 844–852. 2009.

Kalervo Järvelin and Jaana Kekäläinen. IR evaluation methods for retrieving highly relevant documents. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR, pages 41–48, New York, NY, USA, 2000. ACM.

Tony Jebara, Risi Kondor, and Andrew Howard. Probability product kernels. *Journal of Machine Learning Research*, 5:819–844, December 2004.

Jesper Hojvang Jensen, Daniel P. W. Ellis, Mads G. Christensen, and Soren Holdt Jensen. Evaluation of distance measures between gaussian mixture models of MFCCs. In *Proceedings of the 8th International Conference on Music Information Retrieval*, ISMIR, pages 107–108, 2007.

Thorsten Joachims. A support vector method for multivariate performance measures. In *Proceedings of the 22nd International Conference on Machine learning*, ICML, pages 377–384, New York, NY, USA, 2005. ACM.

Thorsten Joachims, Thomas Finley, and Chun-Nam John Yu. Cutting-plane training of structural svms. *Machine Learning*, 77(1):27–59, 2009. ISSN 0885-6125.

A. Kapur, M. Benning, and G. Tzanetakis. Query-by-beat-boxing: Music retrieval for the DJ. In *Proceedings of the 5th International Conference on Music Information Retrieval*, ISMIR, pages 170–177, 2004.

Maurice Kendall and Jean Dickinson Gibbons. *Rank correlation methods*. Oxford University Press, 1990.

J.H. Kim, B. Tomasik, and D. Turnbull. Using artist similarity to propagate semantic information. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009.

Marius Kloft, Ulf Brefeld, Soeren Sonnenburg, Pavel Laskov, Klaus-Robert Müller, and Alexander Zien. Efficient and accurate lp-norm multiple kernel learning. In Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 997–1005. 2009.

P. Knees, T. Pohle, M. Schedl, and G. Widmer. Combining audio-based similarity with web-based data to accelerate automatic music playlist generation. In *ACM international workshop on multimedia information retrieval*, 2006. ISBN 1-59593-495-2.

N. Koenigstein, G. R. G. Lanckriet, B. McFee, and Y. Shavitt. Collaborative filtering based on P2P networks. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, ISMIR, August 2010.

J.B. Kruskal. Nonmetric multidimensional scaling: a numerical method. *Psychometrika*, 29(2), June 1964.

S. Kullback. *Information theory and statistics*. Dover Publications, New York, NY, USA., 2nd edition, 1968.

G.R.G. Lanckriet, Nello Cristianini, Peter Bartlett, Laurent El Ghaoui, and Michael I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004. ISSN 1533-7928.

J.H. Lee. How similar is too similar?: Exploring users' perceptions of similarity in playlist evaluation. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, 2011.

Yen-Yu Lin, Tyng-Luh Liu, and Chiou-Shann Fuh. Dimensionality reduction for data in multiple feature representations. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 961–968. 2009.

Ting Liu, Andrew W. Moore, Alexander Gray, and Ke Yang. An investigation of practical approximate nearest neighbor algorithms. In *Advances in Neural Information Processing Systems*, NIPS, pages 825–832. 2005.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2):129 – 137, mar 1982. ISSN 0018-9448.

B. Logan. Content-based playlist generation: exploratory experiments. In *Proceedings of the 2nd International Symposium on Music Information Retrieval*, ISMIR, 2002.

B. Logan. Music recommendation from song sets. In *Proceedings of the 5th International Conference on Music Information Retrieval*, ISMIR, 2004.

Beth Logan and Ariel Salomon. A music similarity function based on signal analysis. *IEEE International Conference on Multimedia and Expo*, page 190, 2001.

J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297, 1967.

P. C. Mahalanobis. On the generalised distance in statistics. In *Proceedings National Institute of Science, India*, volume 2, pages 49–55, April 1936.

François Maillet, Douglas Eck, Guillaume Desjardins, and Paul Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *Proceedings of the 10th International Conference on Music Information Retrieval*, 2009.

C.D. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

Marcin Marszałek and Cordelia Schmid. Accurate object localization with shape masks. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR, 2007.

B. McFee and G.R.G. Lanckriet. Heterogeneous embeddding for subjective artist similarity. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009a.

B. McFee and G.R.G. Lanckriet. Partial order embedding with multple kernels. In *Proceedings of the 26th International Conference on Machine Learning*, ICML, pages 721–728, 2009b.

B. McFee and G.R.G. Lanckriet. Metric learning to rank. In Johannes Fürnkranz and Thorsten Joachims, editors, *Proceedings of the 27th International Conference on Machine Learning*, ICML, pages 775–782, Haifa, Israel, June 2010.

B. McFee and G.R.G. Lanckriet. Learning multi-modal similarity. *Journal of Machine Learning Research*, 12:491–523, February 2011.

B. McFee and G.R.G. Lanckriet. Large-scale music similarity search with spatial trees. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 55–60, 2011a.

B. McFee and G.R.G. Lanckriet. The natural language of playlists. In *Proceedings of the 12th International Society for Music Information Retrieval Conference*, ISMIR, pages 537–541, 2011b.

B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning similarity from collaborative filters. In *Proceedings of the 11th International Society for Music Information Retrieval Conference*, ISMIR, 2010.

B. McFee, C. Galleguillos, and G.R.G. Lanckriet. Contextual object localization with multiple kernel nearest neighbor. *IEEE Transactions on Image Processing*, 20(2): 570–585, 2011. ISSN 1057-7149.

B. McFee, L. Barrington, and G.R.G. Lanckriet. Learning content similarity for music recommendation. *IEEE Transactions on Audio, Speech, and Language Processing*, 2012a. To appear.

B. McFee, T. Bertin-Mahieux, D.P.W. Ellis, and G.R.G. Lanckriet. The million song dataset challenge. In *Proceedings of the 2nd International Workshop on Advances in Music Information Retrieval*, AdMIRe, 2012b.

Martin F. McKinney and Jeroen Breebaart. Features for audio and music classification. In *Proceedings of the 4th International Conference on Music Information Retrieval*, ISMIR, 2003.

J. Opatrny. Total ordering problem. *SIAM Journal on Computing*, (8):111–114, 1979.

Elias Pampalk, Tim Pohle, and Gerhard Widmer. Dynamic playlist generation based on skipping behaviour. In *Proceedings of the 6th International Conference on Music Information Retrieval*, ISMIR, 2005.

Pandora Media, Inc. The music genome project, 2010. http://www.pandora.com/corporate/.

S. Pauws and B. Eggen. PATS: Realization and user evaluation of an automatic playlist generator. In *Proceedings of the 2nd International Symposium on Music Information Retrieval*, ISMIR, 2002.

John C. Platt, Christopher J. C. Burges, Steven Swenson, Christopher Weare, and Alice Zheng. Learning a gaussian process prior for automatically generating music playlists. In *Advances in Neural Information Processing Systems*. MIT Press, 2002.

Lawrence Rabiner and Biing-Hwang Juang. *Fundamentals of speech recognition*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993. ISBN 0-13-015157-2.

R. Ragno, C. J. C. Burges, and C. Herley. Inferring similarity between music objects with application to playlist generation. In *7th ACM SIGMM international workshop on Multimedia information retrieval*, 2005.

Nikhil Rasiwasia, P.J. Moreno, and N. Vasconcelos. Bridging the gap: Query by semantic example. *IEEE Transactions on Multimedia*, 9(5):923 –938, aug. 2007. ISSN 1520-9210.

J. Reiss, J.J. Aucouturier, and M. Sandler. Efficient multidimensional searching routines for music information retrieval. In *Proceedings of the 2nd International Symposium on Music Information Retrieval*, ISMIR, 2001.

S. Rendle, C. Freudenthaler, Z. Gantner, and L. Schmidt-Thieme. BPR: Bayesian personalized ranking from implicit feedback. In *Uncertainty in Artificial Intelligence*, UAI, 2009.

Volker Roth, Julian Laub, Joachim M. Buhmann, and Klaus-Robert Müller. Going metric: denoising pairwise data. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 809–816, Cambridge, MA, 2003. MIT Press.

Gerard Salton and Chris Buckley. Term weighting approaches in automatic text retrieval. Technical report, Ithaca, NY, USA, 1987.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Reidl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, WWW '01, pages 285–295, New York, NY, USA, 2001. ACM. ISBN 1-58113-348-0.

Dominik Schnitzer, Arthur Flexer, and Gerhard Widmer. A filter-and-refine indexing method for fast similarity search in millions of music tracks. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009.

B. Schölkopf and A.J. Smola. *Learning with Kernels*. MIT Press, 2002.

Bernhard Schölkopf, Ralf Herbrich, Alex J. Smola, and Robert Williamson. A generalized representer theorem. In *Proceedings of the 14th annual Conference on Computational Learning Theory*, pages 416–426, 2001.

Matthew Schultz and Thorsten Joachims. Learning a distance metric from relative comparisons. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems*, NIPS, Cambridge, MA, 2004. MIT Press.

Barry Schwartz. *The Paradox of Choice: Why More Is Less*. HarperCollins, 2004.

Klaus Seyerlehner, Gerhard Widmer, and Peter Knees. Frame level audio similarity - a codebook approach. In *Proceedings of the 11th International Conference on Digital Audio Effects*, DAFx, Espoo, FI, 2008.

Shai Shalev-Shwartz, Yoram Singer, and Andrew Y. Ng. Online and batch learning of pseudo-metrics. In *Proceedings of the 21st International Conference on Machine Learning*, ICML, 2004.

Blake Shaw, Bert Huang, and Tony Jebara. Learning a distance metric from a network. In *Advances in Neural Information Processing Systems*, NIPS. 2011.

Chunhua Shen, Junae Kim, Lei Wang, and Anton van den Hengel. Positive semidefinite metric learning with boosting. In *Advances in Neural Information Processing Systems*, NIPS. 2009.

Noam Shental, Tomer Hertz, Daphna Weinshall, and Misha Pavel. Adjustment learning and relevant components analysis. In *European Conference on Computer Vision*, ECCV, 2002.

Josef Sivic and Andrew Zisserman. Video google: A text retrieval approach to object matching in videos. In *Proceedings of the Ninth IEEE International Conference on Computer Vision - Volume 2*, ICCV '03, pages 1470–, Washington, DC, USA, 2003. IEEE Computer Society. ISBN 0-7695-1950-4. URL http://portal.acm.org/citation.cfm?id=946247.946751.

M. Slaney and W. White. Measuring playlist diversity for recommendation systems. In *1st ACM workshop on Audio and music computing multimedia*, AMCMM '06, pages 77–82, New York, NY, USA, 2006. ACM. ISBN 1-59593-501-0.

M. Slaney and W. White. Similarity based on rating data. In *Proceedings of the 8th International Conference on Music Information Retrieval*, ISMIR, pages 479–484, 2007.

M. Slaney, K. Weinberger, and W. White. Learning a metric for music similarity. In *Proceedings of the 9th International Conference on Music Information Retrieval*, ISMIR, pages 313–318, September 2008.

Sören Sonnenburg, Gunnar Rätsch, Christin Schäfer, and Bernhard Schölkopf. Large scale multiple kernel learning. *Journal of Machine Learning Research*, 7:1531–1565, 2006. ISSN 1532-4435.

Richard Stenzel and Thomas Kamps. Improving content-based similarity measures by training a collaborative model. In *Proceedings of the 6th International Conference on Music Information Retrieval*, ISMIR, pages 264–271, 2005.

S. Sundaram and S. Narayanan. Audio retrieval by latent perceptual indexing. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, ICASSP, pages 49 –52, 2008.

Matus Telgarsky. Personal communication, 2010.

Matus Telgarsky and Andrea Vattani. Hartigan's method: k-means without Voronoi. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *AISTATS*, pages 820–827, 2010.

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

W.S. Torgerson. Multidimensional scaling: 1. theory and method. *Psychometrika*, 17: 401–419, 1952.

Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6:1453–1484, 2005. ISSN 1532-4435.

D. Turnbull, L. Barrington, D. Torres, and G.R.G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):467–476, feb. 2008. ISSN 1558-7916.

Douglas Turnbull, Luke Barrington, David Torres, and G.R.G. Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR, pages 439–446, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-597-7.

R. Typke. *Music Retrieval based on Melodic Similarity*. PhD thesis, Utrecht University, 2006.

G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on Speech and Audio Processing*, 10(5):293 – 302, July 2002. ISSN 1063-6676.

J.K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.

L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Nakul Verma, Samory Kpotufe, and Sanjoy Dasgupta. Which spatial partition trees are adaptive to intrinsic dimension? In *Uncertainty in Artificial Intelligence*, UAI, pages 565–574, 2009.

Alexander Vezhnevets and Olga Barinova. Avoiding boosting overfitting by removing confusing samples. In *European Conference on Machine Learning*, ECML, pages 430–441, 2007.

Fabio Vignoli and Steffen Pauws. A music retrieval system based on user-driven similarity and its evaluation. In *Proceedings of the 6th International Conference on Music Information Retrieval*, ISMIR, 2005.

Maksims N. Volkovs and Richard S. Zemel. Boltzrank: learning to maximize expected ranking gain. In *Proceedings of the 26th International Conference on Machine Learning*, ICML, pages 1089–1096, New York, NY, USA, 2009. ACM.

Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained k-means clustering with background knowledge. In *Proceedings of the 18th International Conference on Machine Learning*, ICML, pages 577–584, 2001.

F. Wang, X. Wang, B. Shao, T. Li, and M. Ogihara. Tag integrated multi-label music style classification with hypergraph. In *Proceedings of the 10th International Society for Music Information Retrieval Conference*, ISMIR, 2009.

Kilian Q. Weinberger and Lawrence K. Saul. Fast solvers and efficient implementations for distance metric learning. In *Proceedings of the 25th International Conference on Machine Learning*, ICML, 2008.

Kilian Q. Weinberger, John Blitzer, and Lawrence K. Saul. Distance metric learning for large margin nearest neighbor classification. In Yair Weiss, Bernhard Schölkopf, and John Platt, editors, *Advances in Neural Information Processing Systems*, NIPS, pages 451–458, Cambridge, MA, 2006. MIT Press.

B. Whitman, G. Flake, and S. Lawrence. Artist detection in music with minnowmatch. In *Neural Networks for Signal Processing XI, 2001. Proceedings of the 2001 IEEE Signal Processing Society Workshop*, pages 559–568, 2001.

Eric P. Xing, Andrew Y. Ng, Michael I. Jordan, and Stuart Russell. Distance metric learning, with application to clustering with side-information. In *Advances in Neural Information Processing Systems*, NIPS, pages 505–512, Cambridge, MA, 2003. MIT Press.

Jun Xu and Hang Li. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR, pages 391–398, New York, NY, USA, 2007. ACM.

Yiming Ying and Peng Li. Distance metric learning with eigenvalue optimization. *Journal of Machine Learning Research*, 13:1–26, January 2012.

K. Yoshii, M. Goto, K. Komatani, T. Ogata, and H.G. Okuno. An efficient hybrid music recommender system using an incrementally trainable probabilistic generative model. *IEEE Transactions on Audio, Speech, and Language Processing*, 16(2):435–447, 2008.

Chun-Nam John Yu and Thorsten Joachims. Training structural svms with kernels using sampled cuts. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD, pages 794–802, New York, NY, USA, 2008. ACM.

Yisong Yue, Thomas Finley, Filip Radlinski, and Thorsten Joachims. A support vector method for optimizing average precision. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR, pages 271–278, New York, NY, USA, 2007. ACM.

Elena Zheleva, John Guiver, Eduarda Mendes Rodrigues, and Nataša Milić-Frayling. Statistical models of music-listening sessions in social media. In *Proceedings of the 19th international conference on World wide web*, WWW '10, pages 1019–1028, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-799-8.

Alexander Zien and Cheng Soon Ong. Multiclass multiple kernel learning. In *Proceedings of the 24th International Conference on Machine Learning*, ICML, pages 1191–1198, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.