

Large-Scale Pattern Discovery in Music

Thierry Bertin-Mahieux

Submitted in partial fulfillment of the
requirements for the degree
of Doctor of Philosophy
in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2013

©2013

Thierry Bertin-Mahieux

All Rights Reserved

ABSTRACT

Large-Scale Pattern Discovery in Music

Thierry Bertin-Mahieux

This work focuses on extracting patterns in musical data from very large collections. The problem is split in two parts. First, we build such a large collection, the Million Song Dataset, to provide researchers access to commercial-size datasets. Second, we use this collection to study cover song recognition which involves finding harmonic patterns from audio features.

Regarding the Million Song Dataset, we detail how we built the original collection from an online API, and how we encouraged other organizations to participate in the project. The result is the largest research dataset with heterogeneous sources of data available to music technology researchers. We demonstrate some of its potential and discuss the impact it already has on the field.

On cover song recognition, we must revisit the existing literature since there are no publicly available results on a dataset of more than a few thousand entries. We present two solutions to tackle the problem, one using a hashing method, and one using a higher-level feature computed from the chromagram (dubbed the 2DFTM). We further investigate the 2DFTM since it has potential to be a relevant representation for any task involving audio harmonic content.

Finally, we discuss the future of the dataset and the hope of seeing more work making use of the different sources of data that are linked in the Million Song Dataset. Regarding cover songs, we explain how this might be a first step towards defining a *harmonic manifold of music*, a space where harmonic similarities between songs would be more apparent.

Table of Contents

I	Introduction	1
1	Introduction	2
1.1	Scaling	3
1.2	Music information retrieval	4
1.3	Contributions	5
1.4	Thesis overview	5
2	Music audio features	6
2.1	Fourier analysis	6
2.2	Pitch-related features	8
2.3	Timbre-related features	10
2.3.1	The Echo Nest timbre features	11
2.3.2	Mel-frequency cepstrum coefficients	12
2.4	Beat-alignment	13
2.5	Features derived from The Echo Nest features	14
3	Machine learning	15
3.1	Feature representation	16
3.1.1	Vector quantization	16
3.1.2	Dimensionality reduction	17
3.2	Supervised learning	20
3.2.1	Decision trees	21
3.2.2	Nearest neighbors	22

3.2.3	Unbiased estimates	23
3.3	Unsupervised learning	23
3.3.1	Clustering	24
II	The Million Song Dataset	26
4	Introduction to the MSD	27
4.1	Problem statement	27
4.2	Other resources	28
4.3	Goals for the MSD	32
4.4	Audio	32
5	Creation of the dataset	33
5.1	History	33
5.2	Creation and datasets	34
5.2.1	The Echo Nest API	34
5.2.2	Song selection	35
5.2.3	Cover songs	36
5.2.4	Lyrics	36
5.2.5	Tags and song similarity	37
5.2.6	Taste profiles	38
5.2.7	Audio collection	38
5.3	Distribution	38
5.4	Problems and lessons learned	40
5.4.1	Duplicate songs	40
5.4.2	Duplicate tracks	40
5.4.3	Song - track mismatches	42
6	Applications and impact	44
6.1	Applications	44
6.1.1	Meta data analysis	44

6.1.2	Year prediction	46
6.1.3	Automatic tagging of music	46
6.2	Impact	47
6.2.1	Playlisting	47
6.2.2	Evolution of pop songs	47
6.2.3	The Million Song Dataset Challenge	48
6.2.4	Discussion on MIR evaluations	50
III	Cover song recognition	52
7	Introduction to cover song recognition	53
7.1	Problem definition	54
7.2	Difficulty of comparing chroma features	56
7.3	Cover song recognition systems	58
7.4	Why scale cover song recognition	59
7.5	Conclusion	60
7.5.1	Working from audio	61
8	Hashing methods	63
8.1	Fingerprint features	64
8.1.1	Example	64
8.1.2	Hash code	65
8.1.3	Encoding and Retrieval	68
8.1.4	Training	69
8.2	Results	70
8.3	Conclusion	72
9	Fourier transform methods	73
9.1	Features	74
9.2	Method	75
9.3	Results	77

9.4	A note on error measures	80
9.5	Understanding the 2DFTM feature	82
9.5.1	Clustering features	82
9.5.2	Comparing distributions of 2DFTM features	83
9.5.3	Importance of the phase	84
9.5.4	Beat-alignment for 2DFTM features	86
9.5.5	Value normalization	87
IV	Conclusions	91
10	Conclusions	92
10.1	Cover songs and manifolds	92
10.2	Using the Million Song Dataset	93
V	Bibliography	95
	Bibliography	96
VI	Appendices	108
A	Content of the Million Song Dataset	109
B	Cover song experiment with audio data	111
C	Publications by this author	113

List of Figures

2.1	Example of an audio signal.	7
2.2	Sample audio signal: close-up and individual samples.	8
2.3	Audio signal and corresponding frequency analysis.	9
2.4	Spectrogram.	10
2.5	Chromagram.	12
2.6	Timbre basis from The Echo Nest analyzer.	13
2.7	Mel-frequency cepstrum coefficients.	14
3.1	Volume of an hypercube and its largest contained sphere in 2D and 3D.	18
3.2	Principal components of a 2D Gaussian distribution.	19
3.3	Simple tree structure (decision stump).	22
3.4	1-NN example with two classes in a 2D space.	22
3.5	Swiss roll manifold.	24
5.1	Screen shot of the Second Hand Songs website.	37
5.2	Duplicates of a Lady GaGa song.	41
5.3	Visualization of a mismatch error in the MSD.	42
6.1	Artist name length by average release date of their song.	45
6.2	Word cloud for lyrics of electronic songs.	45
7.1	Lady Gaga covers on YouTube.	54
7.2	Comparison of chroma patches.	57
7.3	Centroids from clustering chroma patches.	58

7.4	Reconstruction and some error measures.	62
8.1	Two beat-aligned chromagrams and their jumpcodes.	65
8.2	Segments, beats, loudness, and beat-aligned chromas.	66
8.3	Closeup of landmarks and associated jumps.	67
8.4	Landmarks and jumpcodes for 4 cover songs.	68
9.1	Examples of 2-dimensional Fourier transforms.	75
9.2	Beat-aligned chroma patch and corresponding 2DFTM.	76
9.3	Recall using the 2DFTM method on the MSD.	80
9.4	Cluster stability for three features.	83
9.5	Learned weights for 2DFTM features.	88
9.6	Learned weights for 2DFTM features after z -scoring.	89
9.7	Principal components.	90

List of Tables

2.1	4th and 5th octaves with the note frequencies.	11
4.1	Size comparison between some other datasets.	31
6.1	Year prediction results.	46
6.2	Top contestants in the MSD Challenge.	49
8.1	Results on 500 binary tasks.	71
9.1	Results on 500 binary tasks.	78
9.2	Results on the training set (12,960 songs).	79
9.3	Results on 1M songs.	79
9.4	Top- k results on 1M songs.	81
9.5	Results on using mean and variance.	84
9.6	Results on using phase.	86
9.7	Results for multiple beat-alignments.	87
9.8	Results on the training set with z -scoring.	89
A.1	List of the 55 fields provided in each per-song HDF5 file in the MSD.	110

Acknowledgments

Getting through the PhD was a team effort, and it is difficult to thank everyone that made it possible for me. This is an attempt, and I apologize to everyone I am forgetting.

First and foremost, I would not be writing this if it was not for my adviser, Dan Ellis. He took a chance on me by letting me join the LabROSA. He showed tremendous patience each time I presented him with a questionable idea and did his best to steer me back to the right track. Dan, I could not have asked for a better mentor.

The LabROSA at Columbia is an extraordinary place to study, and I want to thank everyone that I encountered there, in particular Courtenay Cotton, Michael Mandel, Christine Smit and Ron Weiss. Joining after them felt both like an honor and a great responsibility.

Fortunately, I had the chance to have been prepared by some of the best researchers I know before moving to Columbia, and I would not have made it without them. This goes to everyone at the LISA (Université de Montréal) and the CIRMMT, in particular Douglas Eck, my Master adviser.

Two internship opportunities made these last few years more exciting and forced me to learn what coding actually means. To Ron Weiss at Google who had to review endlessly my commits, I promise, I now put a comma at the end of my comments! To Paul Lamere at The Echo Nest (and Sun Microsystems when I first met him), thank you for all those times you supported me and offered me both opportunities and advice, I would not be here today without you.

I had the chance to be supported by a NSERC Postgraduate scholarship during my PhD years; I wish this great program will keep on supporting Canadian students at home and abroad.

It has been ten years since I started college, and the one constant source of support that got me through it is my family. I will never thank them enough for their support and the trust they have had in me. To my parents: I made it! And it is because of you.

And of course, for encouraging me no matter what, to Caitlin, with all my love.

To Graham G.

Part I

Introduction

Chapter 1

Introduction

The need for methods to manage and organize music has never been greater. Streaming services such as Spotify¹ claim a collection of more than 16 million songs available to listeners. Such amounts require commercial systems to be able to search, group, and recommend music without humans in the loop. These abilities are part of what is called “music data mining”, or “music information retrieval” as it is often referred to by practitioners.

While text data mining has made tremendous progress because of the popularity of search engines such as Google², the exploration of audio data remains challenging. This is particularly true for music; as machines can transcribe speech and identify a speaker, no system can really claim to “understand music” the way humans do. Such an understanding would include recognizing instruments and individual notes, grouping songs in categories based on genre and style, commenting on a particular performance and the underlying emotions, and identifying common motifs in the score.

In this work we focus on the latter, finding motifs in the score, in the case where we have access to the audio but not said score. We believe this problem to be the one where humans will outperform machines the most easily. Reasons include:

- motifs can have variations;
- motifs can span any length in time;
- motifs can be expressed with any instrument or sound effect;

¹<http://www.spotify.com>

²<http://www.google.com>

thus the problem can not be expressed as a straightforward supervised machine learning task (see Chapter 3 if machine learning is an unfamiliar concept).

We approach the problem of finding motifs through the task of identifying cover songs, i.e. a musical work with different recorded interpretations. Such focus is not very restrictive as we are looking for motifs that define a song, therefore that are constant across versions of a song even if the instrumentation, tempo, and style change. The upside is that we gain a measurable goal for our experiments.

Cover song identification has been studied for a few years. The additional constraint we give ourselves is that our system must work on a commercial-size database. This is necessary if we want our research to have any practical impact. Furthermore, it is possible that leveraging more data is the only way for machines to come close to solving the task, as common motifs will repeat themselves across a large number of songs and stand out from the rest of the data.

Given the state of the music information retrieval research field, working on a collection comparable in size to that of a commercial system implies building this collection ourselves. As a consequence, we present in the first part of this thesis the Million Song Dataset, a freely-available collection of audio features and metadata for a million contemporary popular music tracks.

The hope is that, by creating this research collection and providing scalable cover song recognition systems, we pave the way for more intelligent systems in music data mining. Discovering complex patterns in recorded music gets us closer to handling songs the same way humans do.

1.1 Scaling

We will talk often in this work about the need to tackle problems on a larger scale, in terms of data size. The main reasons to justify that need are summarized below.

- For a system to have an impact, it needs to be adopted by users. As we mentioned, commercial systems require algorithms that can deal with millions of songs, often in a few seconds. Working with small datasets encourages the development of computationally expensive algorithms that cannot be adapted to work with those commercial systems. Therefore, it limits the possibility of technology transfer between academic research and the industry.
- In music technology (and other fields), we deal with extremely complex data. In terms of

features, a song can be related to audio descriptors, symbolic data such as a score or lyrics, cultural data regarding its popularity, its era, the region of its artist, its influence on other song, etc. The list can go on almost endlessly. One consequence is that there are subtle but real trends in the data that are difficult to see. As an example, in the Million Song Dataset, only five songs are tagged as “world beat”³. If we were working with a thousand songs, the chance of seeing that tag would be close to zero. Certain trends can only be seen at a large scale.

- There have been a set of recent successes with new algorithms, often dubbed “deep learning methods”. It usually leverages layers of algorithms, a lot of unsupervised learning, and large quantities of data. For instance, to obtain a 70% improvement in object visual recognition, [Le *et al.*, 2011] trained their algorithm using 16K computer processors and 10M images. Having larger collections of music data opens up the possibility of solving existing problems with new methods that require such large resources to train.

In general, there have been many successes in different fields using larger collections, which has fueled a “big data” craze. This term has been so widely used that it difficult to define what it means, but a good starting point is reading [Cohen *et al.*, 2009]. The work by [Le *et al.*, 2011] mentioned before is an example of success in vision. We can not predict whether more data will help resolve issues in music technology, but we need at least to have the possibility to try. Thus, we need larger datasets.

1.2 Music information retrieval

Music information retrieval (MIR) is an interdisciplinary research field focused on the analysis of music data using rigorous scientific tools. This work can be broadly categorized as machine learning applied to audio problems, which encompasses modern statistical tools and digital signal processing. However, MIR also links to musicology, psychology, ethnology and library studies. This explains the very heterogeneous sources of data in the Million Song Dataset (Part II).

³Tag count based on the Last.fm dataset which includes a list of tags applied to songs in the Million Song Dataset. Tags and their respective song count can be found at http://labrosa.ee.columbia.edu/millionsong/sites/default/files/lastfm/lastfm_unique_tags.txt.

1.3 Contributions

The contributions of this thesis are twofold. First, (Part II) the creation of the Million Song Dataset. As part of that project, we:

- created the largest research collection in MIR;
- provided code and demos to encourage its adoption;
- maintained a website to centralize all efforts surrounding the data;
- organized a successful music recommendation contest.

Then (Part III), we used the Million Song Dataset to develop two methods for large-scale cover song recognition. Contributions include:

- driving the scaling of the cover song recognition task in MIR;
- developing a hashing-based solution;
- developing and understanding a fixed-dimensional representation for cover songs.

The experiments in Part III are reproducible (code and data is available) and represent the current published state-of-the-art.

1.4 Thesis overview

The next two chapters give an overview of digital signal processing and music audio features, then machine learning. Chapters 4 and 5 explain why and how we created the Million Song Dataset, while Chapter 6 focus on its impact on the research community. We introduce the cover song recognition task in Chapter 7. Our two methods are described in Chapters 8 and 9. Finally, we conclude in Chapter 10.

Chapter 2

Music audio features

This chapter is a minimalist introduction to digital signal processing. Most of the content is assumed to be known and we present only what is required to introduce the music audio features used in this work. To the reader who wishes to learn more about this topic, here are a few reading suggestions. A great resource for digital signal processing is [Mitra, 2006]. For audio processing with a good overview of music applications, please refer to [Gold *et al.*, 2011]. A description of typical features used for classification in the MIR field can be found in [Aucouturier and Pachet, 2003; Pohle *et al.*, 2005].

2.1 Fourier analysis

For most practical applications, an audio signal (Figure 2.1) is considered encoded using linear pulse code modulation (LPCM) ([Oliver *et al.*, 1948]) which is the basis of the Windows *WAVE* format¹. It means that a continuous audio signal has been sampled every T seconds. The value $1/T$ is the sampling frequency, the most common value for commercial music is $44.1kHz$. At every sample the amplitude (i.e. the change of atmospheric pressure) of the signal is recorded (Figure 2.2). The higher the amplitude, the louder the audio sounds.

The discrete fourier transform (DFT) is a way to express a time series in the frequency domain (Figure 2.3). It is essential since, loosely speaking, it is the active frequencies that get recognized by the human brain. Given a finite duration sequence $x(n), 0 \leq n \leq N - 1$, the DFT is defined

¹<http://www.digitalpreservation.gov/formats/fdd/fdd000011.shtml>

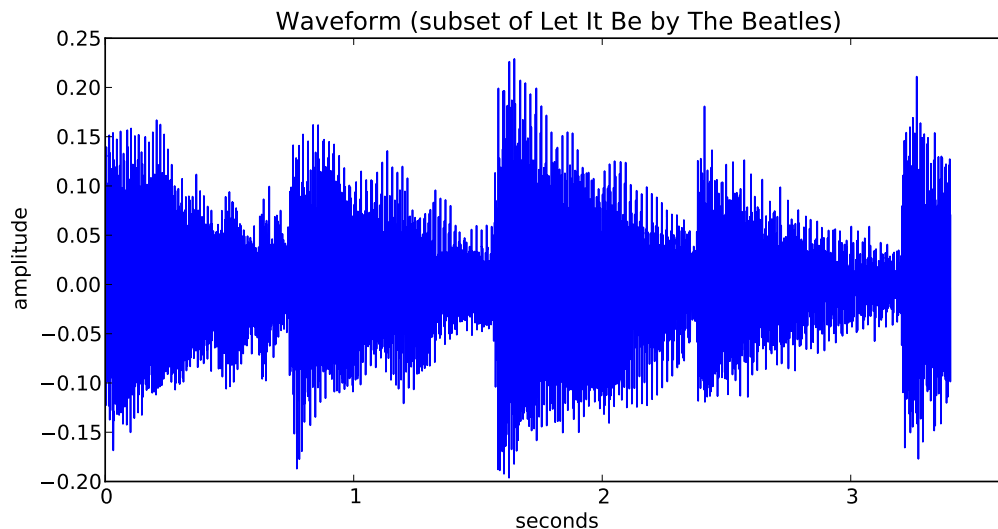


Figure 2.1: Example of an audio signal.

as

$$X(k) = \sum_{n=0}^{N-1} x_n W^{nk}$$

with $W = e^{-j2\pi/N}$. It can be interpreted as the projection of a N -dimensional vector into a sinusoidal basis set. The DFT can be efficiently computed using the Fast Fourier Transform (FFT) algorithm ([Cooley and Tukey, 1965]).

A *spectrogram* (Figure 2.4) shows the spectral variations of a signal over time. Typically it is a $2D$ graph where the horizontal axis represents time and the vertical axis represents the frequencies. The energy of a particular frequency at a particular time is represented by the intensity (on a gray scale) or the color at that particular $2D$ point.

Given a discrete signal $x(n)$, a spectrogram for a particular point in time t is computed through the FFT of the x_i inside a time window centered around t . Aside from the sampling rate of the signal $x(n)$, typical spectrogram parameters include:

- resolution of the spectrogram, which can be parametrized by the size of the time windows and the overlap of adjacent time windows;
- the use of a log scaling of the display of intensity (e.g. dB mapping);
- the use of a log scaling of the frequency axis (usually to come closer to human sensitivity);

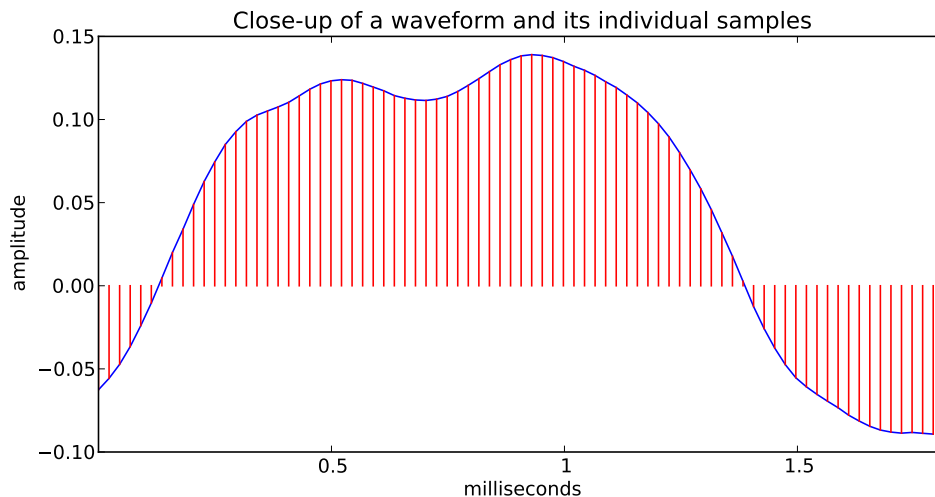


Figure 2.2: Sample audio signal: close-up and individual samples.

- the use of a window function to attenuate the border effects of the Fourier transform (e.g. Hann window or Hamming window, described in [Mitra, 2006]).

We present two broad categories of music audio features: pitch-related features and timbre-related features. Both are derived from a spectrogram, but they capture different aspect of the music. Pitch-related features give an indication of what notes are being played. Timbre deals with the way a “sound sounds”. This definition is explicitly vague as timbre has been described as “the multidimensional waste-basket category psychoacoustician’s for everything that cannot be labeled pitch or loudness” by [McAdams and Bregman, 1979]. Trying to be more specific, timbre-related features attempt to describe characteristics such as the relative position of the harmonics or the amount of distortion in the signal over a short period of time.

2.2 Pitch-related features

The spectrogram gives us the energy associated with pitches at different point in time. Since in music every note is a given pitch (e.g. the reference A4 is often $440Hz$, see Table 2.1), the spectrogram is sufficient to build a simple score extraction algorithm as in [Piszczałski and Galler, 1977].

In music, an octave is the interval between a give note and the one with double its frequency. For instance, the interval A4 ($440Hz$) and A5 ($880Hz$) is an octave. An octave is split into 12

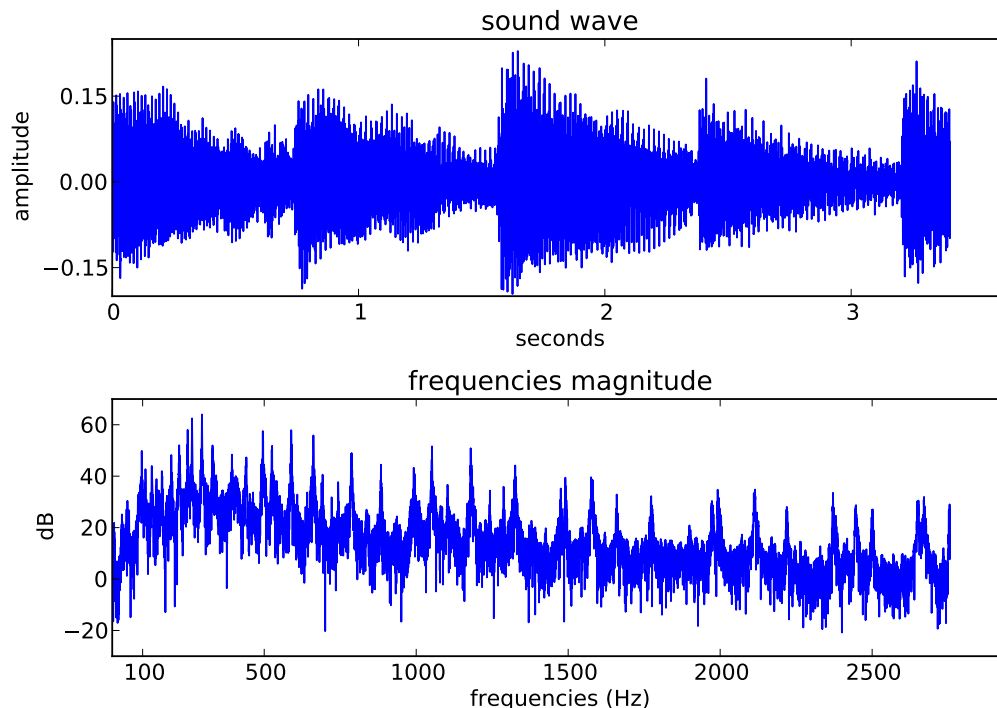


Figure 2.3: Audio signal and corresponding frequency analysis.

For the frequencies, v in dB is defined as $20 * \log_{10}(v)$.

semitones as shown in Table 2.1. If one imagines that the table extends both right and left, each row would be a pitch class. Notes in the same pitch class (e.g. all As) share a similar *quality* which is beyond the scope of the thesis but is the basis of western music theory. Thus, chords are defined by specific pitch classes, not specific notes.

Chroma features were introduced as pitch-class profiles (PCP) by [Fujishima, 1999]. He describes a PCP as a 12 dimension vector which represents the intensities of the 12 semitone pitch classes. Thus the chromagram is similar in spirit to a spectrogram except that pitch content is folded into a single octave of 12 discrete bins, each corresponding to a particular semitone (Figure 2.5).

Many variants have been derived, including HPCP ([Gómez, 2006]), EPCP ([Lee, 2006]) and CENS ([Müller *et al.*, 2005]); an overview can be found in [Jiang *et al.*, 2011]. There is even evidence that chromas can be learned from a simple similarity task [İzmirli and Dannenberg, 2010]. The tuning (i.e. the exact note frequencies) do not have to be known in advance; [Harte, 2005] uses a 36-bin chromagram to first define the note boundaries, then compute the usual 12-bin chromagram.

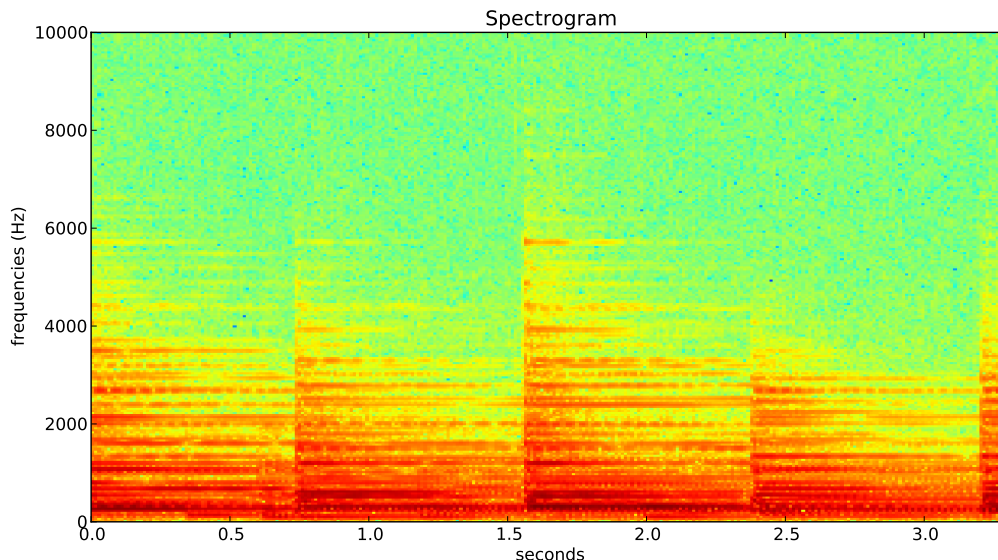


Figure 2.4: Spectrogram.

Computed from signal in Figure 2.1.

We use a Hanning window of size 1024 with an 512 overlap.

Chroma was rapidly seen as a suitable feature to describe the structure of music, for instance Bartsch and Wakefield use it for thumbnailing [Bartsch and Wakefield, 2001]. Other applications include music classification [Ellis, 2007], cover song recognition [Serrà, 2011], audio alignment [Hu *et al.*, 2003], chord recognition [Oudre *et al.*, 2011; Sheh and Ellis, 2003] and segmentation [Weiss and Bello, 2010].

The Echo Nest API ([EchoNest,]) provides an implementation of the chroma feature. The values are normalized between 0 and 1 by dividing by the maximum value at each time sample. Other implementations normalize by the total energy at each time sample, or even do not normalize at all.

2.3 Timbre-related features

We present two features whose goal is to describe the timbre of music, something that is not captured by the pitch-related features.

note	frequency	note	frequency
<i>C</i> 4	261.63	<i>C</i> 5	523.25
<i>C</i> #4	277.18	<i>C</i> #5	554.37
<i>D</i> 4	293.66	<i>D</i> 5	587.33
<i>D</i> #4	311.13	<i>D</i> #5	622.25
<i>E</i> 4	329.63	<i>E</i> 5	659.26
<i>F</i> 4	349.23	<i>F</i> 5	698.46
<i>F</i> #4	369.99	<i>F</i> #5	739.99
<i>G</i> 4	392.00	<i>G</i> 5	783.99
<i>G</i> #4	415.30	<i>G</i> #5	830.61
<i>A</i> 4	440.00	<i>A</i> 5	880.00
<i>A</i> #4	466.16	<i>A</i> #5	932.33
<i>B</i> 4	493.88	<i>B</i> 5	987.77

Table 2.1: 4th and 5th octaves with the note frequencies.

Frequencies in Hz . We assume a tuning with $A4 = 440Hz$.

2.3.1 The Echo Nest timbre features

The Echo Nest API ([EchoNest,]) offers a timbre feature which describes spectrogram patches. It uses an auditory-model-based spectrogram: the frequency axis is scaled to approach the auditory system, the magnitude is mapped to approximate perceived loudness, and an auditory-based smoothing across time is applied. The Echo Nest algorithm discovers the presence of musical events through an onset detector ([Jehan, 2005]). The time interval surrounding an event is called a *segment*. The spectrogram patch of a segment is normalized and the coefficients of the first 12 principal components are computed ([Jehan and DesRoches, 2011]). Principal component analysis will be discussed in Section 3.1.2.2, but in this case it explains the spectrogram patch as a combination of the basis in Figure 2.6. As described in [Jehan and DesRoches, 2011], “the first [dimension of the 12-dimension timbre vector] represents the average loudness of the segment; second emphasizes brightness; third is more closely correlated to the flatness of a sound;” etc.

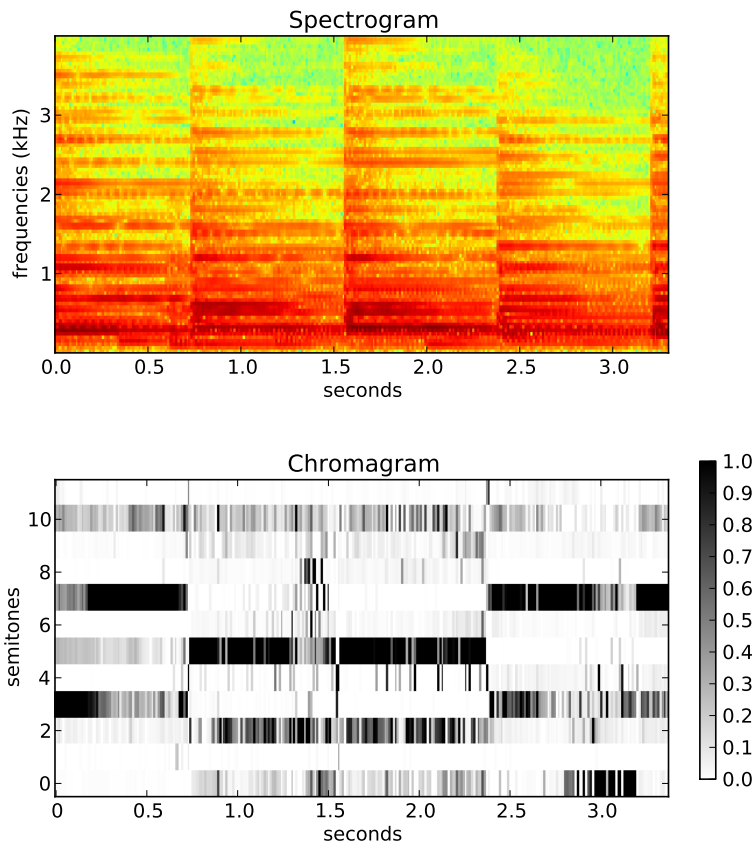


Figure 2.5: Chromagram.

The chromagram (bottom image) computed from the signal in Figure 2.1. The spectrogram is displayed for comparison (e.g., see how the musical event starting around 0.75 seconds is visible in both images).

2.3.2 Mel-frequency cepstrum coefficients

Heavily used for speech recognition, Mel-frequency cepstrum coefficients (MFCC) have been introduced for MIR tasks in [Logan, 2000]. MFCC are used to create a compact representation of a spectral envelope, hence describing timbre with a few values. Given a discrete signal, the MFCC are computed with the following steps over a short window:

1. apply the discrete Fourier transform;
2. compute the magnitude;
3. scale the frequency axis using the Mel scale;

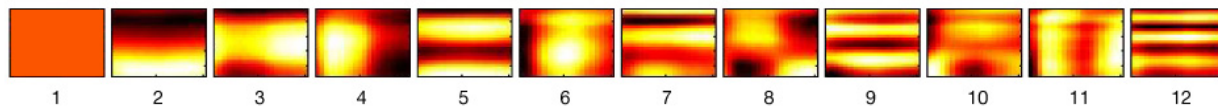


Figure 2.6: Timbre basis from The Echo Nest analyzer.

Those basis are obtained by PCA on a collection of normalized spectrogram patches. Image taken from [Jehan and DesRoches, 2011].

4. take the logs of the values for each Mel frequency;
5. apply the inverse discrete Fourier transform.

The Mel scale ([Stevens and Volkman, 1940]) is based on a mapping between actual frequency and perceived pitch as apparently the human auditory system does not perceive pitch in a linear manner. The mapping is approximately linear below 1kHz and logarithmic above ([Logan, 2000]).

As for the spectrogram, we can compute successive MFCCs and stack them horizontally (Figure 2.7). This feature is powerful as it describes the relationship between the frequencies without knowing about the actual pitches. It is compact because, through the inverse DFT, it describes the spectral envelope as a sum of orthogonal sinusoidal basis (the inverse DFT is very similar to the DFT itself).

2.4 Beat-alignment

We will often mention beat-aligned features in this work. This implies that we:

- ran a beat tracking algorithm;
- performed a linear scaling of the features to get one value (or vector) per beat.

Consider a chromagram computed for every 50 ms starting at 0, and a particular beat b ranging from 200 to 400 ms. The chroma vector for this beat will be the average chroma vector from intervals 200 – 250 ms, ..., 350 – 400 ms. When working with non-linear time frames, we use a weighted average. An example is shown later in this work, in Figure 8.2.

Beat-alignment often adds robustness as it is a natural time frame for music. Think about the same song played twice, but with different tempos. Computing features using a fixed time window

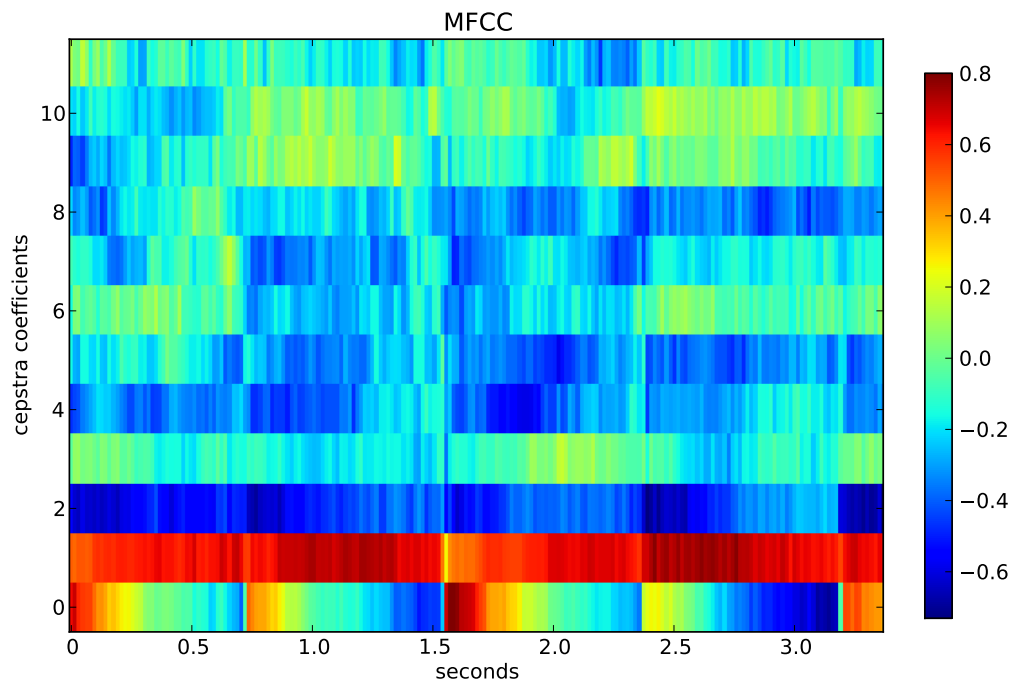


Figure 2.7: Mel-frequency cepstrum coefficients.

Computed from the signal in Figure 2.1.

We use a window size of length 32 ms with an overlap of 16 ms. We subtracted the average level of the first coefficient (c_0) (equivalent to scaling the waveform) to get cepstra with a more equal range of values between c_0 and the rest.

(e.g., 50 ms) will give two very different feature sets. However, once beat-aligned, the features from the two performances should be virtually the same.

2.5 Features derived from The Echo Nest features

We briefly mention the recent work by [Schindler and Rauber, 2012]. Working with the features provided by The Echo Nest is difficult because of the varying timescale (based on “musical event”). The authors provide methods to aggregate those event-level features over time. The newly derived features seem to be useful for different music classification tasks, e.g. genre recognition. This work on The Echo Nest features is a result of the growing popularity of the Million Song Dataset (Part II).

Chapter 3

Machine learning

This chapter is intended to briefly cover aspects of machine learning that will be used throughout this thesis. It is expected to be a reminder, the reader who needs a more complete introduction is referred to books such as [Bishop, 2006; Duda *et al.*, 2000; Mitchell, 1997; Russell and Norvig, 2009].

Machine learning tries to uncover patterns in data and use them in some meaningful way, which can be described as *learning*. As T. Mitchell [Mitchell, 1997] puts it: “a computer program is said to learn from experience E with respect to some class of tasks T and performance measure P , if its performance at tasks in T , as measured by P , improves with experience E ”. A specific example from [Duda *et al.*, 2000] is classifying two types of fish based on their length and skin lightness. A number of correctly classified examples are provided. The task T is (without loss of generality) to assign to each fish $x_i \in X$ a label in $\{0, 1\}$. Each x_i is a two-dimensional vector containing the values for length and lightness of one particular fish. The performance measure P can simply be the number of misclassified fish. Finding a proper function $F : X \rightarrow \{0, 1\}$ is the core machine learning problem. The experience E is to present an algorithm with some pairs of (fish, label), called the “ground truth”, in order to create F .

In the previous examples, the vectors x_i contains the *features*, the descriptors of the studied object (the fish). Here those features are length and lightness. Most often we deal with numerical features, but it could be anything in theory, for instance a label naming the lake where the fish were found. Some common music-related features were described in Chapter 2.

Modeling the problem through statistics, we assume that all x_i in the population X come from

a given, fixed distribution. Understanding this distribution can be used to make predictions. For instance, discovering that the first type of fish tends to be smaller than some threshold t , while the other type tends to be longer, is useful since yet unseen fish should have the same property. Note that as learning is different than remembering, the goal is to create a function F that works for previously unseen examples. This concept is called *generalization* and is fundamental in machine learning. It also explains why F is often called a *predictor*.

The chapter is organized as follows, first we present some general techniques to process features. Then, we present the category of supervised learning tasks, to which the fish example above belongs. This sections includes basic algorithms and methods to measure the generalization of a predictor. Finally, we discuss unsupervised tasks, mostly clustering.

3.1 Feature representation

We assume that we have a population $X = \{x_i\}$, each x_i being the d -dimensional vector of features for the i th element of X . Furthermore, we assume that these features are real numbers, which is the case for all data throughout this work.

We present two general techniques that have proven useful in various settings: vector quantization and dimensionality reduction. Usually, those methods are used to reduce the noise in the data and reduce the size of high-dimensional feature vectors (e.g. $d > 100$). The latter is of particular interest in this work.

3.1.1 Vector quantization

Shannon defines *source coding* as the mapping of symbols from an information source to a sequence of alphabet [Shannon, 1948]. If the source symbols can not be recovered exactly, it is called *lossy source coding*.

In our case, we have a dataset of feature vectors x_i , each of dimension d , that would be the symbols in Shannon's definition. We also assume we have a *codebook* of M d -dimensional vectors, the alphabet. Vector quantization [Gersho and Gray, 1991] replaces each feature vector by its closest *codeword* in the codebook. The distance between a feature vector and a codeword is often the Euclidean distance between the two vectors, but it could be any other measure, e.g. Kullback-

Leibler divergence [Kullback and Leibler, 1951] if the features come from a distribution. One method to build the codebook is to use the centroids of the clusters found through k -means, see Subsection 3.3.1 below.

We skip the information theory analysis of the method, such as the trade-offs between level of compression and codebook size. In our work, the interest in vector quantization is twofold. First, codewords can be seen as templates, and replacing a feature vector by a template removes variations that might be irrelevant to the task at hand. Second, note that we could use the codeword index as a feature instead of the original feature vector, replacing d values by one. Therefore, in the example above, we could discover that all fish whose features are mapped onto the first codeword are of the first type.

3.1.2 Dimensionality reduction

Having additional features to describe items should never hurt performance, as one can always ignore those additional features. Unfortunately, the reality is a little more tricky. A lot of algorithms depend on measuring a distance between feature vectors (Euclidean distance for instance), and these measure often behave in surprising ways when the dimension d is large. This phenomenon has been dubbed the *curse of dimensionality* and we try to illustrate it below.

A solution is to apply a linear transformation to the original feature space where the new space is of a lesser dimension. For instance, in Chapter 9, we project the original 900-dimensional features into a 50-dimensional space. We present methods on how to do this while preserving the information in the data. These are essential tools for anyone working with large-scale highly-dimensional datasets.

3.1.2.1 Curse of dimensionality

The curse of dimensionality [Bellman, 1961] refers to the severe difficulty that can arise in spaces of many dimensions [Bishop, 2006]. For instance, neighboring data points tend to get “arbitrarily far” from one another.

One way to gain some understanding about the problems of multiple dimensions is to consider a unit sphere in a hypercube¹.

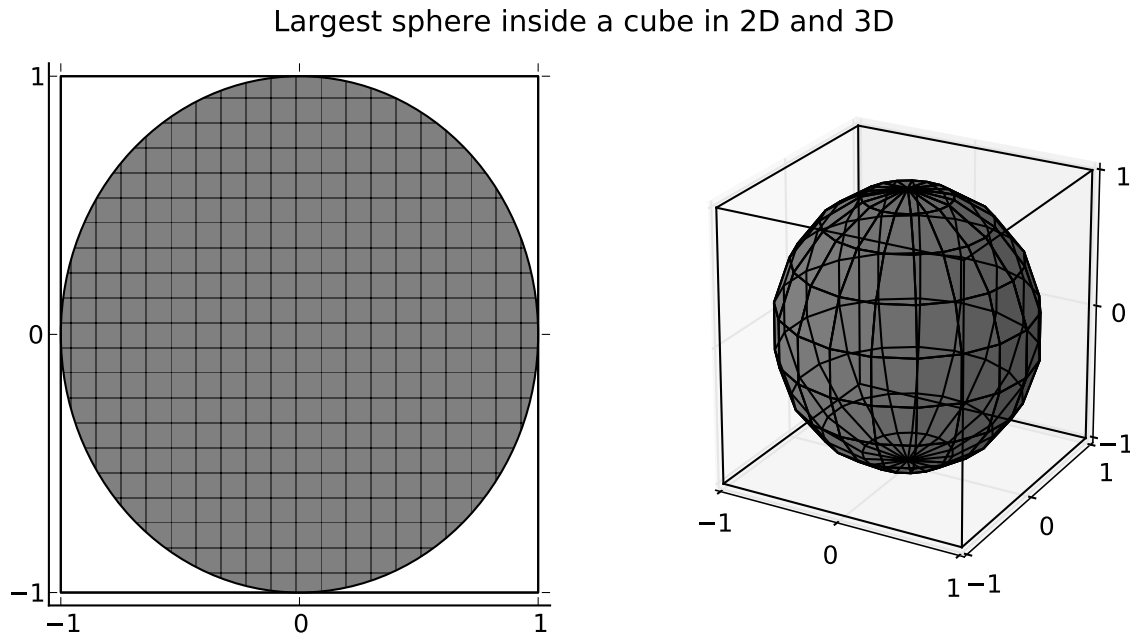


Figure 3.1: Volume of a hypercube and its largest contained sphere in 2D and 3D.

The volume of a unit sphere when the dimension d is even (similar formula for d odd) is

$$V_s^d = \frac{\pi^{d/2}}{(d/2)!}$$

For any dimension, the unit sphere contains all points within a radius 1 of the center. The volume of the smallest hypercube containing that sphere (in 2 dimensions, a square with sides of length 2) is $V_c^d = 2^d$. The ratio V_s^d/V_c^d tends to 0 as $d \rightarrow \infty$, meaning that less and less of the volume of the cube is inside the sphere as d increases. In our case, assume that the center of the sphere (and the cube) is a feature vector, and all other data points are inside the cube. If they are uniformly distributed, the chance to have a data point inside the sphere tends to 0. If we consider the nearest neighbor algorithm and a sample centered at the origin, what tends to 0 is the chance of having a neighbor whose distance is less than 1. Thus, with large d , all data points tend to be far from each other, and comparing two points based on their feature distance loses meaning.

¹Unable to find the origin of this example, we mention this blog post that presents it <http://yaroslavvb.blogspot.com/2006/05/curse-of-dimensionality-and-intuition.html>

3.1.2.2 Principal component analysis

Principal component analysis [Pearson, 1901], or PCA, is an algorithm that applies an orthogonal transformation to a set of data points. Thus, the points are then expressed as a combination of linearly uncorrelated variables called *principal components*. The orthogonal transformation, i.e. the list of principal components, is chosen such as the first K components account for the largest possible variance in the data. Put another way, there exists no orthogonal transformation of K components that yields a lower reconstruction error than the top K principal components of PCA.

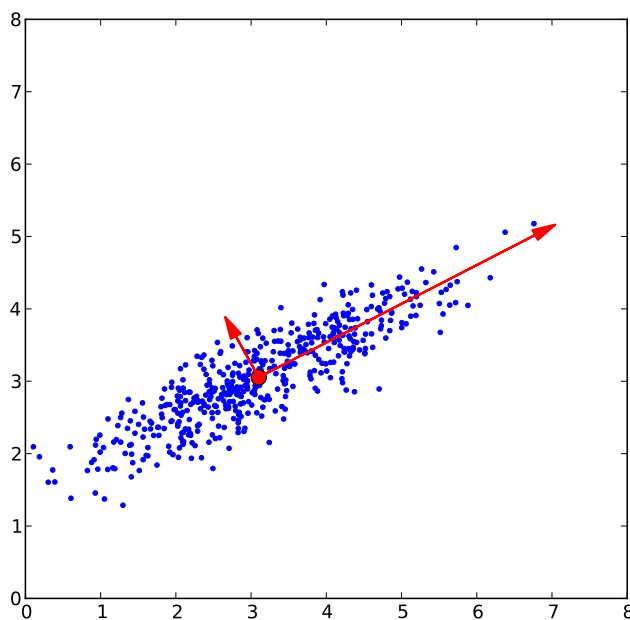


Figure 3.2: Principal components of a 2D Gaussian distribution.

The PCA algorithm works as follow. We have a set of N data points forming X , a $N \times d$ real matrix, each row being a d -dimensional feature vector.

1. Center the data: compute the empirical mean u (d -dimensional vector) and remove it from all rows of X , thus building X_0 .
2. Compute the covariance matrix: $C = \mathbf{E}[X_0 \cdot X_0^T] = 1/N[X_0 \cdot X_0^T]$.
3. Compute eigenvectors and eigenvalues: $V^{-1}CV = D$. The columns of V are the eigenvectors.
4. Rearrange eigenvectors in order of decreasing eigenvalues, V becomes W .

5. Project the data: $Y = W^T \cdot X_0$. To reduce the dimension of Y to $K < d$, use only the first K columns of W .

The method above find linearly uncorrelated components that maximize the variance (steps 2 – 3). For the proof that this method also minimizes the reconstruction error, please refer to [Bishop, 2006].

Note that we skipped preprocessing of the values. For instance, values in the columns of X could be replaced by their z -score, or linearly transformed so that the values fall in $(-1, +1)$.

3.1.2.3 Random projections

There have been many developments related to the previous method, for example the introduction of non-linearities in PCA through the use of kernels [Mika *et al.*, 1999]. At the same time, in the direction of less complexity, random projection methods have received a lot of attention lately.

The Johnson-Lindenstrauss lemma [Johnson and Lindenstrauss, 1984] shows that a set of N points in high dimensional Euclidean space can be mapped down into an $O(\log N/\sigma^2)$ dimensional Euclidean space such that the distance between any two points changes by only a factor of $O(1 \pm \sigma)$. A proof can be found in [Dasgupta and Gupta, 1999]. This means that we can perform approximate nearest neighbor search in a space of smaller dimension.

The random projection matrix can be extremely simple, for instance this one from [Achlioptas, 2001]:

$$r_{ij} = \sqrt{3} \times \begin{cases} +1 & \text{with probability} & 1/6 \\ 0 & \text{..} & 2/3 \\ -1 & \text{..} & 1/6 \end{cases}$$

Random projections are the backbone of the locality-sensitive hashing (LSH) algorithm [Indyk and Motwani, 1998].

3.2 Supervised learning

We are in a supervised learning setting when we have access to a set of objects with their features x_i and a corresponding set of labels y_i . The goal is to learn a predictor $F(x_i) = \hat{y}_i$ that minimizes some error measure between y_i and \hat{y}_i for previously unseen x_i data points. This setting is opposed

to unsupervised learning where the labels are not known in advance and the goal is to organize the data in a suitable manner, see Section 3.3.

The two most common tasks are *classification*, where the labels are a finite discrete set, and *regression*, where the labels are continuous values. Note that there are some tasks that do not fall naturally in those two categories, for instance learning to rank (e.g. for recommendations, more on this in Section 6.2.3) or to organize objects in a sequence (e.g. creating a playlist as in [Maillet *et al.*, 2009]). Also, some data points can belong to many classes, possibly with different level of membership. Such an example is the task of applying tags (i.e. keywords) to songs [Bertin-Mahieux *et al.*, 2010a].

In this section we mention two very simple and often used classification algorithms: *decision trees* and *k-nearest neighbors*. The goal is to illustrate a typical machine learning approach and to explain how we measure generalization using held-out data. The decision trees are not actually used in this work, but it is important to present a *model-based* classifier, i.e. a classifier that does not remember samples once the training is done. This is the opposite of a *memory-based* classifier such as *k-nearest neighbors*.

3.2.1 Decision trees

In Figure 3.3 we see an extremely simple decision mechanism for the fish example: if the length of a fish is less than some threshold t , it is a fish of type A . Otherwise, it is of type B . This structured is often called a *decision stump*. It looks at only one feature, and in the binary case, it can assign up to two classes.

It is easy to imagine a deeper structure. If the length is greater than t , instead of assigning a class, we could ask an additional question. For instance, is the length smaller or greater than a second threshold t_2 , or is the luminosity smaller or greater than t_3 . Such a structure is called a *decision tree*. The decision stump in Figure 3.3 is the simplest version of it. The procedure to find the “best” questions to ask based on some data is not trivial, but many solutions have been proposed, for instance the C4.5 algorithm [Quinlan, 1993].

The important part is that, given a built decision tree, it is easy to classify a new data point. One simply asks the questions starting at the top of the tree (the *root* node), follow the path downward based on the answers, and the class is given by the last node reached (a *leaf* node).

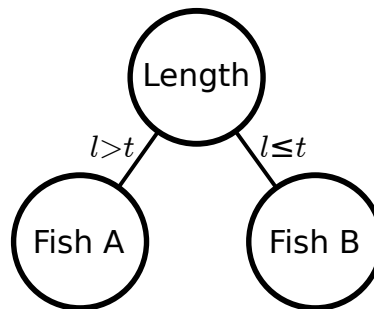


Figure 3.3: Simple tree structure (decision stump).

If the length is larger than a given threshold t , we believe it is fish A .

Otherwise, it is fish B .

Decision trees are one of the oldest machine learning algorithms, and are still widely popular today.

3.2.2 Nearest neighbors

The k -nearest neighbors (k -NN) algorithm relies on the assumption that “nearby” data points are similar. Given a set of known data points (features and labels), we can classify a new data point based on the labels of its k nearest neighbors. In Figure 3.4, we use a 1-NN algorithm, meaning we look only at the closest neighbor. In the illustration, the unknown data point would be label as a *red circle* instead of a *blue square*.

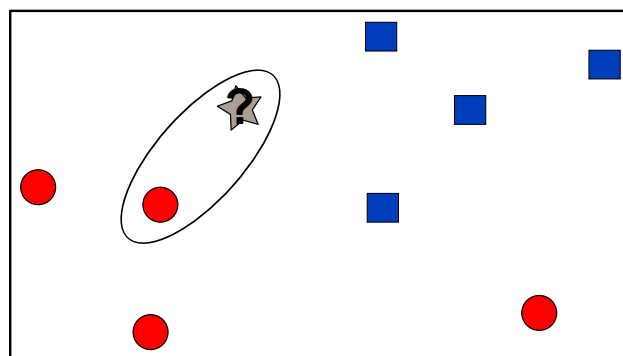


Figure 3.4: 1-NN example with two classes in a 2D space.

Note that we skip the discussion about finding a proper space where neighbors do share the same label. What is important is that k -NN is simple to implement and to understand. It is a good candidate as the first algorithm to try on any new problem. It does require a certain amount

of memory as we need to keep track of all the known data points and their labels. Finding the nearest neighbors can also be expensive if the dimension is high or if the dataset is large.

3.2.3 Unbiased estimates

As we mentioned earlier, the goal of machine learning is to make predictions about unseen data. Therefore, we can not evaluate a predictor on data that was used to create this predictor. To illustrate this, think of the 1-NN algorithm in Figure 3.4. Every known point would be the closest neighbor to itself, thus the algorithm would assign the right label every time. Of course, that does not tell us anything about how often we would classify unseen data correctly. In general, if an algorithm performs better on data it was trained on, it is *biased* towards it. This effect is called *overfitting*.

In order not to be overestimate the performance of an algorithm, we set aside data that we only use to test the algorithm. For example, if we have 1,000 data points with labels, we can use 800 of them to train the algorithm, and 200 to test its generalization performance. Unsurprisingly, we call these two collections the *train set* and *test set*.

Most algorithms have meta-parameters that are often set by humans. For instance, the k in k -NN, or the learning mechanism and the maximum depth of decision trees. The best performing one on the training set might only be the most biased. The same problem arises if we use the test set. Therefore, we usually keep a third subset of the data called the *validation set*. The typical machine learning work flow would be:

1. train predictors on the training set with different meta-parameters;
2. choose the best performing predictor on the validation set;
3. measure its generalization error on the test set.

3.3 Unsupervised learning

Sometimes we have to make sense of data without any labels, hence the term *unsupervised*. There is no ground truth to tell the algorithm if it is mistaken. The most common task is clustering, i.e. grouping items that appear similar. Another application is manifold learning, where we find and

characterize the subspace where (most of) the data actually lie. An example taken from [Weinberger and Saul, 2004] is the “Swiss roll” shown in Figure 3.5. The data in the 3D space actually lies on a 2D surface.

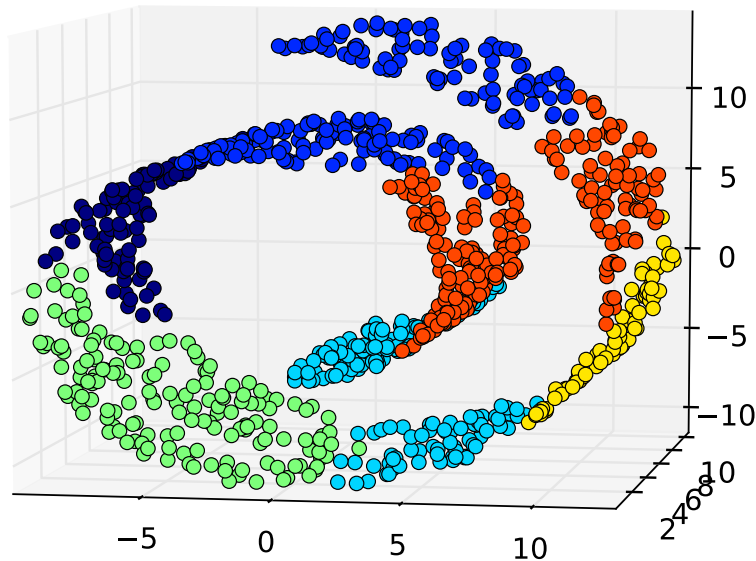


Figure 3.5: Swiss roll manifold.

Reproduced from [Weinberger and Saul, 2004] using code from

http://scikit-learn.github.com/scikit-learn.org/0.9/auto_examples/cluster/plot_ward_structured_vs_unstructured.html.

3.3.1 Clustering

“Cluster Analysis could be considered a field all its own, part art form, part scientific undertaking. One seeks to arrange an unordered collection of objects in a fashion so that nearby objects are similar. There are many ways to do this, serving many distinct purposes, and so no unique best way” [Donoho, 2000].

Since summarizing such a field is beyond the scope of this chapter, we will focus on K -means [MacQueen, 1967], the most famous clustering algorithm. Most of what follows is derived from [Bishop, 2006]. Given a data set of N points $\{x_i\}$, each a d -dimensional vector, we want a set of K vectors μ_k that represent cluster centroids. K -means is looking for μ_k that minimize the distance

J between each data points and its closest centroid:

$$J = \sum_{n_1}^N \sum_{k=1}^k r_{nk} \|x_n - \mu_k\|^2$$

where

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg \min_j \|x_n - \mu_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

To this end, after initializing the μ_k , we repeat the following two steps until convergence.

1. Assign each data point to its closest centroid.
2. Update μ_k as the mean of all data points assigned to them.

This is a simple example of an expectation maximization (EM) algorithm. In the first step, we fix μ_k and minimize J by finding the optimal r_{kn} . In the second step, we fix r_{kn} and minimize J by finding the optimal μ_k . J can only improve at each step so it has to converge to a local minimum.

We ignored the question of initializing the centroids. One can chose random points from the dataset or sample the space covered by the x_i . Some initialization methods has been developed to try to speed up K -means, most notably [Arthur and Vassilvitskii, 2007].

Finally, cluster centroids can form a codebook for vector quantization, described in Section 3.1.1.

Part II

The Million Song Dataset

Chapter 4

Introduction to the MSD

The Million Song Dataset (MSD) is a resource for researchers that we released in collaboration with The Echo Nest¹, a company from Massachusetts. This chapter explains the difficulty of sharing music data among researchers and presents some existing datasets. Chapter 5 details the creation of the dataset and its content. Finally, Chapter 6 gives some insight into the possible uses of the dataset. It includes work from other MIR laboratories that have already taken advantage of the MSD in their publications.

4.1 Problem statement

Preparing and releasing a research dataset requires a surprising amount of work and time, and the MSD is no exception. The details of the dataset will be presented in Chapter 5, but let's first explain what was missing to MIR researchers to justify a new resource. In particular, we focus on the problem of distributing large-scale music content.

Two characteristics of the music copyright laws are their vagueness and their local differences. For instance, the concept of *fair use* in the United States versus *fair dealing* in Canada, or the length of the protection awarded to the copyright holders [Berti, 2009]. From an academic viewpoint, it means that no one can share copyrighted audio to fellow researchers worldwide without incurring some risk. This is a major problem when considering the reproducibility of an experiment. One can run an algorithm on one's own personal music collection and share the code, but not the data.

¹<http://the.echonest.com>

Therefore, any dataset of interest has to be freely available.

One could object that an easy workaround for the copyright issues is simply to buy the music. Assuming the price of \$1 per song, we can imagine an academic laboratory buying a few hundred or thousand songs. Would such an amount of audio be enough? For many applications, we argue that it is not. Commercial application such as Spotify² report a listenable collection of 16 million songs. Other companies such as Rdio³ and Last.fm⁴ have a catalog size of the same order of magnitude, although the exact number is not disclosed. Thus an algorithm coming out of academia and working on a few thousand songs might simply not work on a commercial-size database. Of course, buying a million songs is not likely an option for any non-profit organization.

The goal is to have access to a large collection of music data that can be distributed and that is comparable to commercial catalogs in size. In the next section, we present existing resources and explain why they do not fulfill these requirements.

4.2 Other resources

The most famous MIR collection might be the GTZAN genre dataset [Tzanetakis and Cook, 2002]. It contains 1,000 audio excerpts equally divided into 10 genres. This dataset has been used numerous times for evaluating genre recognition algorithms. The audio is relatively easy to access as researchers have been sharing it privately. Due to the small size and the sources of audio (a mixture including radio recorded through a microphone and CDs from a university library), this data most likely falls into the *fair use* category and the risk of sharing it is low. The problems with GTZAN are its small size, the restrictive nature of genre recognition as a task (e.g. a song can only belong to one genre), and the errors in its creation, for instance duplicate songs⁵.

The Real World Computing (RWC) Music Database [Goto *et al.*, 2002; Goto *et al.*, 2003] contains six original components: the Popular Music Database (100 pieces), the Royalty-Free Music Database (15 pieces), the Classical Music Database (50 pieces), the Jazz Music Database (50

²<http://www.spotify.com>

³<http://www.rdio.com>

⁴<http://www.last.fm>

⁵http://media.aau.dk/null_space_pursuits/2012/01/faults-in-the-tzanetakis-music-genre-dataset.html

pieces), the Music Genre Database (100 pieces), and the Musical Instrument Sound Database (50 instruments) [Goto, 2004]. It is distributed via a set of CDs and DVDs, the cost to researchers is approximately \$500⁶. While this database provides access to a wide range of music-related audio and royalty-free music, it remains small in size.

The USPOP dataset [Berenzweig *et al.*, 2004] contains audio features for 8,752 pop songs chosen from popular artists on a file sharing network. Audio features (MFCCs) are freely distributed⁷. The argument is that, since MFCCs are not sufficient to reconstruct the audio in any meaningful way, they fall into the *fair use* category and can be used for research. The USPOP is a great resource to study “recent” (as of 2002) popular western music.

The CAL-500 dataset [Turnbull *et al.*, 2007; Turnbull *et al.*, 2008] is a collection of 500 songs accompanied by 1,700 human annotations. This is mostly used for the task of music tagging [Bertin-Mahieux *et al.*, 2010a] and the features are distributed for free⁸ in a similar way to USPOP. The same group from San Diego later released CAL-10K [Tingle *et al.*, 2010], a collection of 10,870 songs with 475 *acoustic tags* and 153 *genre tags* from Pandora’s Music Genome Project. The features from The Echo Nest [EchoNest,] are provided as well. These two datasets had a major impact in shifting the community’s attention from genre recognition to automatic tagging, a more realistic task. Unfortunately, for simplicity and comparison purposes, most published work use the rather small CAL-500 instead of CAL-10K. The annotations in CAL-500 are also problematic, as some songs were only tagged by 3 humans who can greatly disagree [Bertin-Mahieux *et al.*, 2008].

Magnatagatune [Law and von Ahn, 2009] is one of the most complete MIR research datasets available. It contains⁹: 1) 25,863 sound clips from Magnatune¹⁰, freely shareable because of their license, 2) human annotations from the online game TagATune [Law *et al.*, 2007], and 3) precomputed features from The Echo Nest [EchoNest,]. Unfortunately, this dataset has been less used than CAL-500, maybe because the songs from Magnatune are mostly unknown. Size could be another deterrent.

⁶<http://staff.aist.go.jp/m.goto/RWC-MDB/>

⁷<http://labrosa.ee.columbia.edu/projects/musicsim/uspop2002.html>

⁸<http://cosmal.ucsd.edu/cal/projects/AnnRet/>

⁹<http://tagatune.org/Magnatagatune.html>

¹⁰<http://magnatune.com/>

The Music Information Retrieval Evaluation eXchange (MIREX) is an annual evaluation campaign for MIR algorithms [Downie, 2008]. Tasks vary every year depending on the interest from the community, but they usually include genre classification, onset detection, audio tag classification, and cover song identification among others¹¹. MIREX is an excellent way to compare an algorithm with the state-of-the-art from other MIR laboratories. Unfortunately, these evaluations usually deal with a few hundred or a few thousand songs.

MusiCLEF [Orio *et al.*, 2011] is a benchmark activity resembling MIREX. The 2012 edition focus on music tagging, participants have a remote access to 975 songs with metadata, audio and precomputed audio features, related web pages, collaborative tags from Last.fm, and ground truth tags from human annotators¹². Annotations and precomputed features are usually released after the end of the evaluation.

Some other recent resources (some discovered through Mark Levy's blog¹³) include:

1. 45,000 classical scores scanned using optical mark recognition [Viro, 2011];
2. the Billboard project¹⁴, it contains chord annotations for 1,000 pop songs [Burgoyne *et al.*, 2011];
3. the structural annotation for 500 songs from the RWC collection, the Quaero project¹⁵, and the Eurovision contest [Bimbot *et al.*, 2011];
4. the SALAMI project¹⁶, it provides 2400 structural annotations of nearly 1400 musical recordings [Smith *et al.*, 2011].
5. 56,000 crowd-source mood annotations for TV theme tunes by the BBC¹⁷;

¹¹http://www.music-ir.org/mirex/wiki/MIREX_HOME

¹²<http://www.multimediaeval.org/mediaeval2012/newtasks/music2012/>

¹³<http://mir-in-action.blogspot.com/2011/11/ismir-2011-data-data-and-more-data.html>

¹⁴<http://ddmal.music.mcgill.ca/billboard>

¹⁵<http://www.quaero.org>

¹⁶<http://ddmal.music.mcgill.ca/salami>

¹⁷<http://www.musicalmoods.org/>

6. 40,000 tempo labels and bpm estimates crowd-sourced from 2,000 Last.fm listeners [Levy, 2011].

Table 4.1 summarizes most of these resources based on their size. The comparison is often difficult to make, for instance between a music genre collection and a set of structural annotations. Nevertheless, it should give an idea of the order of magnitude in size of the datasets that are actively investigated.

dataset	# songs / samples	audio available
RWC	465	Yes
Bimbot et al.	500	No
CAL-500	502	No
GZTAN genre	1,000	Yes
Billboard	1,000	No
MusiCLEF	1,355	Yes
SALAMI	1,400	No
USPOP	8,752	No
CAL-10K	10,870	No
Magnatagatune	25,863	Yes
MSD	1,000,000	No

Table 4.1: Size comparison between some other datasets.

The datasets are described in Section 4.2, except the MSD which is the subject of the next Chapter. Remember that the comparison is often difficult to make, for instance between a music genre collection and a set of structural annotations.

Finally, the OMRAS2 project [Fazekas *et al.*, 2010] aims at bridging the gap between MIR and the *semantic web*. The core resource, DBtune¹⁸ “hosts a number of servers, providing access to music-related structured data, in a Linked Data fashion”. The project is a collection of resources that are varied and difficult to describe, ranging from Musicbrainz [Musicbrainz, 2012] metadata to play counts on Last.fm. OMRAS2 provides tools to extract, share and organize audio features,

¹⁸<http://dbtune.org/>

but not to share the actual audio. OMRAS2 is an interesting platform for researchers to work on large scale data without the audio, but note that the content can change at any point in time.

4.3 Goals for the MSD

The review of the datasets available to the MIR community helps us highlight the goals we set ourselves for the MSD:

- large-scale, comparable to commercial databases;
- freely available to any researcher, no fee attached;
- fixed, so experiments can be reproduced;
- easy to get started to encourage its adoption.

In the following chapters we explain how the MSD was built and what tasks can be performed with it. The goals we set ourselves are an answer to a lack of large-scale applications developed by the MIR community.

4.4 Audio

In order to provide a commercial-scale database that is freely available, we basically could not, under the current legal climate, provide the actual audio. Providing derived features was the only choice. Additionally, the audio for the whole MSD would correspond to an additional 30 TB of data¹⁹, which would be quite impractical to distribute and difficult to process.

However, it has to be recognized that since the original audio is not available, a large amount of research - the development of novel front-end signal representations - is not possible with the MSD. This issue can be circumvented in a few cases using data we present in Section 5.2.7. A group of researchers has downloaded snippets for all the million song and can provide some access to the audio.

¹⁹Counting 30 MB per song in FLAC format times one million.

Chapter 5

Creation of the dataset

We start with a brief history of the Million Song Dataset and present how the collaboration between LabROSA¹ at Columbia University and The Echo Nest² came to existence. Then, we introduce the data sources for the MSD and how we selected a million tracks. The starting point is The Echo Nest API [EchoNest,] which we used to select the tracks, obtain their metadata, audio features, and more. Based on that core collection, we added data from other sources. Those include cover songs, lyrics, tags, song similarities and user data.

5.1 History

The idea for the Million Song Dataset arose a couple of years ago in the minds of D. Ellis and B. Whitman while they were discussing ideas for a proposal to the US National Science Foundation's GOALI (Grant Opportunities for Academic Liaison with Industry) program. They wanted an idea that would not be possible without academic-industrial collaboration, and that would appeal to the NSF as contributing to scientific progress [Bertin-Mahieux *et al.*, 2011a].

One of the long-standing criticisms of academic music information research from colleagues in the commercial sphere is that the ideas and techniques developed in academia are simply not practical for real services, which must offer hundreds of thousands of tracks at a minimum. But, as academics, how can we develop scalable algorithms without the large-scale datasets to try them

¹<http://labrosa.ee.columbia.edu/>

²<http://the.echonest.com/>

on? The idea of a “million song dataset” started as a flippant suggestion of what it would take to solve this problem. But the idea stuck – not only in the form of developing a very large, common dataset, but even in the specific scale of one million tracks.

5.2 Creation and datasets

5.2.1 The Echo Nest API

The Echo Nest³ is a “music intelligence” company that enables developers to create music-related applications. They crawl the web and get data from different partners to provide information such as artist biographies, similar artists, discographies, etc. They also have access to audio in order to compute features that they can use for content-based similarity. Their most common business model is to release their data via their API [EchoNest,] to the creator of an app in exchange of a percentage of the profits.

The Echo Nest API can be queried for free by researchers and developers (with usage limits). Below A typical API JSON response from The Echo Nest about a particular track by *Weezer* with a duration of 243 seconds.

```
{
  "response": {
    "status": {
      "code": 0,
      "message": "Success",
      "version": "4.2"
    },
    "track": {
      "artist": "Weezer",
      "audio_md5": "e16bde82eaecd13bde9261b2710aa991",
      "audio_summary": {
        "analysis_url": "https://echonest-analysis.s3.amazo...",
        "danceability": 0.5164314670162907,
```

³<http://the.echonest.com>

```
"duration": 243.64363,  
...
```

The core of the MSD is extracted from that API. Most fields listed in Appendix A originate from it. The notable exceptions are *year* and *artist_mbtags* from Musicbrainz⁴. The Echo Nest API was used to obtain artist, album and track names. For all tracks we gathered audio features (see Chapter 2 and The Echo Nest’s co-founder thesis: [Jehan, 2005]) and a popularity measure. We added similar artists, artist tags, and artist location information. Finally, we queried for identifiers from additional sources, e.g. we have the 7digital⁵ identifiers for all tracks.

This represents the core of the MSD because it is the reference point for what songs are included in the collection. Complementary datasets were all created using the meta data (usually artist name and song title) from The Echo Nest API. Section 5.2.2 explains how we actually selected a million tracks from this API.

5.2.2 Song selection

Choosing a million songs is surprisingly challenging. Our selection was made by:

1. getting the most “familiar” artists according to The Echo Nest, then downloading as many songs as possible from each of them;
2. getting the 200 top terms⁶ from The Echo Nest, then using each term as a descriptor to find 100 artists, then downloading as many of their songs as possible;
3. getting the songs and artists from the CAL500 dataset;
4. getting “extreme” songs from The Echo Nest search parameters, e.g. songs with highest energy, lowest energy, tempo, etc;
5. having a random walk along the similar artists links starting from the 100 most familiar artists.

⁴<http://musicbrainz.org/>

⁵<http://www.7digital.com/>

⁶a *term* is a *tag* in The Echo Nest world.

The number of songs was approximately 8,950 after step 1), step 3) added around 15,000 songs, and we had around 500,000 songs before starting step 5). The code used for creating the dataset is freely available⁷. The MSD contains 44,745 unique artists, meaning each artist has an average of 22 songs.

5.2.3 Cover songs

We partnered with Second Hand Songs⁸, a community-driven database of cover songs (Figure 5.1), to provide the SecondHandSong dataset⁹ (SHS). It identifies 18,196 cover songs in the MSD grouped into 854 works (or cliques). For comparison, the MIREX 2010 Cover Song evaluation used 869 cover songs¹⁰. Since most of the work on cover recognition has used variants of the chroma features which are included in the MSD (pitches), it is now the largest evaluation set for this task. We provide a train / test split so the results are easily comparable between publications, the train set contains 18,196 covers and the test set 5,854 (the sets are disjointed). We will talk more about cover song in Part III.

5.2.4 Lyrics

In partnership with musiXmatch¹¹, we have released the musiXmatch dataset¹², a collection of lyrics from 237,662 tracks of the MSD. The lyrics come in a bag-of-words format and are stemmed, partly for copyright reasons. As a possible application, mood prediction from lyrics (a recently-popular topic) could be investigated with this data. Another example will be shown in Section 6.1.1.

⁷<https://github.com/tb2332/MSongsDB/tree/master/PythonSrc/DatasetCreation>

⁸<http://www.secondhandsongs.com/>

⁹<http://labrosa.ee.columbia.edu/millionsong/secondhand>

¹⁰ MIREX used two collections: the first one contains 330 cover songs out of 1,000 audio files, and the second collection contains 539 cover songs (with no additional files). Since the collections were not merged, it gives a total of 869 queries, each query done on a collection of at most 1,000 songs.

¹¹<http://musixmatch.com/>

¹²<http://labrosa.ee.columbia.edu/millionsong/musixmatch>

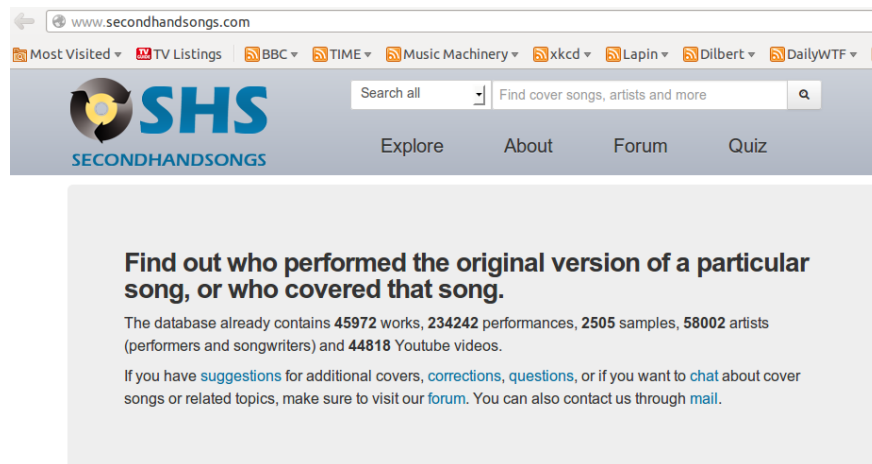


Figure 5.1: Screen shot of the Second Hand Songs website.

Taken on November 29th, 2012.

5.2.5 Tags and song similarity

Through an association with Last.fm¹³, we obtained song-level tags and song-to-song similarity measures. This is known as the Last.fm dataset¹⁴. Last.fm matched 943,347 MSD tracks to their collection. They gave us 8,598,630 unique (track, tag) pairs and 56,506,688 unique track-to-track similarity values. Put another way, 505,216 MSD tracks have at least one tag, and 584,897 MSD tracks have at least one other known similar track.

The Last.fm dataset is a tremendous addition. The MSD contains artist-level tags and similarity from The Echo Nest, but most applications require song-level data, for example, automatic tagging of music (Section 6.1.3). The amount of tagging data easily available is also radically different, the next largest tagging collection is Magnatagatune (Section 4.2) with only 25,863 tagged samples. Finally, the similarity values that are in part derived from collaborative filtering¹⁵ provide the opportunity to develop content-based predictors to do similarity and recommendation.

¹³<https://www.last.fm/>

¹⁴<http://labrosa.ee.columbia.edu/millionsong/lastfm>

¹⁵For an introduction to collaborative filtering, see [Jannach *et al.*, 2010].

5.2.6 Taste profiles

Contributed once again by The Echo Nest but at a later date, the Taste Profile Subset¹⁶ is a collection of user data from an undisclosed source. It tells us which songs were listened to by actual users. The data comes in triplets: user identifier, track identifier, and play count. This dataset is essential in order to perform collaborative filtering with the MSD, and it is at the core of the Million Song Dataset Challenge presented in Section 6.2.3. The Taste Profile Subset gives information about 1,019,318 unique users that have collectively listened to 384,546 unique MSD songs, for a total of 48,373,586 (user, track, play count) triplets.

5.2.7 Audio collection

Independent of us or anyone involved in creating the MSD, a research group in Austria recently downloaded audio snippets for almost all tracks in the MSD [Schindler *et al.*, 2012]. Even if those snippets are probably copyrighted themselves, the researchers can compute new features (such as MFCCs, Section 2.3.2) and release them to the community¹⁷. This external source opens up the MSD data to researchers that absolutely need to compute their own audio features.

5.3 Distribution

A challenge for a research laboratory that does not own large servers is to distribute a large data collection. The MSD takes about 280GB to download, not something people would do every day. In this section we discuss the solutions we found to this problem.

The core of the MSD, the data coming from The Echo Nest, is the largest by far because of the audio features. Data for an average song contains two matrices of 12×450 elements for the *pitches* and *timbre* features, and these numbers add up. We decided to distribute the data for each song as an independent file. The main reason is that it makes it easy to break up the MSD into standalone subsets. Each file needs to contain various type of information: strings, integers, floats, some as size-varying arrays or matrices. We settled on the HDF5 file format¹⁸, a format developed

¹⁶<http://labrosa.ee.columbia.edu/millionsong/tasteprofile>

¹⁷<http://www.ifs.tuwien.ac.at/mir/msd/>

¹⁸<http://www.hdfgroup.org/HDF5/>

in part by NASA to handle complex collections of astronomical data. It has also been ported to many languages, including C++, Matlab, and our language of choice, Python.

To host the MSD and handle the downloads, we partnered with Infochimps¹⁹, a “big data platform for the cloud”. They host datasets for free or for a fee (to the person downloading it) and they provide an API to query some of the sources. The download remains a somewhat painful manual task. Since most download tools do not handle more than a few gigabytes at a time, the dataset was broken into 26 independent downloads. Based on personal experience, we estimate that a researcher with a good internet connection can get the data in about 8 hours of non-intensive work.

Releasing the data for each song in an independent file has some drawbacks. For instance, to build the list of all artists in the MSD, one has to iterate over 1M files. To alleviate this, we prepared additional files with certain kinds of information. For instance, we provide a list of all the artist and song titles as a text file. We also prepared SQLite databases²⁰ with all the metadata. SQLite datasets are contained in a single file, and are thus easy to distribute. A sample SQLite query is shown in Figure 5.2. Finally, we dumped the whole dataset into a few large text files, comma-separated, on Amazon Web Services (AWS). This makes it easy to use a MapReduce system such as Hadoop²¹ to work on the MSD. The AWS part of the work was done with the help of P. Lamere²².

The one decision we would reconsider is the use of the HDF5 format. It is a great format for medium-sized collections since it can perform efficient SQL-like queries. However, this search feature is of a little use when we save the data of only one song in a HDF5 file. Also, since it is not a well-known format, some users have described it as a barrier of entry to using the dataset. There is an argument to be made for releasing the MSD in plain text files. It would be ugly: matrices would have to be flattened out, a line would take thousands of characters, and string sanitizing could be challenging for artist names and song titles. However, users would have a better grasp of what they are dealing with.

¹⁹<http://www.infochimps.com/>

²⁰<http://www.sqlite.org/>

²¹<http://hadoop.apache.org/>

²²<http://musicmachinery.com/2011/09/04/how-to-process-a-million-songs-in-20-minutes/>

5.4 Problems and lessons learned

Building the largest research collection of your field is not something you do everyday. Thus, a lot of decisions turned out to be suboptimal, and some errors got through that we could have caught. In this section we mention the most common errors we found in the dataset after its creation. Errors in a million-entry collection are to be expected, but the more we prevent in the future the better.

5.4.1 Duplicate songs

The first common error we mention is to have multiple copies of the same song in the MSD, but with different tracks. In The Echo Nest world, a song is a musical work, but it can be recorded by different tracks, each track being a very specific recording. Recordings of a given song should be considered identical by humans, e.g. the audio quality or the codec used might have changed, but no major difference can be heard. Therefore, it should be useless to have different tracks for the same song in the MSD. Unfortunately, we found 53,471 of them for a total 78,190 undesired tracks (some songs have more than one duplicate).

The main reason for those duplicates is the way we built the dataset (see Section 5.2.2):

- find an artist (e.g. by popularity);
- find its songs;
- for each song, download one track (first one returned).

We made sure we never downloaded the same track twice, but if we reached a given song twice, and each time it gave us a different track, we ended up with duplicates. A way to spot those tracks is simply to refuse duplicate song ID, not just track ID. This simple trick would have been useful during the creation of the MSD.

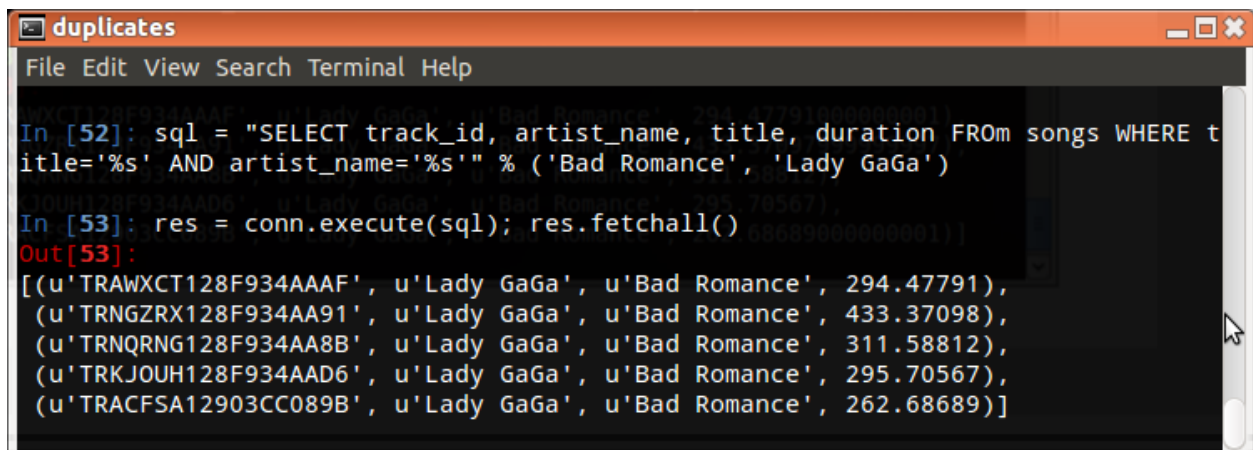
5.4.2 Duplicate tracks

Another duplication problem, this time more difficult to spot, is when the same song possesses two song identifiers in The Echo Nest world. The Echo Nest handles a collection of over 25 million songs through human and automated processes. Unfortunately, no such system is perfect. For

instance, the following two tracks from *Soda Stereo* have the same title and the same duration within a second.

```
track: TRTZIEN128F934048F  song: SOSQCAM12AB01893B0
      artist: Soda Stereo  title: Tele-Ka
      duration: 144.53506  7digital ID: 5668949
track: TRVBTFD128F9333BC0  song: SONDCUK12AB018596E
      artist: Soda Stereo  title: Tele-Ka
      duration: 145.3971   7digital ID: 5617503
```

However, they have a different track ID and song ID from The Echo Nest and different 7digital track IDs. The two track IDs could be justified by the slight different in duration, but how to justify a different song ID?



```
duplicates
File Edit View Search Terminal Help
In [52]: sql = "SELECT track_id, artist_name, title, duration FROM songs WHERE t
title='%s' AND artist_name='%s'" % ('Bad Romance', 'Lady GaGa')
In [53]: res = conn.execute(sql); res.fetchall()
Out[53]:
[(u'TRAWXCT128F934AAAF', u'Lady GaGa', u'Bad Romance', 294.47791),
 (u'TRNGZRX128F934AA91', u'Lady GaGa', u'Bad Romance', 433.37098),
 (u'TRNQRNG128F934AA8B', u'Lady GaGa', u'Bad Romance', 311.58812),
 (u'TRKJOUH128F934AAD6', u'Lady GaGa', u'Bad Romance', 295.70567),
 (u'TRACFSA12903CC089B', u'Lady GaGa', u'Bad Romance', 262.68689)]
```

Figure 5.2: Duplicates of a Lady GaGa song.

Duplicates found in the MSD through an SQLite query based on the artist name and title. We also report the track ID and duration in seconds.

We found many of these in the dataset, and received confirmation that some are indeed issues in The Echo Nest database. Some errors seem to come from The Echo Nest and some from their partner 7digital. It is unclear if those are mostly human errors, fingerprinting errors, database merging errors, etc. On our side, we probably should have kept a list of existing artist names and titles, and prevented duplicates. However, often the artist names or titles can vary slightly. It is also unclear if two tracks with different durations should be consider the same or not. Another

example is shown in Figure 5.2 where we have 5 problematic cases with greatly varying durations (Figure 5.2 is also an example on how to use SQLite databases to query the MSD metadata).

One lesson learned is that dealing with a music collection of that size will inevitably bring its share of duplication problems. Moreover, some can only be fixed by a human actually listening to the songs, and even then some cases might be non trivial.

5.4.3 Song - track mismatches

The trickiest problem we encountered was mismatches between songs and tracks. In a perfect world, songs have tracks, and a track belongs to that parent song. Thus, we can query for tracks given a song, and for a song given a track. To handle this, The Echo Nest must maintain “forward links”: a list of tracks associated with each song, and “backward links”: a song associated with each track. Those links do not appear in the MSD since we usually have only one track per song, and we built the dataset using only the forward links.

A mismatch happens when those two links have lost synchronicity: a song has a track, but the track thinks it belongs to another song (an example is provided in Figure 5.3). In short, we can get a track that is completely unrelated to the song we had selected. Remember that we built the MSD using a top-down approach (Section 5.2.2) from artist to song to track.

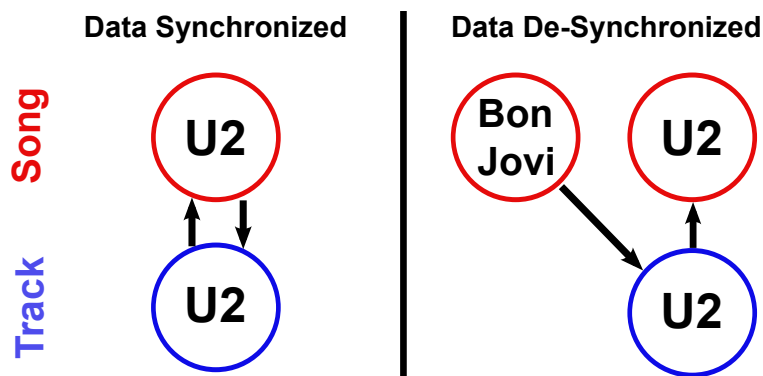


Figure 5.3: Visualization of a mismatch error in the MSD.

Reasons for such de-synchronization between songs and tracks in The Echo Nest database are unclear, and probably are a result of bugs or human manipulations. One thing is certain, the data were not saved in relational database at the time of the MSD creation. A database such MySQL would not allow mismatches to be saved.

The problem became apparent after building the Taste Profile Subset (Section 5.2.6). Data from the undisclosed source were matched to songs using The Echo Nest API. To get the info about a song, one would simply have to look at one of its tracks in the MSD. The problem was that mismatches made some of those tracks wrong for the song.

We found most of these mismatches using artist name and title matching, and the details were posted online²³. However, given the difficulty of matching music metadata among other things, this issue will never be completely fixed in the MSD. It is yet another reminder not to blindly trust professional sources.

²³<http://labrosa.ee.columbia.edu/millionsong/blog/12-2-12-fixing-matching-errors>

Chapter 6

Applications and impact

Dropping more data into the world is rather meaningless unless it comes with some demonstrable uses and starting points. Therefore, we give a quick overview of tasks that can be performed with the MSD. Note that we were not involved with all the work presented in this chapter. The goal is to provide some insight of what is possible with this collection, and how the MIR research community might use it.

6.1 Applications

In this section we present straightforward ways to perform traditional MIR tasks on the MSD. These examples can be considered as demonstrations, and many were released as part of the initial announcement of the MSD [Bertin-Mahieux *et al.*, 2011a].

6.1.1 Meta data analysis

To start our exploration of the MSD, a good first step is analyzing the meta data: number of artists, number of songs per artist, number of tags, etc. We can turn those questions into something more playful, for instance does the average length of artist names increase or diminish over time? By plotting artist name length by the average release date of their songs, (Figure 6.1) a simple linear regression would suggest that they are getting shorter. The least squared regression has parameters: gradient = -0.022 characters/year and intercept = 55.4 characters at year 0 AD. That being said, we should be careful before attaching too much importance to such a simple analysis.

6.1.2 Year prediction

As a first task using audio features, we introduced year prediction in [Bertin-Mahieux *et al.*, 2011a]. The goal is to predict the release year based solely on audio features. As features, we use the average and covariance of the timbre vectors for each song. Using only the nonredundant values from the covariance matrix gives us a feature vector of 90 elements (12 means plus 78 unique covariances) per track. We compare two algorithms: k -NN and Vowpal Wabbit (VW, [Langford *et al.*, 2007]), both chosen because of their applicability to large-scale problems. We also report the error obtained by the best constant predictor. Results are presented in table 6.1. We report the average absolute year difference and the average squared difference between the release year and the predicted one.

method	diff	sq. diff
constant pred.	8.13	10.80
1-NN	9.81	13.99
50-NN	7.58	10.20
VW	6.14	8.76

Table 6.1: Year prediction results.

Taken from [Bertin-Mahieux *et al.*, 2011a].

This is only a proof of concept, the features we use are too simplistic and we only compare two widely different algorithms. What is interesting is that year recognition had never really been studied before due to the lack of a proper dataset. We will return to this problem in Section 6.2.2.

6.1.3 Automatic tagging of music

One of the most-studied machine learning tasks in MIR is automatic tagging of music. Traditionally, the goal is to apply tags, or keywords, to a song solely based on the audio. Numerous teams have tackled this problem, early work includes [Lamere, 2008; Eck *et al.*, 2008; Bertin-Mahieux *et al.*, 2008; Bertin-Mahieux *et al.*, 2010a; Mandel and Ellis, 2008; Turnbull *et al.*, 2008].

Using artist terms from The Echo Nest, we provide a train and test set of artists in the MSD. As a preliminary analysis, restricting ourselves to the 300 most popular terms, we know that each artist is tagged with 19 terms on average. If we tag all artists with the same top 19 terms (i.e.

no learning involved), we obtain a precision of 2.3% and a recall of 6.3%. One can now extract features, run an algorithm such as k -NN, and make the results fully reproducible. At the time of this writing, no such result has been reported in the literature.

The later addition of the Last.fm dataset (Section 5.2.5) made the task even more interesting since researchers can merge artist-level and song-level tags. Researchers from Vienna also released tags extracted from All Music Guide², available online³.

6.2 Impact

In this section we focus on more ambitious projects built on top of the MSD, mostly led by other MIR researchers. The goal is to highlight progress on tasks that were very difficult to study before the MSD was available.

6.2.1 Playlisting

McFee and Lanckriet [McFee and Lanckriet, 2011; McFee and Lanckriet, 2012] crawled playlists from the website The Art of the Mix⁴. Using the playlists as training data, they built a generative model of playlists based on features from the Million Song Dataset: audio features, user data, tags, years, and lyrics.

The advantage of the MSD in their work is that they can quickly measure the influence of different features on their generative model. For instance, tags seem to be more relevant than lyrics while making a playlist. Finding each of these distinct feature types for a common, large set of songs would have been quite a challenge.

6.2.2 Evolution of pop songs

Serrà et al. took on the ambitious task to characterize music evolution over time, especially through its harmonic structure [Serrà *et al.*, 2012]. The idea is to create a network of chroma transitions for different time periods (e.g., the 1950s, the 1960s, etc) based on the MSD audio features and track

²<http://allmusic.com>

³<http://www.ifs.tuwien.ac.at/mir/msd/download.html>

⁴<http://www.artofthemix.org>

release years. By describing these networks, the authors found trends that they associate with a simplification of the harmonic structure over time.

The originality of the work and its scope are exciting. The MSD provides enough data that someone can reasonably try to measure the evolution of western pop music over the last 50 years. We can only hope that this work will spawn more studies on the subject, and that “music evolution” will become a typical MIR task.

6.2.3 The Million Song Dataset Challenge

Using the user data from the Taste Profile subset (Section 5.2.6), we set up a music recommendation contest in 2012 [McFee *et al.*, 2012]. The contest was hosted by Kaggle⁵ and everyone could participate. It attracted 193 participants spread across 153 teams for a total of over 990 submissions. The winning spot was highly contested, as shown in Table 6.2. Note the increase in score from the simple benchmark we provided at the beginning of the contest.

We are still drawing conclusions from the results, but a first result is that collaborative filtering is the obvious method of choice and outperforms attempts at content-based recommendation. The winner of the first edition is [Aioli, 2012] and a summary of his method has been written by Mark Levy⁶. Aioli used an item-based recommender, and tweaking the distance function between items provided the most improvement.

As there might be a second iteration of the contest, it is worthwhile to state its goals:

- to raise awareness of MIR problems outside the MIR community;
- to encourage the use of diverse data for music recommendation;
- to drive the scaling of academic music recommender systems.

Obviously, there is a great overlap with the goals of the MSD (Section 4.3).

A major criticism raised by participants was the lack of context in the data, a hot topic in recommender systems (e.g. see [Su *et al.*, 2010]). Context information could be details about the user, his activity while listening, or the time stamps of the plays. Time stamps are expected to

⁵<http://www.kaggle.com/c/msdchallenge>

⁶<http://mir-in-action.blogspot.com/2012/09/collaborative-filtering-still-rules.html>

Team	Score (mAP)	# of entries
aio	0.179	27
learner	0.171	26
nohair	0.159	18
Team ubuntu	0.157	44
TheMiner	0.156	17
Cygnus	0.155	15
KXEN	0.151	49
kosmos	0.150	21
spring	0.146	13
werewolf	0.143	2
benchmark	0.021	-
random	0.000	-

Table 6.2: Top contestants in the MSD Challenge.

Team name, score (mAP, see Section 9.3) on the test set, and number of entries for the top 10 participants in the MSD Challenge. The “benchmark” submissions consist of submitting the same 500 most popular songs for every users. Full data available at

<http://www.kaggle.com/c/msdchallenge/leaderboard>.

be of particular importance in music recommendation: users do not listen to the same song on a Monday morning and on a Saturday night. Also, researchers believe music should be recommended based on what was recently played, making the problem closer to playlist generation (Section 6.2.1). Unfortunately, no open, large-scale music dataset exists with this information. Also, companies that do possess such data might be worried about releasing it because of privacy concerns. If someone manages to de-anonymize the data based on the listening behaviors (i.e., finding the identity of users, similar to what [Narayanan and Shmatikov, 2008] did with the Netflix data), public backlash and lawsuits could be envisioned.

6.2.4 Discussion on MIR evaluations

The creation of the MSD and the organization of a contest like the MSD Challenge (Section 6.2.3) brings us to a discussion of the way common evaluations are performed in MIR. The reference for the last few years has been MIREX⁷, a yearly contest organized in parallel with the ISMIR conference⁸. A more recent but similar evaluation campaign is MusiClef⁹. Headed by Stephen Downie¹⁰, MIREX is a set of challenges (or tasks) proposed by the community. A MIREX task include at least one dataset and one or more error measures. For instance, for the “Audio Tag Classification” task, one dataset used is called MajorMinor, and one error measure is the area under the receiver operating characteristic curve (AUC-ROC). In most tasks, data is private, and participants upload their code to a platform that runs it on the data. The upside is that it protects the data: a team cannot easily overfit on the test set, and it overcomes the copyright issue for data that cannot be distributed. The downside is, maintaining a platform that runs code uploaded by external contributors in many different languages is difficult: it requires a lot of person-hours, which in academia implies money for grad students.

Recently, there has been a lot of informal discussion regarding the future of MIREX, in part because of the difficulty of financing such an endeavor. Resources like the MSD and platforms such as Kaggle and TunedIT¹¹ offer an alternative. A team that wants to sponsor a competition can gather the data, decide which error measure to use, and set up the online platform to do so. All that is left is advertising the contest and analyzing the results. It has been a successful recipe in the case of the MSD Challenge. It would enable the creation of a decentralized MIREX where the top organizers only need to keep track of the different contests.

To be fair, not all MIREX competitions can be set up as a Kaggle task. At the moment, the training data has to be downloaded, which brings us back to the copyright issues. Therefore, most tasks that require the actual audio would be problematic (this includes the tasks of Audio Beat Tracking and Audio Melody Extraction for example). However, when it is possible to move

⁷http://www.music-ir.org/mirex/wiki/MIREX_HOME

⁸<http://www.ismir.net/>

⁹<http://www.multimediaeval.org/mediaeval2012/newtasks/music2012/>

¹⁰<http://www.lis.illinois.edu/people/faculty/jdownie>

¹¹<http://tunedit.org/>

a task onto a platform such as Kaggle, we strongly recommend it. The MIR community simply cannot sustain the use of paid graduate students to run (and debug) everyone else's code for every imaginable task.

Part III

Cover song recognition

Chapter 7

Introduction to cover song recognition

Music videos are abundant on the web, and tracing the dissemination of a given work is a challenging task. Audio fingerprinting [Wang, 2003] can be used to find an almost exact copy of a given music track¹, but the same technology will not work for finding novel versions of the original work, i.e. “cover songs”. We use the term “cover song” to include both new or live versions of a song by the original artists, as well as a new recording by different musicians.

Cover song recognition is of interest for many reasons, both commercial and academic. Commercial reasons include the correct handling of songwriting royalties and detection of copyright infringement. For instance, on YouTube, the copyright holder of the musical work can have the covers of her work removed, or she can receive part of the advertising revenue from the video². Figure 7.1 shows how easy it is to find thousands of covers online from the metadata alone, but many more are not identified as covers. Cover song recognition can also be used as a discovering tool, e.g. to find live versions of a given song.

From an academic viewpoint, cover song recognition involves finding and understanding transformations of a musical piece that retain its essential identity. This can help us develop intelligent audio algorithms that recognize common patterns among musical excerpts. This vision can be found in [Downie *et al.*, 2008] where the authors describe cover song recognition as the starting

¹Fingerprinters have to be robust to encoding, sound quality, and additional noise, but the original song is expected to remain untouched.

²From <http://www.youtube.com/t/faq>: “content owners are given choices with Content ID to block, track, or leave up and make money from the videos uploaded to YouTube”.

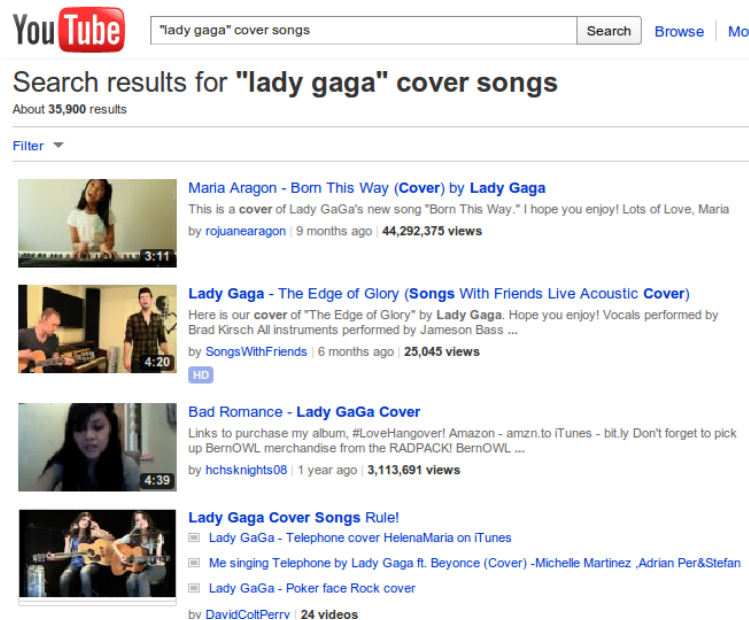


Figure 7.1: Lady Gaga covers on YouTube.

Search result for self-identified cover songs of Lady Gaga on YouTube on November 22nd, 2011.

This simple query produces about 35,900 results.

point in the quest for higher-level music descriptors.

This interpretation is the motivation of this work. We are looking for representations and methods that can efficiently describe a large amount of music data in a “meaningful way”. An example of what we would call meaningful is an audio feature space where cover songs would be neighbors. Thus, by finding cover songs, we implicitly describe a tonal manifold of music (possibly reminiscent of the “Swiss roll” manifold in Figure 3.5).

7.1 Problem definition

We dive into the definition of cover songs and cover song recognition before going any further. A song is a cover of another if they are two different performances of the same musical work. Serrà lists 10 labels used to characterize different versions (what we are calling covers) in [Serrà *et al.*, 2010; Serrà, 2011]:

- remaster;

- instrumental;
- mashup;
- live performance;
- acoustic;
- demo;
- standard;
- medley;
- remix;
- quotation.

In short, we consider any song that can be described with one of these labels to be a cover. Note that we also consider this a symmetric property, i.e. if a song A is a cover of a song B , B is also a cover of A .

This definition is far from perfect. For instance, *remix* is a rather ambiguous term as Serrà points out [Serrà, 2011]. A remix might or might not include substantial changes to a song, and it could very well consist only of improvements to the audio quality. Such a modification would not confuse a music fingerprinter, and should probably not be considered a cover. Another example is the *quotation*, “the incorporation of a relatively brief segment of existing music in another work” [Serrà, 2011]. If the length of this segment tends to zero, it might not be long enough to create a cover.

Our definition is loose, and we could argue endlessly on what makes a cover. In practice, these extreme cases seldom arise. One reason is the way the SHS dataset is built since it only includes items manually submitted by users. Users do not bother with uninteresting covers, e.g. a slightly modified recording that would sound identical to most listeners. The other reason to include a maximum number of songs in the definition of a cover is to simplify the task description and the possible inclusion of new data, even if it is at the expense of increased accuracy.

Using the definition of a cover song, the task is: given a query a song and a database of music (usually not containing the query), identify the cover songs of the query.

Since making a binary decision is still difficult given the state of the art, an easier question is: given a query, rank all songs from most likely to being a cover to least likely. In this work, we will try to solve the latter question.

7.2 Difficulty of comparing chroma features

Cover song recognition is almost always reduced to the problem of comparing chroma representations (where chroma features were introduced in Section 2.2). Since the instrumentation and the style can vary widely between two covers, the common information is related to the score. As we saw in Section 2.2, such information would be encoded through chroma features or one of its variant.

Unfortunately, comparing chroma in a “musical way” is a particularly difficult task. We performed a set of experiments in [Bertin-Mahieux *et al.*, 2010b; Bertin-Mahieux *et al.*, 2011b] to show that comparing a patch of chroma vectors bin by bin using the most obvious methods does not give satisfactory results. Figure 7.2 provides some intuition. To a human, the second patch would be the closest to the original one (top one). It is indeed the same patch with an offset of one step in time. However, according to the Euclidean distance, it is the least similar patch among the four. The reason is that Euclidean distance (and most common distances, see [Bertin-Mahieux *et al.*, 2011b]) compares each bin individually and does not look at the overall structure.

With more details, the first experiment [Bertin-Mahieux *et al.*, 2010b] was to build a codebook using k -means and Euclidean distance. Figure 7.3 shows the most common codes from such a codebook. The main result is how *boring* these k -means centroids look. Using the Euclidean distance has a huge averaging effect, and the best centroids are those that represent one note, or maybe one transition. It could be the start of a chord recognition system, but it is not powerful enough to compare melodies.

Experiments from [Bertin-Mahieux *et al.*, 2011b] confirm this conclusion. In this work, we masked out an arbitrary block of 12 successive chroma values, and compared a number of schemes for inferring these missing values from what remained. Figure 7.4 summarizes the results. Taking many algorithms (e.g. a row-wise average, k -NN, a linear transform, ...), we compare error measures and see which ones favor reconstructions that do not look like actual music.

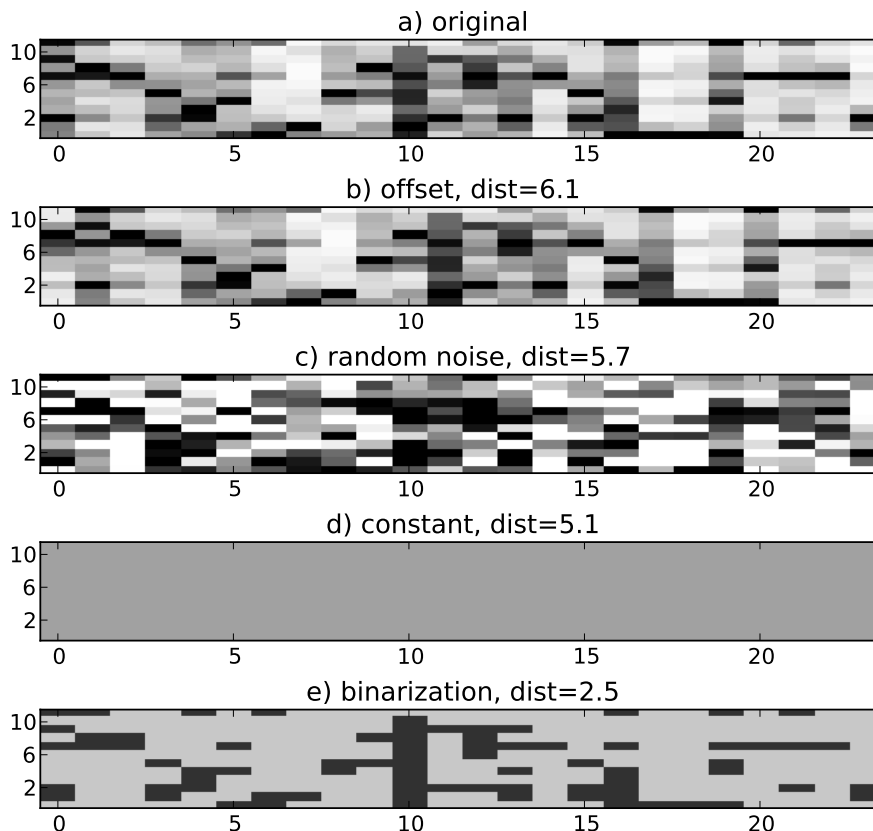


Figure 7.2: Comparison of chroma patches.

Horizontal axis is time. The figures contain a) the original chroma patch, 12×24 , b) the original patch with an time offset of 1, c) the original patch with added random noise, d) the patch with the average of the original values, e) the patch binarized to two values. Patches b) to e) are ordered in decreasing Euclidean distance to the original one.

In Figure 7.4, the third figure is the worst reconstruction according to the Euclidean distance, and the sixth figure is the best one. This is similar to the experiment in Figure 7.2 and most humans would disagree. Another result shows that this is not a problem exclusive to the Euclidean distance. Out of 14 distance measures we tried³, most agree with Euclidean distance, including Kullback-Leibler and the Jensen difference. “delta diff.” and “D-ent” are attempts at building more structure-based measures, but they can not be used on their own.

For cover songs, what matters is the structure and not the localized variations. The rest of

³<http://www.columbia.edu/~tb2332/proj/imputation.html>

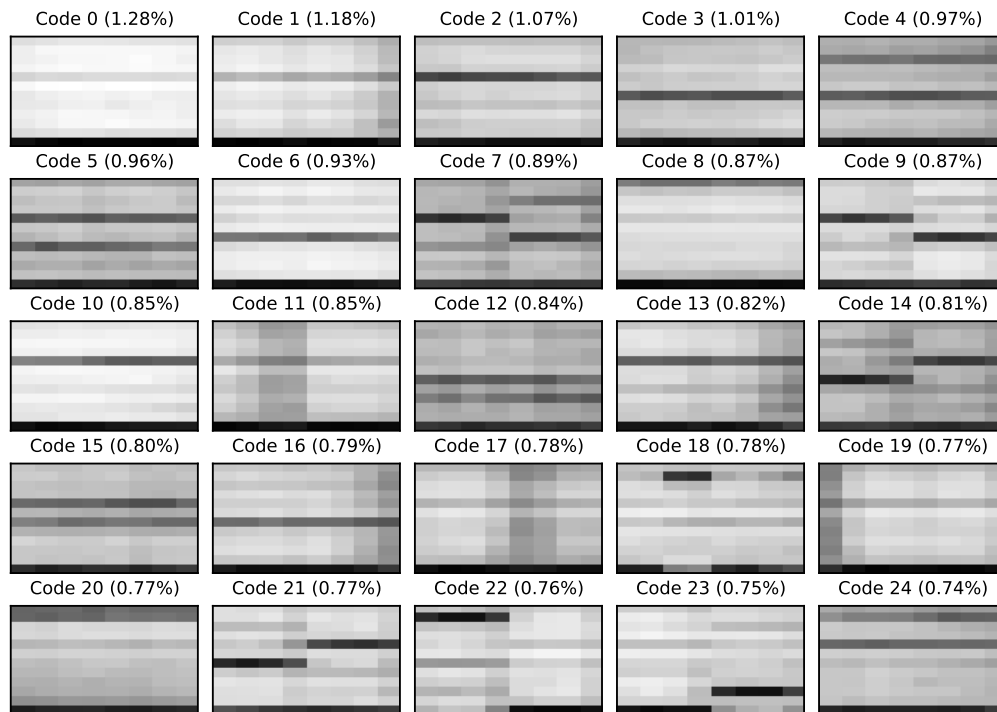


Figure 7.3: Centroids from clustering chroma patches.

Taken from [Bertin-Mahieux *et al.*, 2010b], represents the 25 codes that were the most commonly used in our dataset. Codes are from a 200-entry codebook trained on 2 bars resampled to 12×8 patches. The proportion of patches accounted for by each pattern is shown in parentheses.

this work describes representations that can encode and summarize these structures so they can be compared between covers. Thus, we do not have to compare chroma patches directly bin by bin.

7.3 Cover song recognition systems

Cover song recognition has been widely studied in recent years, including a specific task within MIREX since 2007 [Downie, 2008]. An early system is Ellis and Poliner [Ellis and Poliner, 2007] and a good overview is in Serrà's thesis [Serrà, 2011]. A significant amount of work has been done with classical music [Kim and Narayanan, 2008; Miotto *et al.*, 2010; Miotto and Orio, 2008; Müller *et al.*, 2005] but popular music can present a richer range of variation in style and instrumentation.

Most cover song works were tested on a few hundred or thousand songs, a size not comparable

to commercial collections (Spotify⁴ claims more than 16M tracks). However, some of the work was made to scale and could be extended to larger sets. Kurth and Müller [Kurth and Müller, 2008] use a codebook of CENS features to encode songs, thus creating an index that can be searched efficiently. Casey and Slaney [Casey and Slaney, 2007] use locally-sensitive hashing (LSH) to quickly compare chroma patches, or *shingles*. Yu et al. [Yu et al., 2009] also use LSH to compare different statistics about a song, and [Marolt, 2008] experiment with LSH using different representations, one of which is highly similar to the one in Chapter 9. Kim and Narayanan [Kim and Narayanan, 2008] look at chroma changes over time and suggest using these changes as hash codes. Another hash code-based method is from [Martin et al., 2012]. The authors transform chroma vectors into simple chords and match chord sequences using a very fast algorithm.

The idea of using a mid-level representation that is invariant to key changes has been explored in [Jensen et al., 2008; Marolt, 2008]. In [Jensen et al., 2008], the authors use an 2-dimensional cross-correlation function, thus it finds the best offset in time and the best rotation along the semitone axis of the chromagram in order to match two songs.

Similarly, [Marolt, 2008] computes the two-dimensional power spectrum of a chromagram in order to get a representation that is invariant to both axes, time and semitone. This is the core of our method in Chapter 9, where we use the $2D$ magnitude of the Fourier transform, i.e. the square root of the $2D$ power spectrum.

7.4 Why scale cover song recognition

As we saw in Part II, many experiments can be performed on a large scale using the MSD. In this section, we further justify our choice of investigating cover song recognition instead of any other task.

First, as we saw in this chapter, the difference between cover songs can be extremely varied. This implies that a system that performs well on this task understands many music-specific transformations: change of instrumentation, change of tempo, change of key, etc. Thus, a solution to this problem is likely to be useful for other MIR tasks. For example, music similarity and music segmentation, two problems dealing in part with the tonal content of a song, could benefit from

⁴<http://www.spotify.com>

lessons learned on cover song recognition.

Second, this task needs to “find its second wind”. In the past few years, it had been developed almost entirely with MIREX in mind. This has led to solutions optimized for the dataset used in that evaluation. Such solutions do not have to scale past a few thousand examples, for which DTW-like algorithms are fast enough. Some contestants also overspecialized their system. For instance, in their submission to MIREX 2009, [Serrà *et al.*, 2009] improved their system by clustering songs based on the pairwise distance between all entries in the dataset. This method works because each song possesses many covers in the MIREX evaluation. That extra knowledge (and the use of a full distance matrix) would likely not generalize to a real, large-scale dataset.

Those two reasons make cover song recognition a perfect task to further study on a large scale. It has the potential to provide insight for many MIR tasks, and current systems are constrained because they are developed with a certain artificial dataset in mind.

7.5 Conclusion

In this chapter we explained the difficulty of comparing chroma or chroma-like features, and explored different methods to overcome this. They usually fall in one of these categories:

1. correlation-based methods with varying offsets and rotations (e.g. [Ellis and Poliner, 2007]);
2. dynamic-time warping methods or similar time-adjusting comparison (e.g. [Serrà *et al.*, 2008]);
3. model-based methods (e.g. [Serrà, 2011, Chap. 5]);
4. hashing-based methods (e.g. [Casey and Slaney, 2007]);
5. methods with higher level features (e.g. [Marolt, 2008]).

The first category is limited in complexity and accuracy, and the second and third are difficult to scale, i.e. a query has to be compared with every instance in a database. Therefore, in the rest of this work, we present an hashing-based method (Chapter 8) and a more promising higher level feature computed from chroma (Chapter 9). We will be particularly interested in the higher level feature since it might be useful to understand the music harmonic space in general, and not just for improving cover song recognition.

7.5.1 Working from audio

In this chapter and the subsequent ones, we always assume we have access to precomputed chroma features, usually on the beat level (as described in Section 2.4). This is the case when working with the MSD. However, we explain in Appendix B how to obtain these from audio data. The method closest to what we do here is to use The Echo Nest API [EchoNest,] but other code freely available online can compute similar features. In particular, we explain how to start from a video on YouTube as the query song.

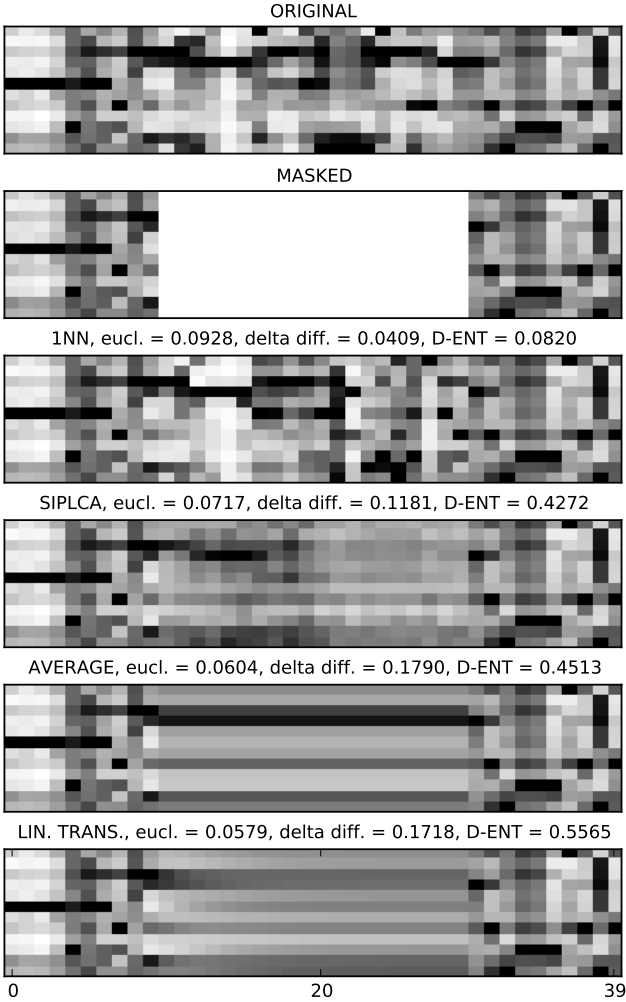


Figure 7.4: Reconstruction and some error measures.

Taken from [Bertin-Mahieux *et al.*, 2011b], four algorithms are used to create reconstruction.

These are evaluated with three measures, including Euclidean.

Chapter 8

Hashing methods

Given the size of the dataset we are working with, a natural starting point is hashing methods ([Knuth, 1973]). The idea behind hashing is to create a code that two identical items would share. Comparing the codes must be fast. Finding whether a given item is in a database only requires an examination of the items that are associated with its code. If designed correctly, there should be few such items.

Music fingerprinting methods (e.g. [Wang, 2003]) use a similar idea that we broadly simplify here. Small sections of the song are attributed a (hash) code that should be constant across encodings of a song (different audio quality, some added noise, etc). Given a query piece of audio and its associated codes, we can find all songs in a database that share those codes. If all goes well, the correct song should stand out as having most hash codes in common. The advantage of this method is the querying time. Once all the hash codes are entered in a database system (e.g. MySQL¹), retrieving particular codes can be extremely fast. Thus, audio fingerprinting methods can scale to millions of songs.

The problem with cover songs is designing hash codes that would be constant across covers. A lot can change between an original song and its cover, including:

- instruments;
- style;

¹<http://www.mysql.com/>

- tempo;
- key.

In particular, the hash codes used in [Wang, 2003] would be useless. The authors use the relative position between local landmarks, a local landmark being a given frequency and point in time that contains more energy than anywhere else in its spectrogram neighborhood. Those landmarks would vary greatly between covers for the reasons mentioned above.

In this chapter we present our attempt at building hash codes that are better suited to cover songs. Our system also served as a proof of concept, as it was the first result reported on such a large collection. Our method was presented in [Bertin-Mahieux and Ellis, 2011].

8.1 Fingerprint features

In this Section we present the core of the algorithm, i.e. how to pass from a chroma matrix to a few dozen integers for each song, then how to query a database using those codes. We always start from a beat-aligned chroma matrix as illustrated at the bottom of Figure 8.2.

8.1.1 Example

First, let's look at two covers that are quite similar to one another. In Figure 8.1, at the top, we see two chromagram patches from two songs: “Yellow” by Coldplay, and “Yellow” by G4. Coldplay does not need any introduction and they wrote the original song; G4 is a vocal group from England. The chromagram patches presented here were chosen to ensure that they represent the same section of the song. They are indeed quite similar. The darker bins represent a pitch played with higher energy (e.g., at position (3, 4) in the top left image). Most of the time, the “darkest line” represents the melody line. The melody line often does not change much (modulo key changes, which is not the case here) from one cover to another, although there is no guarantee.

The idea of our hashing method is to find salient points in the chromagram. They are represented in red in the bottom row of Figure 8.1. As explained above, these salient points will most likely be part of the melody line. We can then encode how to “pass from one point to another”, the blue lines, which we also call “jumps”. We can use these jumps to compare two songs: if they have a lot

in common, the two songs are likely to be covers. This is the intuition behind our method, which will be reexplained in detail in the following section.

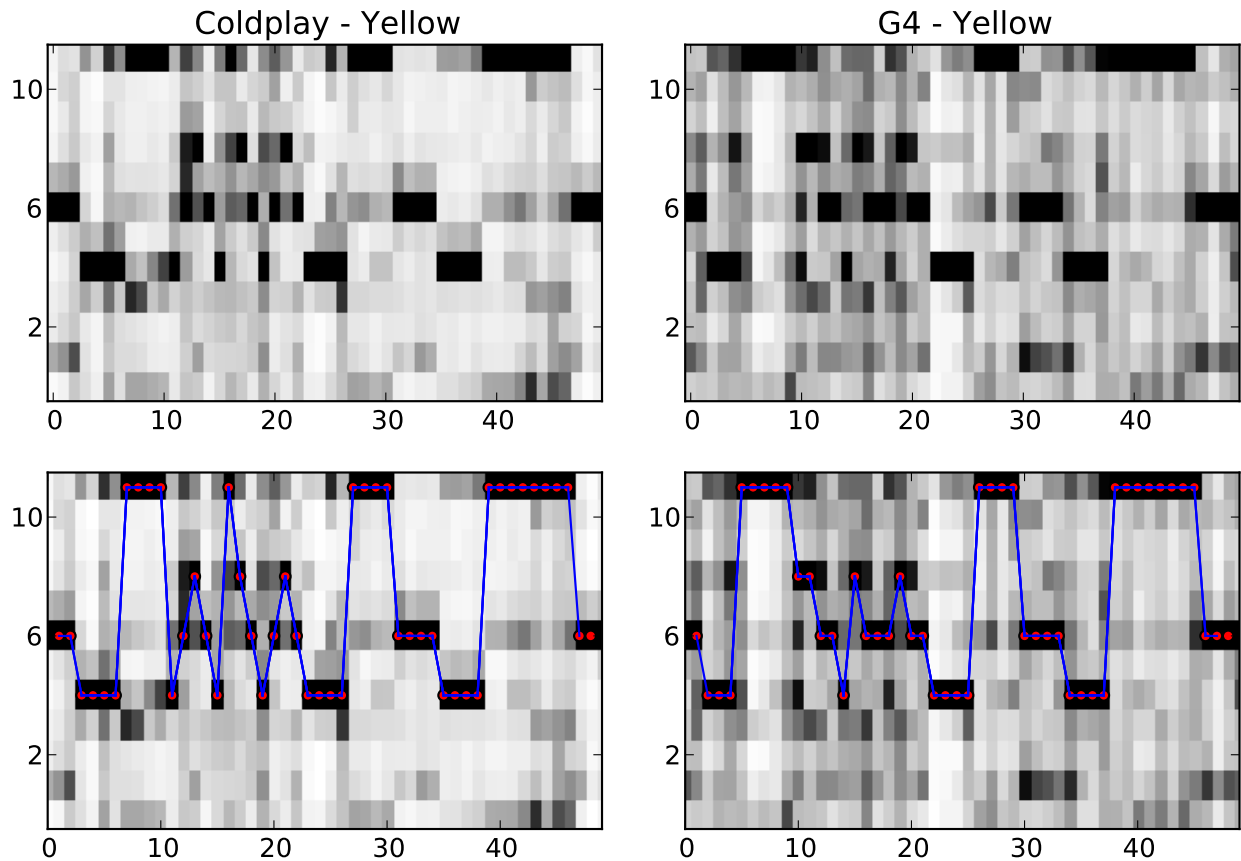


Figure 8.1: Two beat-aligned chromagrams and their jumpcodes.

The two songs are covers (the original one is on the left). The chromagram was shortened in time for visualization purposes.

8.1.2 Hash code

We explain how to get a set of hash codes from the beat-aligned chroma representation as in Figure 8.2. As previously mentioned, we can average the chroma values over beats. In our experiments, we saw that averaging over two beats yielded similar results while reducing the dimensionality. By analogy with the technique proposed in [Wang, 2003], we identify landmarks, i.e. “prominent” bins in the resulting chroma matrix. We use an adaptive threshold as follow:

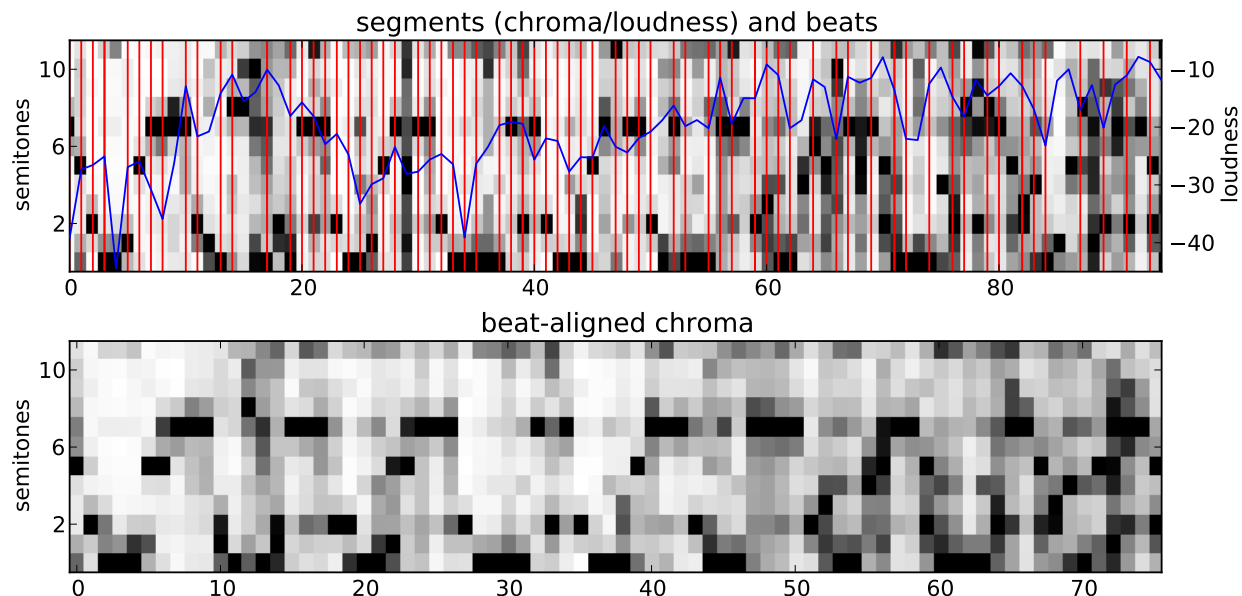


Figure 8.2: Segments, beats, loudness, and beat-aligned chromas.

The top figure contains chroma values for each segments, the beats (red vertical line) and loudness (blue curve). The bottom figure is a scaling of those values in order to get one 12-dimensional chroma vector per beat.

- Initialize T , the threshold vector of size 12 as the max energy for each semi-tone over the first 10 time frames (one time frame = two beats).
- At time t , we accept a bin as a landmark if its value is above the threshold. Let v be the value of this landmark and T_v the threshold value for that semi-tone, we set $T_v = v$ and $T_{i \neq v} = \max(T_v \psi, T_i)$, ψ being the decay factor. We also limit the number of landmarks per time frame to 2.
- Moving from t to $t + 1$, we use the decay factor ψ , we get $T_{t+1} = T_t * \psi$.

We get landmarks through a forward and a backward phase and we keep the intersection, i.e. landmarks identified in both phases. Once these landmarks are identified, we look at all possible combinations, i.e. set of jumps from landmarks to landmarks, over a window of size W . To avoid non-informative duplicates, we only consider jumps forward, or in one direction if within the same time frame. The simplest set of hash codes consists of all pairs of landmarks that fall within the

window length W . Such a pair can already provide information about musical objects like chords, or changes in the melody line.

A specific example is shown in from Figure 8.3. The red dots are the landmarks, for instance at positions $(1, 1)$ and $(2, 7)$. The blue lines are “jumps” between those landmarks.

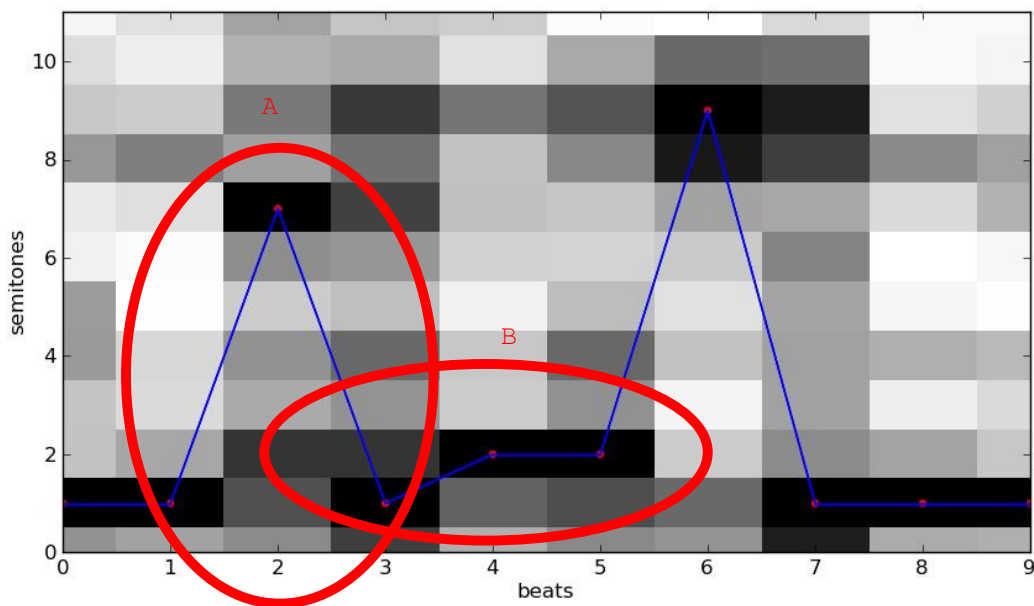


Figure 8.3: Closeup of landmarks and associated jumps.

These sets of jumps are hash codes characteristic of the song, and hopefully also characteristic of its covers. For sake of clarity, we refer to them as “jumpcodes”. A jumpcode is a list of difference in time and semi-tone between landmarks plus an initial semi-tone (the initial time frame is always 0). Taking the example *A* of Figure 8.3, we have three landmarks at positions $(1, 1)$, $(2, 7)$ and $(3, 1)$ in the beat-aligned chroma representation (and a window length W of at least 3), the jumpcode would be $\{((1, 6), (1, 6)), 1\}$. Note that for convenience, we take the jump between semi-tones modulo 12. The example *B* would give a jumpcode of $\{((1, 1), (1, 0)), 1\}$.

Figure 8.4 shows the larger picture, with jumpcodes represented for four cover songs. In this example, as it is often the case, the jumpcodes seem to mostly follow the melody.

In this work, we do not consider the position of the jumpcodes in the song. It might be useful to consider ordered sets of jumpcodes to identify cover songs, but this additional level of complexity

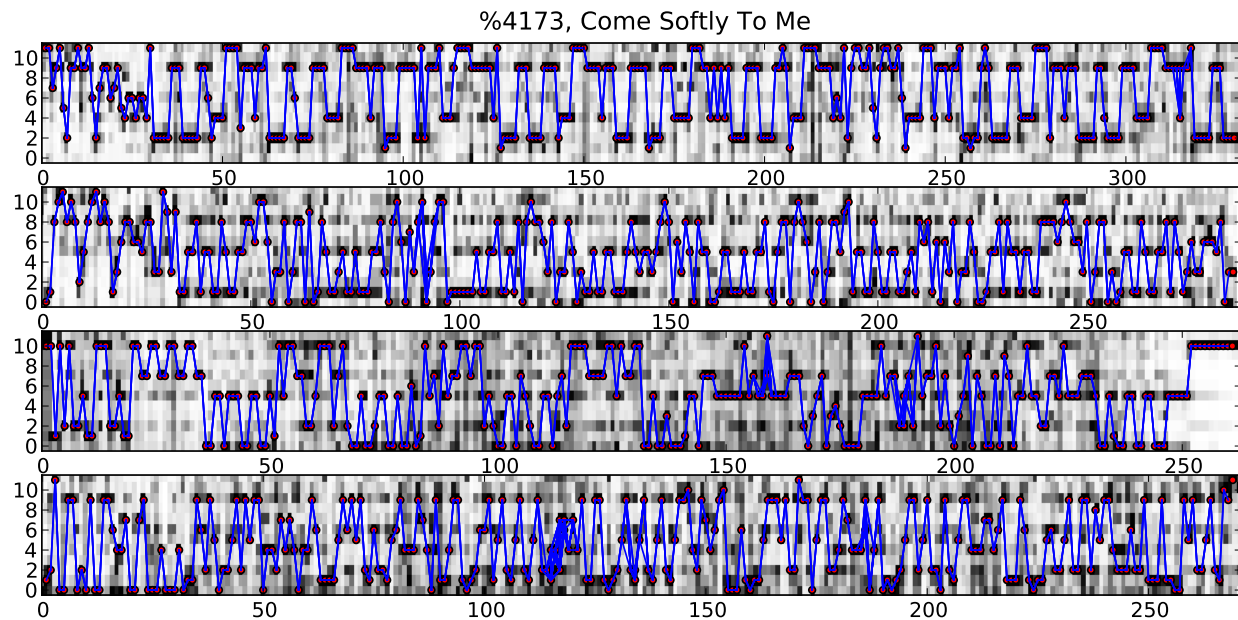


Figure 8.4: Landmarks and jumpcodes for 4 cover songs.

Covers of the work %4173 from the SHS dataset.

would be computationally expensive. Also, covers might use only parts of the original song, or mix the order of the song sections.

8.1.3 Encoding and Retrieval

The jumpcodes are entered in a database so we can use them to compare songs. We use SQLite, and like any other database, it is more efficient to encode jumpcodes as one integer. Many schemes can be devised, we use the following one that lets us easily compute a “rotated jumpcode”, i.e. the jumpcode we get if we rotate the song along its semi-tone axis. Such a rotation is important, remembering that many cover songs are not performed in the same key.

We have a set of k landmarks within a time window of size W . The landmarks are pairs of (chroma bin, time frame): $(b_1, t_1), (b_2, t_2), (b_k, t_k)$. The integer representation of the jumpcode can

be computed along these lines:

$$\begin{aligned} & b_1 + (t_1 - 0 + 1) \times 12 \\ + & [(b_2 - b_1) + (t_2 - t_1 + 1) \times 12 + 1] \times 12 \times (W + 1) \\ + & \dots \end{aligned}$$

The initial semi-tone value can be retrieved using a modulo 12, this is the most important feature of this particular scheme. This lets us rotate a jumpcode, i.e. infer the jumpcode we would get if the song was performed in another key, by removing the initial semi-tone, and adding back a rotation value between 0 and 11.

Experiments showed that some songs generate more jumpcodes than others (counting duplicates). In order to diminish that effect, we assign a weight to the jumpcodes. The weight is the number of times a particular jumpcode appears in the song divided by the logarithm (base 10) of the total number of jumpcodes in that song. Taking the logarithm is an empirical decision, but the idea is that it diminishes the importance of jumpcodes that appear often, and put emphasis on the ones that appear only a few times.

For the retrieval phase, we have all jumpcodes (with positive weights) for all songs in the database. Given a query song, we get all pairs (song, jumpcode) for which 1) the jumpcode also belongs to the query song, and 2) the weight of that jumpcode falls within a margin around the weight of that jumpcode in the query song. The margin is computed given a percentage α as $weight_{query} \times (1 - \alpha) \leq weight \leq weight_{query} \times (1 + \alpha)$. Once all the pairs from all jumpcodes in the query song has been retrieved, the frequency of each song is computed. The higher the frequency, the more likely a song is a cover of the original.

One peculiarity of cover song recognition is that the target might be in another key. Thus, for each query, we actually do 12 independent queries, one per rotation. As explained above, the encoding scheme of the jumpcodes lets us find their rotated values in a computationally efficient way.

8.1.4 Training

Due to the many parameters which are probably interdependent and consequently difficult to set, we created a set of 500 binary tasks out of the 12,960 songs from the training set. Given a query,

our hashing algorithm had to choose the correct cover between two possibilities. The exact list of triplets we used are available on the project’s website². We tuned to this 500-queries subset through many quick experiments in order to come up with a proper setting of the parameters. We do not claim we tried every combination, and it is possible that we missed a setting that would have greatly improved results. Parameters include the normalization of beat-chroma (including loudness or not), the window length W , the maximum number of landmarks per beat, the length (in number of landmarks) of the jumpcodes, the weighting scheme of the hash codes, the allowed margin α around a jumpcode weight, the decay ψ of the threshold envelope, etc. We present the best settings we found to serve as a reference point for future researchers.

8.2 Results

We first present our results on the 500-queries subset. To be thorough, we present two settings for the jumpcodes, the first setting leading to our best result on this subset. This first one, called “jumpcodes 1” used the following parameters: $\psi = 0.96$, $W = 6$, $\alpha = 0.5$, number of landmarks per jumpcode: 1 and 4, chroma vectors aligned to each beat. Unfortunately, it gave too many jumpcodes per song to be practical. The parameters for “jumpcodes 2” were: $\psi = 0.995$, $W = 3$, $\alpha = 0.6$, number of landmarks per jumpcode: 3, chroma aligned to every 2 beats. The results can be seen in Table 8.1 where accuracy is simply the number of times we selected the right cover song. Note the drop from 11,794 to 176 jumpcodes on average per song.

As a comparison, we also report the accuracy from the published algorithm in [Ellis and Poliner, 2007], called “correlation” in the Table. This method is very intuitive, we compute the correlation between beat-aligned chroma representations, including rotations and time offsets. First we conclude that our method performs similarly. The subset is too small to claim more than that. Secondly, since this test appears easier than the one in [Ellis and Poliner, 2007], we wonder whether the cover songs are rather *difficult* in this subset, or maybe the chroma representation from The Echo Nest is less suited for cover recognition than the one use in [Ellis and Poliner, 2007] (e.g. the normalization is different).

Then, we confirm that the result we obtained on the 500 binary queries translate relatively well

²<http://www.columbia.edu/~tb2332/proj/coverongs.html>

	accuracy	#hashes
random	50.0%	-
jumpcodes 1	79.8%	11,794
jumpcodes 2	77.4%	176
correlation	76.6%	-

Table 8.1: Results on 500 binary tasks.

The #hashes fields represents, on average, the number of jumpcodes we get per track.

to a larger set, i.e. the 12,960 training cover songs. From now on we report the average position of the covers for each queries. On the subset, using the settings from “jumpcode 2”, the average position was 4,361. If we think of a query as doing 12,960 binary decisions (comparing the correct answer to every one of the decoys), this gives us an accuracy of 66.3%. We have 2,280,031 (song, jumpcode) pairs in the database for the train songs, this gives us an average of 176 jumpcodes per track.

Finally, over the whole million song dataset using the test set of the SecondHandSongs dataset, we get an average position of **308,369** with a standard deviation of **277,697**. To our knowledge, it is the first reported result on this dataset, and the first on any data set of such large scale. We note that the standard deviation is quite large, probably meaning that some covers are really faithful while others are totally different.

We take a closer look at the successful queries by looking at query pairs, i.e. a pair formed by a query song and a target song. A query pair’s ranking is the ranking of the target song when the dataset is queried using the query song. 120 tracks (0.68% of query pairs) ranked at 100th or better (top 0.01%); a random baseline would be expected to place fewer than 2 of the 17,771 queries at this level. In fact, 12 query tracks ranked their matches at #1 in a million tracks. Some of these turned out to be unfortunate duplicates, an existing problem of the MSD³. But some are actual success, e.g. the query pair “Wuthering Heights” by (*Kate Bush, Hayley Westenra*) or “I don’t like Mondays” by (*The Boomtown Rats, Bon Jovi / Bob Geldof*). This last pair is made of two different versions, both including *Bob Geldof*.

³<http://labrosa.ee.columbia.edu/millionsong/blog/11-3-15-921810-song-dataset-duplicates>

8.3 Conclusion

We presented a hashing-based method to efficiently index and retrieve cover songs. Its execution speed allows us to work with the Million Song Dataset, and the algorithm is relatively intuitive as a first benchmark. Unfortunately, the results, though better than random, leave considerable room for improvement. It is also unclear how our jumpcodes can be used for other applications, such as understanding the music harmonic space.

The next chapter presents a different approach, based on a higher level feature instead of the chromagram. As we will see, the execution time is similar, the results are better, and the feature is promising for other tasks.

Chapter 9

Fourier transform methods

In Chapter 8, we presented a hash-based method for cover song recognition. Another approach would be to project the entire song into a small fixed dimension space in which nearest neighbors are our candidate covers. Nearest neighbor methods are easy to parallelize and scale, and working with a fixed-dimensional representation (instead of a variable-length audio signal) is a great convenience.

Dissatisfied by the results presented in the previous chapter, we now focus on this new option. Beat-synchronous chroma representations form a relatively compact description that retains information relevant to covers, and may be cropped to a constant size. Unfortunately, direct comparison of chroma patterns using common metrics is poorly behaved (Section 7.2). Summarizing a song by extracting a few chroma patches¹ and comparing them with Euclidean distance gives unusable results for several reasons, including the acute sensitivity of direct comparison to temporal alignment. In order to obtain an efficient nearest-neighbor algorithm, we need a representation for the chroma patches with the following properties:

- representation vectors can be compared using a simple metric, e.g. Euclidean distance;
- representation vectors are compact, i.e. low-dimensional;
- representation must be robust to semitone rotations (musical transpositions);
- representation should be robust to different pattern offsets (time alignments).

¹ Unless stated otherwise, chromas now refer to beat-aligned chromas, and a patch is a $12 \times N$ matrix representing the chromas over a N -beat section. An example was given in Figure 8.2 (bottom figure).

The last condition would allow us to match patches without having to identify a universal time alignment which would be very difficult in a large set. Our candidate representation is the two-dimensional Fourier transform magnitude (2DFTM), a representation very similar to the one used in [Marolt, 2008]. The Fourier transform separates patterns into different levels of detail, which is useful for compacting energy (as in image compression schemes such as JPEG) and for matching in Euclidean space. Discarding the phase component provides invariance both to transposition (rotation) in the pitch axes and skew (misalignment) on the beat (time) axis. Thus, taking the 2DFTM of a chroma patch, we obtain a transformation of chroma features that makes Euclidean distance quite useful, even after dimensionality reduction through PCA. Our method encodes each track as a 50-dimensional vector and provides a large improvement over the hash code-based method [Bertin-Mahieux and Ellis, 2011]. On the SHS, this approach gives the best result reported so far.

The starting point of this chapter can be found in our paper [Bertin-Mahieux and Ellis, 2012].

9.1 Features

Taking the two-dimensional Fourier transform is a common task in digital image processing where it is useful for denoising and compression, among other things [González and Woods, 2008]. As illustrated in the examples of Figure 9.1, a single point in the 2D Fourier transform corresponds to a sinusoidal grid of a particular period and orientation in the image (transform) domain; more complex images are built up out of multiple sinusoid grids.

For an $N \times N$ image, the 2D Fourier transform is given by:

$$F(u, v) = \sum_{x, y} f(x, y) e^{-2j\pi \times (ux + vy)/N} \quad (9.1)$$

with $f(x, y)$ the value of the original image at the coordinates (x, y) . The magnitude at every position (u, v) is defined as:

$$\text{mag}(u, v) = \sqrt{\Re(F(u, v))^2 + \Im(F(u, v))^2} \quad (9.2)$$

with \Re returning the real part of a complex number, and \Im the imaginary part. Note that for visualization purposes, the bins are shifted so that the center of the axis system is in the middle of the image². In that representation, the transformed image is symmetrical around the origin.

²*fftshift* in MATLAB, *scipy.fftpack.fftshift* in Python

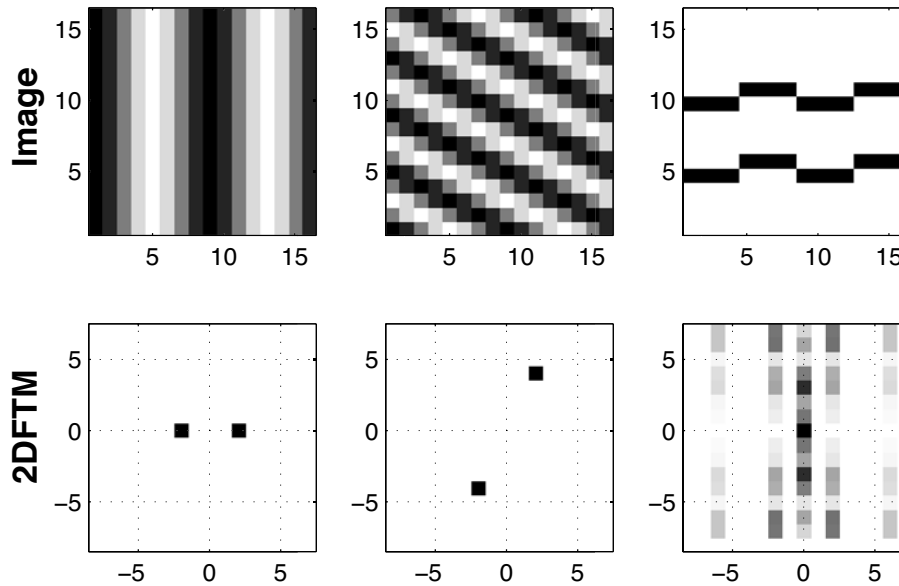


Figure 9.1: Examples of 2-dimensional Fourier transforms.

Figure 9.2 gives an example of the transform applied to a chroma patch.

9.2 Method

We will represent each song by a vector of fixed-length, defining a point in Euclidean space. The closer two feature vectors lie in this space, the more likely the songs are covers. The steps to compute this feature vector for a song are summarized as follows:

1. obtain chroma from the MSD;
2. resample onto beat grid;
3. apply power-law expansion;
4. extract FFT patches and keep magnitude;
5. take the median;

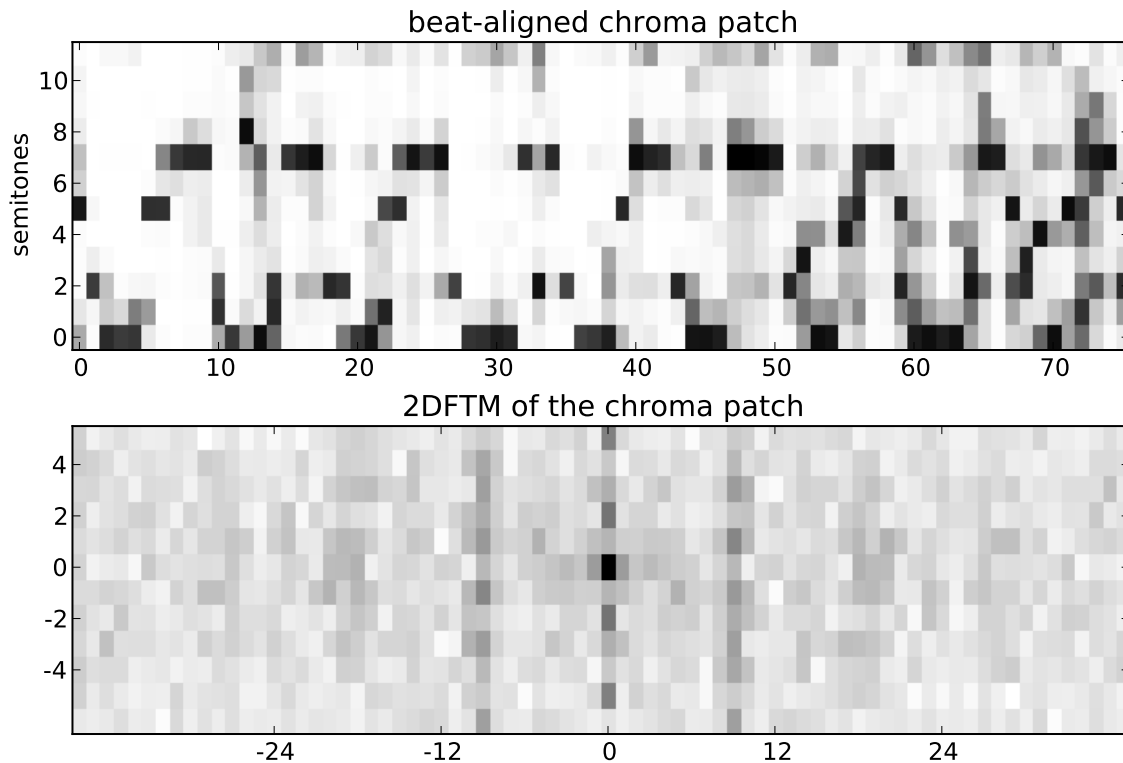


Figure 9.2: Beat-aligned chroma patch and corresponding 2DFTM.

6. apply PCA.

The first step is done using the code provided with the MSD. Beat estimation is also provided in the MSD, and the second step is mostly a linear scaling. The third step, in which we enhance the contrast between strong and weak chroma bins by raising the values to an exponent, was determined to be useful in our experiments. In the fourth step, we take the 2DFTM as shown in Figure 9.2 for every 75-beat long chroma patch. In the fifth step we keep the median, within each bin, across all the transformed patches. Finally, in the sixth step, we use PCA to reduce the dimensionality. PCA is done without normalization, the data are simply centered to zero and rotated. In our experiments PCA was computed on 250K songs that were not identified as covers (i.e., in neither the SHS train set nor the test set).

The fifth step is necessary because we want to represent all the many possible patches within a song with a single value; averaging was one possibility, but median appeared to give better robustness against extremes.

Many of the parameters above were chosen empirically. Presenting all of them would bear little interest. Simply note that the following parameters were chosen based on their ability to identify 500 train covers (see Subsection 8.1.4):

- patches of size 75 beats, we tried number of beats ranging from 8 to 100;
- median as opposed to average;
- raising to the power 1.96, we tried values between 0.25 and 3.0.

Note that working with a 75-beat long music segment is unusual; in previous work, a patch length of 8 or 12 beats is more common. Marolt mentions the risk that large time shifts can create fragments that are not musically similar [Marolt, 2008]. We believe this problem is alleviated by the long segments we consider.

The number of principal components (PC) we keep after PCA is also a parameter, but choosing the best one is more difficult. The number of PCs is a trade-off between accuracy and feature vector size (and hence speed). We believe 50 is the “best” trade-off, but we report results for other numbers of PCs. Still regarding PCA, since we use chroma patches of 75 beats, we have $12 \times 75 = 900$ principal components. Note that half of the components (450) are redundant due to the symmetry in the 2DFTM, and have a variance of zero associated with them.

Figure 9.2 illustrates an interesting fact; there is a lot of energy in column 9. It corresponds to a strong 8-beat period, since 8 beats are approximately one ninth of a 75-beat long patch. In western music, a strong beat every 4 (or some multiple of 4) beats is extremely common.

9.3 Results

As in Chapter 8, the parameters are first tuned on a subset of 500 binary tasks created within the SHS training set (we use the same 500 queries). The goal is: given a query song A and two songs B and C , find which of B or C is a cover of A . The result is the percentage of trials where the algorithm succeeds. We then present the results testing on the training set, mostly as a sanity check. Finally, we report result on the SHS test set using the full MSD.

In our previous Chapter, the main reported result was the average rank of a known cover given a query. For instance, on 1M songs, picking at random would give 500K. We again report

Method	accuracy
random	50.0%
pitch hist.	73.6%
correlation	76.6%
DTW	80.0%
jcodes 1	79.8%
jcodes 2	77.4%
2DFTM (full)	82.0%
2DFTM (200 PC)	82.2%
2DFTM (50 PC)	82.2%
2DFTM (10 PC)	79.6%
2DFTM (1 PC)	66.2%

Table 9.1: Results on 500 binary tasks.

PC is the number of principal components we retain after PCA. Empirically, 50 PC appears to be the best trade-off between accuracy and size.

this measure to permit comparison, but we also present *mean average precision* (mAP) due to a performance increase. Average rank can be misleading since it is dominated by the most difficult covers, of which there will always be a number, and hides differences in performance near the top of the ranking. mAP puts emphasis on results that rank high, i.e., the covers that are in the top k results. We also present the recall curves for a few algorithms (see Figure 9.3).

We introduce a new reference method: comparing pitch histograms. A pitch histogram is computed over a chromagram by summing the rows (semi-tones) and making the norm 1. This creates one 12-dimensional descriptor per song. This feature is very intuitive, it gives the importance of each semi-tone inside a song. If two songs (modulo a semi-tone rotation) have similar semi-tone content, they are likely covers.

We also compare with a DTW-based method using code from S. Ravuri³. This is our best attempt at comparing our methods with the state of the art as defined by MIREX. The 500 binary tasks and the 12,960-song training set are similar in size to what has been previously published. However, results are also very dependent on the features used. For instance, in [Serrà *et al.*, 2008],

³<http://infiniteseriousness.weebly.com/cover-song-detection.html>

the authors use a chromagram with up to 36 semi-tones instead of the 12 we possess. We also did not re-estimate the DTW parameters. Finally, results published based on MIREX results probably suffer from overfitting as explained in Section 7.4. Therefore, the comparison with DTW presented in Table 9.1 only tells us that we get a performance in the same ballpark. Furthermore, there is still a need for fast algorithms even if their performance is lower, as we explain below. The output of a first, fast algorithm can be the input of a second, slower algorithm.

Method	average rank	mAP
pitch hist.	4,032.9	0.0185
2DFTM (full)	3,096.7	0.0891
2DFTM (200 PC)	3,005.1	0.0948
2DFTM (50 PC)	2,939.8	0.0776
2DFTM (10 PC)	3,229.3	0.0265
2DFTM (1 PC)	4,499.1	0.0019

Table 9.2: Results on the training set (12,960 songs).

For average rank, lower is better. For mAP, higher is better.

Method	average rank	mAP
random	500,000	0.0000
pitch hist.	268,063	0.0016
jcodes 2	308,370	0.0021
2DFTM (200 PC)	180,304	0.0295
2DFTM (50 PC)	173,117	0.0200
2DFTM (10 PC)	190,023	0.0029
2DFTM (1 PC)	289,853	0.0000

Table 9.3: Results on 1M songs.

For average rank, lower is better. For mAP, higher is better.

As we see in Tables 9.2 and 9.3, the method based on our 2DFTM representation provides a significant improvement over the method in [Bertin-Mahieux and Ellis, 2011] for both measures. In particular, based on mAP, many more covers are ranked in the first few hundred results, which makes it much more valuable in a commercial system. Note that a second, slower step could be applied to the top k results, k being 1K or 10k. This is similar to the progressive refinement

described in [Yu *et al.*, 2009]. A good candidate for this second step would be the full system of [Serrà *et al.*, 2008].

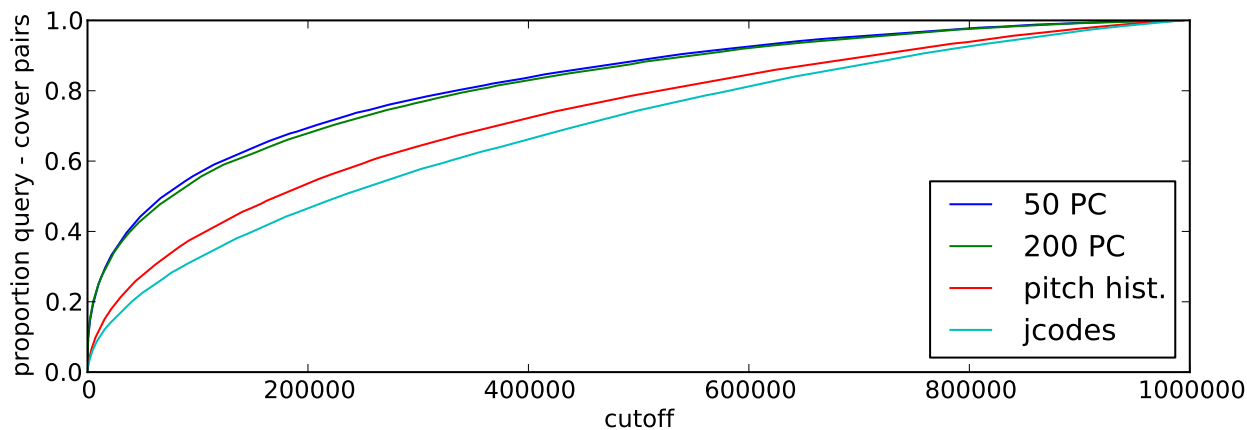


Figure 9.3: Recall using the 2DFTM method on the MSD.

Recall computed using $(query, cover)$ pairs on the full million songs for different systems. Legend is in order from best (upper left corner) to worst.

Using the system with 50 principal components, Figure 9.3 shows us that more than half of the covers are ranked in the top 100k and more than a quarter of them are in the top 10k. For 112 queries, the first song returned was a known cover. This includes the pairs $(Hayley Westenra, Kate Bush)$ performing “Wuthering Heights”, a match considered *easy* in Chapter 8. Note that this number could be greater since we do not count covers that are ranked second after another cover of the same clique, i.e. both are correct answers.

In terms of speed, with a 50-dimensional vector per song, ranking all one million tracks for all 1,726 test covers in Python took 1h 46min on a machine with plenty of RAM. This is around 3-4 seconds per query without any optimization or parallelization. Compared to the 2.7 hours per query of the DTW method, that is a $\sim 2,500x$ speedup.

9.4 A note on error measures

The first error measure we use in this work is the average rank of cover songs. It gives a global idea of how well the system performs, and the value moves quickly for algorithms whose performance is not that far off random. Thus, it is a good metric to get started and for proofs of concept.

As performance increases, the way we measure the quality of our ranking can be refined. As for most ranking systems, what matters is how many true positives are returned in the first few items. The exact number of top items we should consider is application-dependent. If the output of a ranking system is shown to a human user, only the first ten or twenty items really matter. In another setting, the output of our algorithm can be passed on to a second algorithm. This is the progressive refinement we suggest above. In such a case, the mAP measure is more appropriate.

It is important to understand that the error measure is part of the task definition. As the performance of algorithms will increase, methods will be tuned in respect to specific measures. We believe that average rank and mAP are appropriate metrics for academic settings, since they represent an overall performance on cover song recognition. However, for a commercial system that displays its result to users, a metric such as precision at k (sometimes called “top- k ”) makes more sense.

For future reference, in Table 9.4 we provide the results, measured with top- k , for the systems in Figure 9.3. We compute top- k as follow. Given k , queries numbered from 1 to N , and $tp_q(k)$ the number of true positives ranked in the top k for query q , we have:

$$\text{top-}k = \frac{\sum_{q=1}^N tp_q(k)}{Nk}$$

One problem with top- k occurs when we have fewer true positives than k . For instance, if for a given query we know only of 5 covers and we measure top-50, we cannot obtain a result above 10%. This upper bound is dataset-dependent, and is also reported in Table 9.4.

Method	$k = 1$	$k = 10$	$k = 50$	$k = 100$
upper bound	100.00%	31.31%	6.76%	3.38%
pitch hist.	0.17%	0.05%	0.03%	0.02%
jcodes 2	0.23%	0.07%	0.03%	0.02%
2DFTM (200 PC)	3.25%	1.05%	0.34%	0.20%
2DFTM (50 PC)	2.15%	0.74%	0.26%	0.17%

Table 9.4: Top- k results on 1M songs.

Algorithms are the same as in Figure 9.3. Upper bound is the maximum theoretical value we can get given our dataset, details in Section 9.4. For top- k , higher is better.

9.5 Understanding the 2DFTM feature

We provided some insight on why the 2DFTM features, derived from the chroma features, can be used to identify cover songs. In this section, we try to explain the properties of the feature further. Questions we try to answer include:

- why is the 2DFTM space seem better behaved?
- can we aggregate the feature over a whole song?
- can we do better than comparing 2DFTM patches using Euclidean distance?

9.5.1 Clustering features

One reason why comparing chroma was difficult is that adding a small time offset creates a large difference between two patterns. With the 2DFTM, adding a time offset of one would only create a “border effect”: since each bin corresponds to a frequency computed over the whole 2D image, changing the edges has limited results.

We try to demonstrate that effect using the idea of clustering stability, inspired by [Lange *et al.*, 2004]. Given N samples from a common distribution, we can split them in half, cluster each half using k -means, and see how the clustering from the first half predicts the second half. To present this with more details:

1. split the samples in two equal size sets;
2. cluster each sets of $N/2$ samples into M clusters, each cluster receives a unique label $1, \dots, M$ for its set;
3. using vector quantization, associate the samples from the second set to the clusters in the first set;
4. for a given permutation of the labels in the second set, a sample is misclassified if its label from the first and second set are different;
5. find E , the minimum number of misclassifications over all permutations;
6. the proportion of misclassifications is defined as E/N and is inversely proportional to stability.

In short, samples that come from a very stable distribution should create very similar clusters in any subset of the samples. In our case, we cluster all the frames in a song. Our experiment is performed over the whole SHS training set and is summarized in figure 9.4. We cluster beat-aligned chromas patches, 2DFTM patches and 2DFTM patches projected onto 50 dimensions via PCA (patches are 75 beats long). As expected, we have better stability (fewer misclassifications) for 2DFTM features. This demonstrates that two similar patches with different offsets will be more different according to Euclidean distance if they are represented by beat-aligned chroma than by 2DFTM.

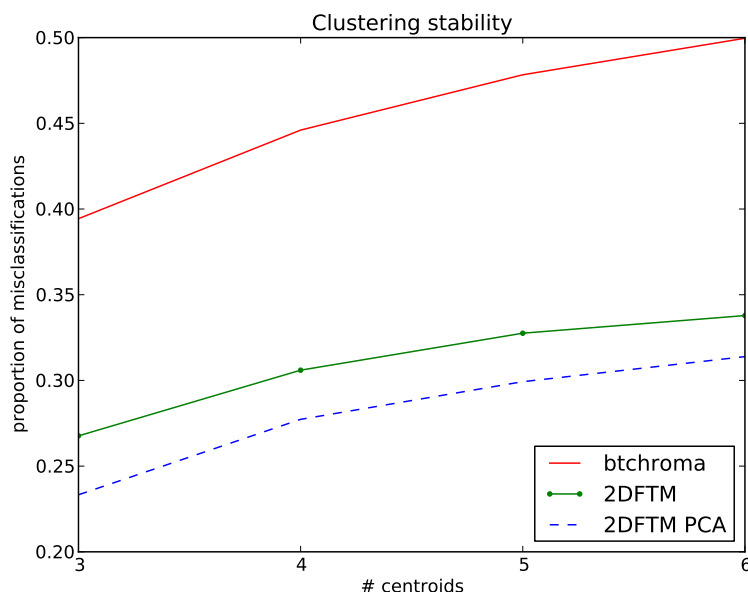


Figure 9.4: Cluster stability for three features.

Compare clustering stability through k -means with 3 to 6 centroids. Features are beat-aligned chroma, 2DFTM, and 2DFTM after PCA with 50 principal components. Lower is better.

9.5.2 Comparing distributions of 2DFTM features

In our experiments on the MSD, we take the median of the 2DFTM patches across a song. The intuition behind this is that it represents the typical amount of a certain 2D sinusoid in this song. That said, it might be an oversimplification. When looking at an animation visualization the evolution of 2DFTM patches during a song, we see significant variations and taking the median might lose a lot of information.

Thus, we can consider the distribution of the 2DFTM in a song by keeping the mean and variance of every bin. To compare two distributions (two songs), we use the symmetric Kullback-Leibler divergence (i is the bin index):

$$D_{KL}(P \parallel Q) = \frac{1}{2} \times \left[\sum_i \ln \left(\frac{P(i)}{Q(i)} \right) P(i) \right] + \frac{1}{2} \times \left[\sum_i \ln \left(\frac{Q(i)}{P(i)} \right) Q(i) \right]$$

The goal is to see if taking a more complex representation than the median can improve cover song recognition results.

The numbers in Table 9.5 seem to show that looking at the 2DFTM distribution does not improve results. The difference is small ($\sim 5\%$), but taking the median gives the best performance in our preliminary test.

Method	accuracy
random	50.0%
median + Euclidean	82.0%
mean/var + KL	77.0%
mean/var + KL (50 PC)	77.4%
mean/var + KL (10 PC)	76.2%

Table 9.5: Results on using mean and variance.

Results on 500 binary tasks. PC is the number of principal components we retain after PCA.

Note that we also looked at the distribution in the PCA space, with 50 and 10 principal components. The distribution of individual bins in that space could have been more meaningful, since each bin is a linear aggregation of many bins in the original 2DFTM feature. However, this does not yield any significant difference in our experiment.

9.5.3 Importance of the phase

So far in this work, we used the magnitude of the 2D Fourier transform of the chroma patches. Removing the phase is an easy way to become invariant to time and semitone offsets. However, we lose more than just those two invariance since the relative positions of the 2D sinusoids are also embedded in the phase. In this section, we try to retain some of that information to see if it can improve cover song recognition results.

The idea is to *normalize* the phase so it is invariant to time and semitone offset by taking the second derivative. One problem is the modulo 2π of the phase. To simplify the issue, we divide by 2π so the values are the same modulo 1. Then, by applying the following derivation kernel (second derivative in both direction):

$$\begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & -2 & 1 \end{bmatrix} = \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

and keeping the decimal part, we obtain a second derivative and sidestep the modulo issue.

The idea is similar for integration, we accumulate from left to right and right to left, followed by top down than bottom up. This procedure inverse the derivation above⁴

Once we have derived and reintegrated the phase, we obtain a *normalized* phase patch that would be the same for a chroma patch and its rolled version. However, not all phase values are created equal: the phase of a sinusoid that has a very low magnitude is likely irrelevant. Thus, we only consider half⁵ of the phase values, the ones associated with the highest magnitudes.

To compare a normalized phase patch, we use the Euclidean distance. We take into account the modulo 1 (values are still divided by 2π), distance being:

$$d_{1,2} = \sqrt{\sum_i dx_i^2}$$

with $dx_i = \min(|x_{i,1} - x_{i,2}|, 1 - |x_{i,1} - x_{i,2}|)$

If a phase value is discarded in one patch (because of a low magnitude) and not in the other patch, we consider it equivalent to a distance of 0.5. In our experiment, we obtain a distance for the phase, one for the magnitude as before, and we sum them to compute a distance between two patches. The results can be seen in Table 9.6. Although the difference is small (5%), we conclude that using the phase on top of the magnitude is cumbersome for no apparent gain.

⁴ Proof in 1D: for the second derivative, we applied an FIR filter with transfer function $(z-1)^2/z$ (thus with 2 zeros at 1). To integrate, we take an IIR function with two poles at 1, its transfer function is $z/(z-1)$. We apply it twice (with reverse indexes the second time), the new transfer function is $(z/(z-1))(z^{-1}/(z^{-1}-1)) = -z/(1-z)^2$. This is the inverse of the transfer function for the second derivative.

⁵We did try other proportions, 50% seemed to yield the best results.

Method	accuracy
random	50.0%
only magnitude	82.0%
phase and magnitude	77.0%

Table 9.6: Results on using phase.

Results on 500 binary tasks with and without using a *normalized* phase.

9.5.4 Beat-alignment for 2DFTM features

We have explained in Section 2.4 that for music features, it makes sense to have one frame per beat. However, the beat of a given song is not a perfectly defined concept. For instance, on a given song, one listener might tap his foot on every note while another listener might tap every half note. Neither of them are wrong a priori. This kind of variation, tapping at a fraction or a multiple of the reference beat, is common for automatic beat trackers too.

Beat-aligned chromagrams depend on the output of an automatic beat tracker, thus they suffer from this kind of problem. For a given song, if the beats are identified at every 4 notes, and for one very similar cover, at every 2 notes, the chromagram of the cover will look like an expanded version of the first chromagram. One easy way to see this is that the chromagram will be twice as long since it has twice as many beats.

To ameliorate this, we can compute more than one chromagram per song (a similar approach was used in [Ellis and Poliner, 2007]). For example, we could have a chromagram with one beat per frame (the usual one), one with 2 beats per frame, and one with half a beat per frame. Although this gives more chances for a cover version to match the original, it does add to the amount of features and the complexity of computing the distance between 2 songs. Since each song is now represented by a set of 2DFTM features from a set of chromagrams, we will define the distance between two songs as the minimum distance between 2DFTM features from each of these songs. For example, we could match a song using features computed with one beat per frame to another song with features computed with two beats per frame.

We perform a preliminary experiment with this idea, the result can be seen in Table 9.7. It does appear that computing multiple 2DFTM features, from multiple chromagrams with different numbers of beat per frame, improves our results. We leave for future work the task of scaling this

method, including implementing the more complex distance function between sets efficiently. It is important, however, to be aware of the influence of the underlying beat tracker on the 2DFTM features.

Method [beat per frame]	accuracy
random	50.0%
2DFTM [1]	82.0%
2DFTM [1, 2]	83.4%
2DFTM [0.5, 1, 2]	82.2%
2DFTM [1, 2, 4]	83.4%

Table 9.7: Results for multiple beat-alignments.

Computed using the 500 binary tasks. The numbers in bracket are the beat per frame we use to compute chromagrams, i.e., for 2DFTM [1,2], it means we have two feature vectors from two chromagrams. 2DFTM [1] is the usual 2DFTM method (without PCA).

9.5.5 Value normalization

We evaluated the 2DFTM feature as a whole, through cover song recognition, but it is rather difficult to evaluate the importance of each value of the feature. Our typical patch has $12 \times 75 = 900$ dimensions, each value (or *bin*) representing a particular sinusoid⁶. In Section 9.2, we apply PCA with no value normalization (e.g. taking the z -score). In practice, it puts emphasis on the “central 2DFTM values”, the bins corresponding to low-frequency sinusoids, because those values tend to be greater.

We can try to understand the value of this decision. MLR code [McFee and Lanckriet, 2010] can learn a linear transformation that creates a better space for ranking. Given classes like cover song cliques, it learns a transformation that tries to make covers closer to one another, while pushing away non covers. This is an example of metric learning.

We apply MLR to the full 900-dimensions patches from Section 9.2, but we restrict the transformation to a diagonal matrix. In effect, we learn a weight on the 900 bins, a higher weight meaning more discrimination power to find covers. For our experiments on the training set, we use the 5,854

⁶Given the symmetry about the origin, there are actually only 451 independent values in a 2DFTM patch.

test cover songs to train. The result is shown in Figure 9.5. We can see that “corner values” are underrepresented.

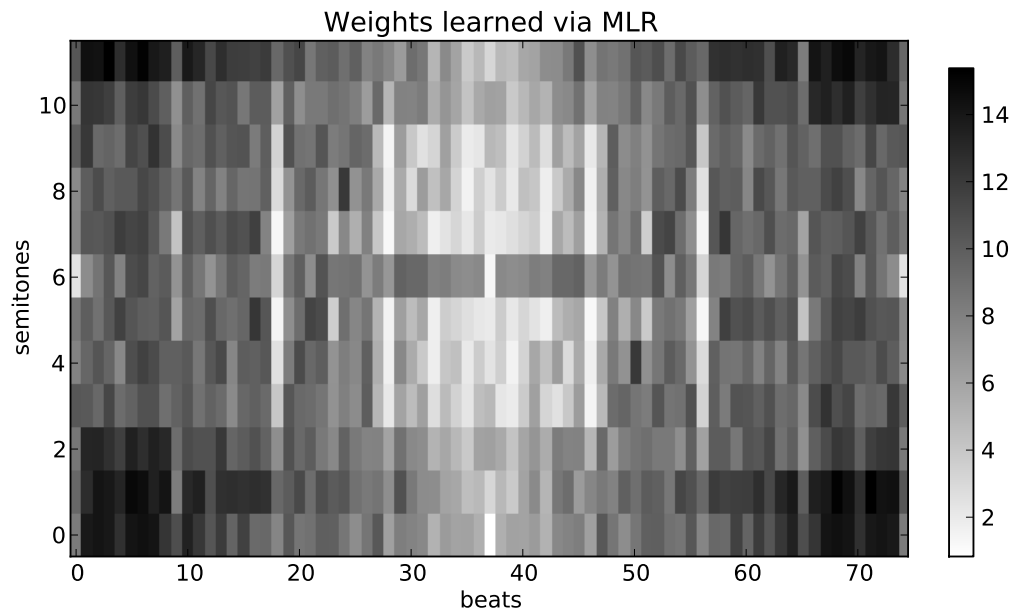


Figure 9.5: Learned weights for 2DFTM features.

Weights learned through MLR [McFee and Lanckriet, 2010] on the test cover cliques. Parameters were $C = 1000$, loss is AUC , the learned projection is diagonal.

The conclusion is that the patches could use some normalization before we apply PCA. We experiment with z -scoring, i.e. removing the sample mean and dividing by the sample standard deviation. The new weights learned through MLR can be seen in Figure 9.6. The difference between low and high weights is greatly reduced, and there is no clear pattern (e.g. in the corners, or in the center). Note that for both Figure 9.5 and 9.6, we ran the MLR algorithm 10 times over random subsamples of the data and took the average. This avoids high values that could come from overfitting.

Everything else in our method described in Section 9.2 remains identical. On the 500 binary queries, using z -score improves the result from 82.2% to 83.0%. Results on the training set are shown in Table 9.8. We observe a small improvement overall.

Now that we have improved on the normalization, it can also be of interest to show the principal components learned through PCA. Figure 9.7 shows the first 3 ones. The main content seems to

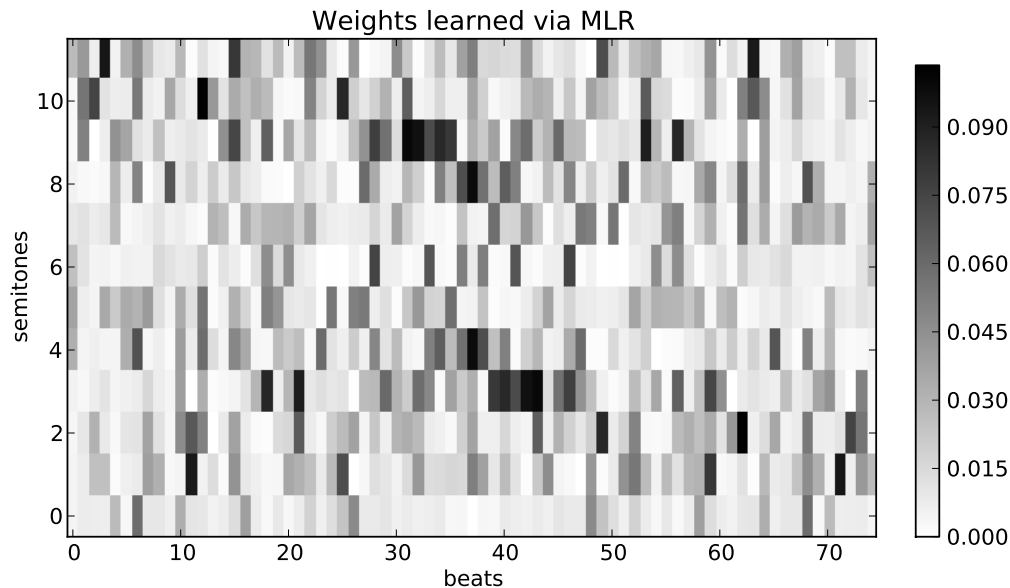


Figure 9.6: Learned weights for 2DFTM features after z -scoring.

Weights learned through MLR [McFee and Lanckriet, 2010] on the test cover cliques. Parameters were $C = 1000$, loss is AUC , the learned projection is diagonal.

be rhythmic, with the second principal component emphasizing columns ± 19 (corresponding to 4-beat periodicity), and the third component balancing the intensity around column 9 (for 8-beat periodicity).

Method	average rank	mAP
2DFTM (200 PC)	3,005.1	0.0948
2DFTM (50 PC)	2,939.8	0.0776
2DFTM (200 PC) z -scoring	2,976.1	0.0909
2DFTM (50 PC) z -scoring	2,911.1	0.0812

Table 9.8: Results on the training set with z -scoring.

z -scoring applied before computing PCA. For average rank, lower is better. For mAP, higher is better.

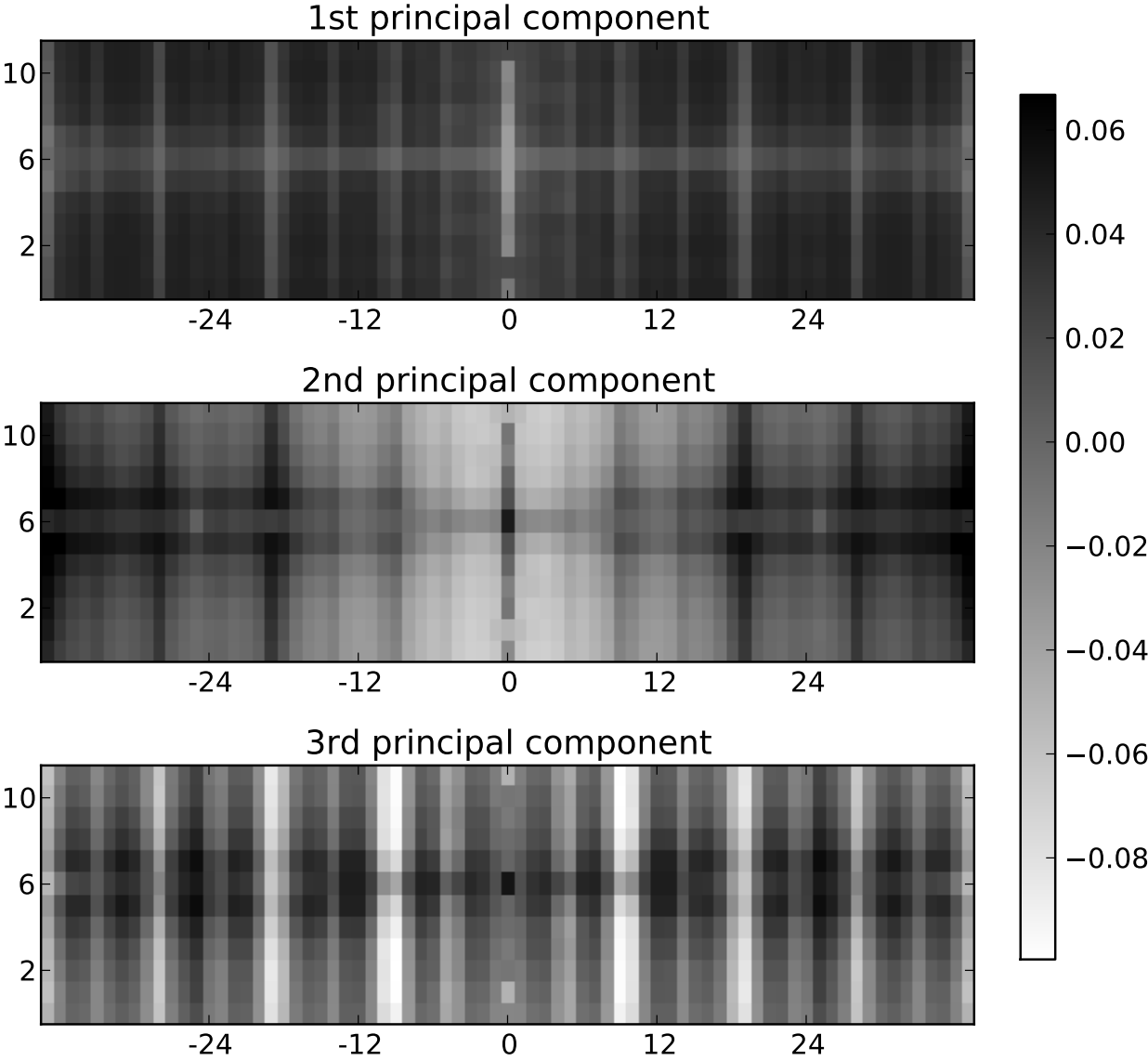


Figure 9.7: Principal components.
PCA trained on 2DFTM patches with z -scoring.

Part IV

Conclusions

Chapter 10

Conclusions

We have discussed the creation of a very large collection of music-related data (Part II) and we investigated two methods to perform cover song recognition on such a collection (Part III). The MSD, with its wide diversity of data ranging from lyrics to user feedback, can be used to study many MIR questions. We see this as our contribution to the field to help it embrace this new era of big data. Cover song recognition is a perfect case study to illustrate the interesting questions, and practical challenges posed by large-scale music problems. The two methods we showed, one relying on hashing and the other on fixed-dimensional representation, are typical solutions for data-intensive problems.

Using this as a starting point, we now mention different directions of research we would like to investigate in the near future. Some are extensions of our work on cover songs. Others simply rely on data available in the MSD. We do however believe that more and more applications will rely on heterogeneous sources of data, and many MIR problems should slowly merge as larger collections become available. Thus, features that were good for cover song recognition could be used for tagging, recommendation, etc.

10.1 Cover songs and manifolds

Regarding our cover song recognition work, there are several experiments on which we would like to follow up. On hashing, researchers at Google have developed the intervalgram [Lyon *et al.*, 2012; Walters *et al.*, 2012], a chroma-like feature derived from a stabilized auditory image. This feature is

designed to support locality-sensitive hashing, thus making it a good candidate for large databases. We do not have access to this feature in the MSD or elsewhere in large quantities, but we would like to understand how different the feature is. If an intervalgram can be derived from chromas, we could leverage a lot of research developed by the Google Music team.

Regarding the 2DFTM features in Chapter 9, we believe they are sufficiently general to be useful for tasks other than cover song recognition. In particular, they could help identify similar patterns that are not covers, but rather influences. Music is not reinvented at every song, especially in terms of harmony. We should be able to identify trends that might be associated with music eras or certain musicians. We believe that defining those trends should amount to finding a harmonic manifold of music, a subspace where neighbors are song sections with similar melodies and harmonies.

Leveraging the large collection of covers in the SHS, we would like to investigate methods that specifically learn the relationship between covers. It could take the form of learning a metric that pushes covers closer to one another while pushing away other songs¹ At another level of complexity, it could focus on some aspect of the audio based on the song genre. After all, what matters most for pop covers is probably the melody, while the chord sequence is more important than the solo for jazz sessions.

10.2 Using the Million Song Dataset

The Million Song Dataset is the largest, heterogeneous collection of MIR data available to researchers. Most tasks that have been studied in the past (automatic tagging of music, similarity, etc) should be re-investigated in the context of a million data points. What really interests us are tasks that could not be studied before because of the lack of data.

¹MLR code [McFee and Lanckriet, 2010] from Section 9.5.5 is a good starting point.

First, we hope to see more work about managing large music collections. Metadata matching has been largely ignored by the research community, but it is an extremely difficult problem in practice. As an example, the metadata below applies to the same song, first in the Musicbrainz (BRZ) database, then in the MSD. The first “-” represents the separation between artist and song title.

BRZ: Edu K - Sex-0-Matic (feat. Deize Tigrona) (DJ Mavi & DJ Sany mix)

MSD: EDU K feat. Deize Tigrona - Sex-0-Matic (DJ Mavi+DJ Sany Baile Funk Mix)

There is enough overlap that a human could be confident about the match. However, this is not something that can be solved by a string edit distance and a few rules. Another example involving classical music and multiple languages is below. For non-Russian readers, the Musicbrainz artist is “Pyotr Ilyich Tchaikovsky”.

BRZ: Пётр Ильич Чайковский - Piano Concerto No. 1 in B-flat
 minor, Op. 23: II. Andantino semplice - Prestissimo
 - Tempo I (feat. conductor: Pietro Argento,
 piano: Van Cliburn)

MSD: Van Cliburn - Concerto No 1 In B Flat Minor_ Op23:
 Second Movement - Andante Semplice; Prestissimo;
 Tempo Primo

We strongly believe that these matching issues should be discussed in research publications. At the moment, each organization or researcher reinvents the wheel and hacks a frequently-unsatisfying solution.

Second, we should use this collection to perform recommendation and understand which features matter. [McFee and Lanckriet, 2012] do something similar for playlist generation where they show that tags are more useful than release year, which is more useful than lyrics.

Finally, we have mentioned in the previous section the idea of discovering harmonic trends in the MSD. By introducing information about the release year of songs, we can consider studying music evolution. We have seen year prediction in [Bertin-Mahieux *et al.*, 2011a], and [Serrà *et al.*, 2012] claim to have shown that music is becoming more dull. We believe both are baby steps, albeit in the right direction. Understanding music evolution from a data-driven point of view, without cultural inputs from musicologists, would be a tremendous achievement in music technology. Computers could finally claim to understand music on a level similar to that of a human.

Part V

Bibliography

Bibliography

- [Achlioptas, 2001] Dimitris Achlioptas. Database-friendly random projections. In *Proceedings of the twentieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '01, pages 274–281, New York, NY, USA, 2001. ACM.
- [Aioli, 2012] F. Aioli. A preliminary study on a recommender system for the million songs dataset challenge. In *Proceedings of the ECAI Workshop on Preference Learning: Problems and Application in AI*, Montpellier, France, 2012.
- [Arthur and Vassilvitskii, 2007] D. Arthur and S. Vassilvitskii. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1035. Society for Industrial and Applied Mathematics, 2007.
- [Aucouturier and Pachet, 2003] J.J. Aucouturier and F. Pachet. Representing musical genre: A state of the art. *Journal of New Music Research*, 32(1):83–93, 2003.
- [Bartsch and Wakefield, 2001] M. A. Bartsch and G. H. Wakefield. To catch a chorus: using chroma-based representations for audio thumbnailing. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Platz, NY, October 2001.
- [Bellman, 1961] R.E. Bellman. *Adaptive control processes: a guided tour*. Rand Corporation Research studies. Princeton University Press, 1961.
- [Berenzweig *et al.*, 2004] A. Berenzweig, B. Logan, D.P.W. Ellis, and B. Whitman. A large-scale evaluation of acoustic and subjective music-similarity measures. *Computer Music Journal*, 28(2):63–76, 2004.

- [Berti, 2009] J. Berti. Copyright infringement and protection in the internet age. *IT professional*, 11(6):42–45, 2009.
- [Bertin-Mahieux and Ellis, 2011] T. Bertin-Mahieux and D.P.W. Ellis. Large-scale cover song recognition using hashed chroma landmarks. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Platz, NY, 2011.
- [Bertin-Mahieux and Ellis, 2012] T. Bertin-Mahieux and D.P.W. Ellis. Large-scale cover song recognition using the 2d fourier transform magnitude. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
- [Bertin-Mahieux *et al.*, 2008] T. Bertin-Mahieux, D. Eck, F. Maillet, and P. Lamere. Autotagger: a model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research, special issue: "From genres to tags: Music Information Retrieval in the era of folksonomies."*, 37(2), June 2008.
- [Bertin-Mahieux *et al.*, 2010a] T. Bertin-Mahieux, D. Eck, and M. Mandel. Automatic tagging of audio: The state-of-the-art. In Wenwu Wang, editor, *Machine Audition: Principles, Algorithms and Systems*, pages 334–352. IGI Publishing, 2010.
- [Bertin-Mahieux *et al.*, 2010b] T. Bertin-Mahieux, R. Weiss, and D. Ellis. Clustering beat-chroma patterns in a large music database. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010.
- [Bertin-Mahieux *et al.*, 2011a] T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Bertin-Mahieux *et al.*, 2011b] T. Bertin-Mahieux, G. Grindlay, R. Weiss, and D. Ellis. Evaluating music sequence models through missing data. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011.
- [Bimbot *et al.*, 2011] F. Bimbot, E. Deruty, G. Sargent, and E. Vincent. Methodology and resources for the structural segmentation of music pieces into autonomous and comparable blocks. In

- Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Bishop, 2006] C. Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, 2006.
- [Burgoyne *et al.*, 2011] J. Burgoyne, J. Wild, and I. Fujinaga. An expert ground-truth set for audio chord recognition and music analysis. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Casey and Slaney, 2007] M. Casey and M. Slaney. Fast recognition of remixed music audio. In *Proceedings of the 2007 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Signal Processing Society, 2007.
- [Cohen *et al.*, 2009] J. Cohen, B. Dolan, M. Dunlap, J.M. Hellerstein, and C. Welton. Mad skills: new analysis practices for big data. *Proceedings of the VLDB Endowment*, 2(2):1481–1492, 2009.
- [Cooley and Tukey, 1965] J.W. Cooley and J.W. Tukey. An algorithm for the machine calculation of complex fourier series. *Math. Comput*, 19(90):297–301, 1965.
- [Dasgupta and Gupta, 1999] Sanjoy Dasgupta and Anupam Gupta. An elementary proof of the Johnson-Lindenstrauss lemma. Technical Report TR-99-006, UC Berkeley, Berkeley, CA, 1999.
- [Donoho, 2000] D.L. Donoho. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000.
- [Downie *et al.*, 2008] J.S. Downie, M. Bay, A.F. Ehmann, and M.C. Jones. Audio cover song identification: Mirex 2006-2007 results and analyses. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, pages 468–473, 2008.
- [Downie, 2008] J.S. Downie. The music information retrieval evaluation exchange (2005–2007): A window into music information retrieval research. *Acoustical Science and Technology*, 29(4):247–255, 2008.
- [Duda *et al.*, 2000] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley-Interscience Publication, 2000.
- [EchoNest,] The Echo Nest Analyze, API, <http://developer.echonest.com>.

- [Eck *et al.*, 2008] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008.
- [Ellis and Poliner, 2007] D. Ellis and G. Poliner. Identifying cover songs with chroma features and dynamic programming beat tracking. In *Proceedings of the 2007 International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE Signal Processing Society, 2007.
- [Ellis, 2007] D. Ellis. Classifying music audio with timbral and chroma features. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*, 2007.
- [Fazekas *et al.*, 2010] G. Fazekas, Y. Raimond, K. Jacobson, and M. Sandler. An overview of semantic web activities in the omras2 project. *Journal of New Music Research*, 39:295–311, 2010.
- [Fujishima, 1999] T. Fujishima. Realtime chord recognition of musical sound: a system using common lisp music. In *Proceedings of the International Computer Music Conference (ICMC)*, 1999.
- [Gersho and Gray, 1991] A. Gersho and R. Gray. *Vector quantization and signal compression*. Kluwer Academic Publishers, Norwell, MA, USA, 1991.
- [Gold *et al.*, 2011] B. Gold, N. Morgan, and D. Ellis. *Speech and Audio Signal Processing: Processing and Perception of Speech and Music*. Wiley-Interscience, 2011.
- [Gómez, 2006] E. Gómez. Tonal description of polyphonic audio for music content processing. *INFORMS Journal on Computing*, 18(3):294–304, 2006.
- [González and Woods, 2008] R.C. González and R.E. Woods. *Digital image processing*. Pearson/Prentice Hall, 2008.
- [Goto *et al.*, 2002] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka. Rwc music database: Popular, classical, and jazz music databases. In *Proceedings of the 3th International Conference on Music Information Retrieval (ISMIR 2002)*, volume 2, pages 287–288, 2002.

- [Goto *et al.*, 2003] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka. Rwc music database: Music genre database and musical instrument sound database. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, volume 3, pages 229–230, 2003.
- [Goto, 2004] M. Goto. Development of the rwc music database. In *Proceedings of the 18th International Congress on Acoustics (ICA 2004)*, volume 1, pages 553–556, 2004.
- [Harte, 2005] M. Harte, C.; Sandler. Automatic chord identification using a quantised chromagram. In *Audio Engineering Society Convention 118*, 5 2005.
- [Hu *et al.*, 2003] N. Hu, R.B. Dannenberg, and G. Tzanetakis. Polyphonic audio matching and alignment for music retrieval. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, pages 185–188, New Platz, NY, October 2003.
- [Indyk and Motwani, 1998] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing, STOC '98*, pages 604–613, New York, NY, USA, 1998. ACM.
- [İzmirli and Dannenberg, 2010] Ö. İzmirli and R.B. Dannenberg. Understanding features and distance functions for music sequence alignment. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010.
- [Jannach *et al.*, 2010] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich. *Recommender systems: an introduction*. Cambridge University Press, 2010.
- [Jehan and DesRoches, 2011] T Jehan and D. DesRoches. *Analyzer documentation*. The Echo Nest, Sept. 2011. Version 3.08.
- [Jehan, 2005] T. Jehan. *Creating music by listening*. PhD thesis, Massachusetts Institute of Technology, 2005.
- [Jensen *et al.*, 2008] J.H. Jensen, M.G. Christensen, and S.H. Jensen. A chroma-based tempo-insensitive distance measure for cover song identification using the 2d autocorrelation function. MIREX cover song identification contest, 2008.
- [Jiang *et al.*, 2011] N. Jiang, P. Grosche, V. Konz, and M. Müller. Analyzing chroma feature types for automated chord recognition. In *Proceedings of the 42nd AES Conference*, 2011.

- [Johnson and Lindenstrauss, 1984] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mapping into hilbert space. In *Conf. in modern analysis and probability*, volume 26 of *Contemporary Mathematics*, pages 189–206. American Mathematical Society, 1984.
- [Kim and Narayanan, 2008] S. Kim and S. Narayanan. Dynamic chroma feature vectors with applications to cover song identification. In *Multimedia Signal Processing, 2008 IEEE 10th Workshop on*, pages 984–987. IEEE, 2008.
- [Knuth, 1973] D.E. Knuth. *The Art of Computer Programming, Volume III: Sorting and Searching*. Addison-Wesley, 1973.
- [Kullback and Leibler, 1951] S. Kullback and R. A. Leibler. On information and sufficiency. *Annals of Mathematical Statistics*, 22:49–86, 1951.
- [Kurth and Müller, 2008] F. Kurth and M. Müller. Efficient index-based audio matching. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):382–395, 2008.
- [Lamere, 2008] P. Lamere. Semantic tagging and music information retrieval. *Journal of New Music Research, special issue: "From genres to tags: Music Information Retrieval in the era of folksonomies."*, 2008.
- [Lange *et al.*, 2004] T. Lange, V. Roth, M.L. Braun, and J.M. Buhmann. Stability-based validation of clustering solutions. *Neural computation*, 16(6):1299–1323, 2004.
- [Langford *et al.*, 2007] J. Langford, L. Li, and A. L. Strehl. Vowpal wabbit (fast online learning), 2007. <http://hunch.net/~vw/>.
- [Law and von Ahn, 2009] E. Law and L. von Ahn. Input-agreement: a new mechanism for collecting data using human computation games. In *Proceedings of the 27th international conference on Human factors in computing systems*, pages 1197–1206. ACM, 2009.
- [Law *et al.*, 2007] E. Law, L. von Ahn, R. Dannenberg, and M. Crawford. Tagatune: a game for music and sound annotation. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*, 2007.

- [Le *et al.*, 2011] Q. V. Le, R. Monga, M. Devin, G. Corrado, K. Chen, M.'A. Ranzato, J. Dean, and A. Ng. Building high-level features using large scale unsupervised learning. *CoRR*, abs/1112.6209, 2011.
- [Lee, 2006] K. Lee. Automatic chord recognition from audio using enhanced pitch class profile. In *Proc. of the International Computer Music Conference (ICMC), New Orleans, USA, 2006*.
- [Levy, 2011] M. Levy. Improving perceptual tempo estimation with crowd-sourced annotations. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Logan, 2000] B. Logan. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, volume 28, page 5, 2000.
- [Lyon *et al.*, 2012] R.F. Lyon, T.C. Walters, D. Ross, et al. Intervalgram representation of audio for melody recognition, April 2012.
- [MacQueen, 1967] J.B. 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. University of California Press, 1967.
- [Macrae and Dixon, 2012] R. Macrae and S. Dixon. Ranking lyrics for online search. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
- [Maillet *et al.*, 2009] F. Maillet, D. Eck, G. Desjardins, and P. Lamere. Steerable playlist generation by learning song similarity from radio station playlists. In *Proceedings of the 10th International Conference on Music Information Retrieval (ISMIR)*, 2009.
- [Mandel and Ellis, 2008] M. Mandel and D. Ellis. Multiple-instance learning for music information retrieval. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, 2008.
- [Marolt, 2008] M. Marolt. A mid-level representation for melody-based retrieval in audio collections. *Multimedia, IEEE Transactions on*, 10(8):1617–1625, 2008.

- [Martin *et al.*, 2012] B. Martin, D.G. Brown, P. Hanna, and P. Ferraro. Blast for audio sequences alignment: a fast scalable cover identification tool. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
- [McAdams and Bregman, 1979] S. McAdams and A. Bregman. Hearing musical streams. *Computer Music Journal*, 3(4):26–60, 1979.
- [McFee and Lanckriet, 2010] B. McFee and G. Lanckriet. Metric learning to rank. In *Proceedings of the 27th International Conference on Machine Learning (ICML'10)*, June 2010.
- [McFee and Lanckriet, 2011] B. McFee and G. Lanckriet. The natural language of playlists. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [McFee and Lanckriet, 2012] B. McFee and G. Lanckriet. Hypergraph models of playlist dialects. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
- [McFee *et al.*, 2012] B. McFee, T. Bertin-Mahieux, D. Ellis, and G. Lanckriet. The million song dataset challenge. In *Proc. of the 4th International Workshop on Advances in Music Information Research (AdMIRe '12)*, April 2012.
- [Mika *et al.*, 1999] S. Mika, B. Schölkopf, A.J. Smola, K.R. Müller, M. Scholz, and G. Rätsch. Kernel pca and de-noising in feature spaces. *Advances in neural information processing systems*, 11(1):536–542, 1999.
- [Miotto and Orio, 2008] R. Miotto and N. Orio. A music identification system based on chroma indexing and statistical modeling. In *Proceedings of the 9th International Conference on Music Information Retrieval (ISMIR 2008)*, 2008.
- [Miotto *et al.*, 2010] R. Miotto, N. Montecchio, and N. Orio. Content-based cover song identification in music digital libraries. In *Digital Libraries*, pages 195–204. Springer, 2010.
- [Mitchell, 1997] T.M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., New York, NY, USA, 1 edition, 1997.

- [Mitra, 2006] S.K. Mitra. *Digital signal processing: a computer-based approach*. McGraw-Hill Series in Electrical and Computer Engineering. McGraw-Hill, 2006.
- [Müller *et al.*, 2005] M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. In *Proceedings of the 6th International Conference on Music Information Retrieval (ISMIR 2005)*, London, UK, 2005.
- [Musicbrainz, 2012] Musicbrainz: a community music metadatabase, April. 2012. MusicBrainz is a project of The MetaBrainz Foundation, <http://metabrainz.org/>.
- [Narayanan and Shmatikov, 2008] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.
- [Oliver *et al.*, 1948] B.M. Oliver, J.R. Pierce, and C.E. Shannon. The philosophy of pcm. *Proceedings of the IRE*, 36(11):1324 – 1331, Nov. 1948.
- [Orio *et al.*, 2011] N. Orio, D. Rizo, R. Miotto, N. Montecchio, M. Schedl, and O. Lartillot. Musiclef: A benchmark activity in multimodal music information retrieval. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Oudre *et al.*, 2011] L. Oudre, Y. Grenier, and C. Févotte. Chord recognition by fitting rescaled chroma vectors to chord templates. *IEEE Transactions on Audio, Speech and Language Processing*, 19(7):2222 – 2233, Sep. 2011.
- [Pearson, 1901] K. Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.
- [Piszczałski and Galler, 1977] Martin Piszczałski and Bernard A. Galler. Automatic music transcription. *Computer Music Journal*, 1(4):pp. 24–31, 1977.
- [Pohle *et al.*, 2005] T. Pohle, E. Pampalk, and G. Widmer. Evaluation of frequently used audio features for classification of music into perceptual categories. In *Proceedings of the International Workshop on Content-Based Multimedia Indexing*. Citeseer, 2005.
- [Quinlan, 1993] J.R. Quinlan. *C4.5: programs for machine learning*, volume 1. Morgan kaufmann, 1993.

- [Russell and Norvig, 2009] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition, 2009.
- [Schindler and Rauber, 2012] A. Schindler and A. Rauber. Capturing the temporal domain in echonest features for improved classification effectiveness. In *Proceedings of the 10th International Workshop on Adaptive Multimedia Retrieval (AMR 2012)*, 2012.
- [Schindler *et al.*, 2012] A. Schindler, R. Mayer, and A. Rauber. Facilitating comprehensive benchmarking experiments on the million song dataset. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012.
- [Serrà *et al.*, 2008] J. Serrà, E. Gómez, P. Herrera, and X. Serra. Chroma binary similarity and local alignment applied to cover song identification. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(6):1138–1151, 2008.
- [Serrà *et al.*, 2009] J. Serrà, M. Zanin, and R.G. Andrzejak. Cover song retrieval by cross recurrence quantification and unsupervised set detection. MIREX cover song identification contest, 2009.
- [Serrà *et al.*, 2010] J. Serrà, E. Gómez, and P. Herrera. Audio cover song identification and similarity: background, approaches, evaluation, and beyond. *Advances in Music Information Retrieval, Studies in Computational Intelligence*, 16:307–332, 2010.
- [Serrà *et al.*, 2012] J. Serrà, Á. Corral, M. Boguñá, M. Haro, and J.L. Arcos. Measuring the evolution of contemporary western popular music. *Scientific Reports*, 2, 2012.
- [Serrà, 2011] J. Serrà. *Identification of versions of the same musical composition by processing audio descriptions*. PhD thesis, Universitat Pompeu Fabra, Barcelona, 2011.
- [Shannon, 1948] C. E. Shannon. A mathematical theory of communication. *Bell system technical journal*, 27, 1948.
- [Sheh and Ellis, 2003] A. Sheh and D. Ellis. Chord segmentation and recognition using EM-trained hidden markov models. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, 2003.

- [Smith *et al.*, 2011] J.B.L. Smith, J.A. Burgoyne, I. Fujinaga, D. De Roure, and J.S. Downie. Design and creation of a large-scale database of structural annotations. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Stevens and Volkman, 1940] S.S. Stevens and J. Volkman. The relation of pitch to frequency: A revised scale. *The American Journal of Psychology*, 53(3):329–353, 1940.
- [Su *et al.*, 2010] J.H. Su, H.H. Yeh, P.S. Yu, and V.S. Tseng. Music recommendation using content and context information mining. *Intelligent Systems, IEEE*, 25(1):16–26, 2010.
- [Tingle *et al.*, 2010] D. Tingle, Y.E. Kim, and D. Turnbull. Exploring automatic music annotation with acoustically-objective tags. In *Proceedings of the international conference on Multimedia information retrieval*, pages 55–62. ACM, 2010.
- [Turnbull *et al.*, 2007] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *SIGIR '07: Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 439–446, New York, NY, USA, 2007. ACM.
- [Turnbull *et al.*, 2008] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *IEEE Transactions on Audio, Speech & Language Processing*, 16(2), 2008.
- [Tzanetakis and Cook, 2002] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Trans. on Speech and Audio Processing*, 10(5):293–302, 2002.
- [Viro, 2011] V. Viro. Peachnote: Music score search and analysis platform. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011.
- [Walters *et al.*, 2012] T.C. Walters, D.A. Ross, and R.F. Lyon. The intervalgram: An audio feature for large-scale melody recognition. In *Proc. of the 9th International Symposium on Computer Music Modeling and Retrieval (CMMR)*, June 2012.
- [Wang, 2003] A. Wang. An industrial strength audio search algorithm. In *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR 2003)*, 2003.

- [Weinberger and Saul, 2004] K.Q. Weinberger and L.K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 2, pages II-988. IEEE, 2004.
- [Weiss and Bello, 2010] R. Weiss and J. Bello. Identifying repeated patterns in music using sparse convolutive non-negative matrix factorization. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010.
- [Yu *et al.*, 2009] Y. Yu, M. Crucianu, V. Oria, and L. Chen. Local summarization and multi-level LSH for retrieving multi-variant audio tracks. In *Proceedings of the seventeen ACM international conference on Multimedia*, pages 341–350. ACM, 2009.

Part VI

Appendices

Appendix A

Content of the Million Song Dataset

In Table A.1 we list each field provided by each of the million HDF5 files contained in the Million Song Dataset.

Most are self-explanatory, but more details can be found online¹. It can be sufficient to note there are metadata fields (e.g. *artist name* or *title*), identifier fields (e.g. *track_id* or *artist_7digitalid*) and audio features (e.g. *segments_pitches* or *beats_start*).

¹<http://labrosa.ee.columbia.edu/millionsong/faq>

analysis_sample_rate	artist_7digitalid
artist_familiarity	artist_hotttnesss
artist_id	artist_latitude
artist_location	artist_longitude
artist_mbid	artist_mbtags
artist_mbtags_count	artist_name
artist_playmeid	artist_terms
artist_terms_freq	artist_terms_weight
audio_md5	bars_confidence
bars_start	beats_confidence
beats_start	danceability
duration	end_of_fade_in
energy	key
key_confidence	loudness
mode	mode_confidence
num_songs	release
release_7digitalid	sections_confidence
sections_start	segments_confidence
segments_loudness_max	segments_loudness_max_time
segments_loudness_start	segments_pitches
segments_start	segments_timbre
similar_artists	song_hotttnesss
song_id	start_of_fade_out
tatums_confidence	tatums_start
tempo	time_signature
time_signature_confidence	title
track_7digitalid	track_id
year	

Table A.1: List of the 55 fields provided in each per-song HDF5 file in the MSD.

Appendix B

Cover song experiment with audio data

Using audio as the input for cover song matching is a challenge, specifically because beat-synchronous features rely on a consistent beat tracker. Fortunately, The Echo Nest provides an open API which will convert any uploaded audio into the format provided in the Million Song Dataset.

We experimented with this using cover songs found on YouTube. Results below are obtained through the 2DFTM method. For instance, the song “Summertime” by *Ella Fitzgerald and Louis Armstrong*¹ was correctly associated with a cover version by *Joshua Redman* (first match). The *Ella Fitzgerald and Louis Armstrong* version present in the SHS was found at rank 9. The fact that this was not the first match might be explained by the lower audio quality on YouTube, or it could be a different version. Also, the features returned by The Echo Nest have slightly changed from the version in the MSD (version 4 versus 3). Considering the two interchangeable might be a mistake.

A few notes on extracting audio from Youtube: we use `youtube-dl.py`² to download the video, and `ffmpeg`³ to extract the audio to a wavfile. The command is:

```
ffmpeg -i <youtube file> -vn -ar 44100 -ac 2 -ab 192 -f wav output.wav
```

¹<http://www.youtube.com/watch?v=MID0EsQL71A>

²<http://rg3.github.com/youtube-dl/>

³<http://ffmpeg.org/>

If one does not want to rely on The Echo Nest API and has a large music collection to analyze, many feature extractors provide chroma features and beat estimation. We mention a few that are widely used. This is not an endorsement, nor is it an exhaustive list.

- Dan Ellis' chroma feature analysis and synthesis Matlab code (<http://labrosa.ee.columbia.edu/matlab/chroma-ansyn/>);
- Dan Ellis' beat tracking Matlab code (<http://labrosa.ee.columbia.edu/projects/beattrack/>);
- The MIR Toolbox contains both chroma computation and beat tracking code in Matlab (<https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox>);
- Vamp Plugins exist for both chroma computation and beat tracking code (<http://www.vamp-plugins.org/download.html>).

Appendix C

Publications by this author

- T. Bertin-Mahieux and D.P.W. Ellis. Large-scale cover song recognition using the 2d fourier transform magnitude. In *Proceedings of the 13th International Conference on Music Information Retrieval (ISMIR 2012)*, 2012
- B. McFee, T. Bertin-Mahieux, D. Ellis, and G. Lanckriet. The million song dataset challenge. In *Proc. of the 4th International Workshop on Advances in Music Information Research (AdMIRe '12)*, April 2012
- T. Bertin-Mahieux and D.P.W. Ellis. Large-scale cover song recognition using hashed chroma landmarks. In *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, New Platz, NY, 2011
- T. Bertin-Mahieux, D. Ellis, B. Whitman, and P. Lamere. The million song dataset. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2011)*, 2011
- T. Bertin-Mahieux, G. Grindlay, R. Weiss, and D. Ellis. Evaluating music sequence models through missing data. In *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2011
- T. Bertin-Mahieux, R. Weiss, and D. Ellis. Clustering beat-chroma patterns in a large music database. In *Proceedings of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, 2010

- T. Bertin-Mahieux, D. Eck, and M. Mandel. Automatic tagging of audio: The state-of-the-art. In Wenwu Wang, editor, *Machine Audition: Principles, Algorithms and Systems*, pages 334–352. IGI Publishing, 2010
- T. Bertin-Mahieux. Apprentissage statistique pour l’étiquetage de musique et la recommandation. Master’s thesis, Université de Montréal, 2009
- P.-A. Manzagol, T. Bertin-Mahieux, and D. Eck. On the use of sparse time relative auditory codes for music. In *9th International Conference on Music Information Retrieval (ISMIR 2008)*, 2008
- B. Kégl, T. Bertin-Mahieux, and D. Eck. Metropolis-hastings sampling in a filterboost music classifier. In *ICML Workshop on Music and Machine Learning*, 2008
- T. Bertin-Mahieux, D. Eck, F. Maillet, and P. Lamere. Autotagger: a model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research, special issue: “From genres to tags: Music Information Retrieval in the era of folksonomies.”*, 37(2), June 2008
- D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green. Automatic generation of social tags for music recommendation. In *Advances in Neural Information Processing Systems 20*. MIT Press, Cambridge, MA, 2008
- D. Eck, T. Bertin-Mahieux, and P. Lamere. Autotagging music using supervised machine learning. In *Proceedings of the 8th International Conference on Music Information Retrieval (ISMIR 2007)*, 2007