# COMBINING TIMBRIC AND RHYTHMIC FEATURES FOR SEMANTIC MUSIC TAGGING

RELATORE: Ch.mo Prof. Nicola Orio
Dipartimento dei Beni Culturali: Archeologia,
Storia dell'Arte, del Cinema e della Musica

LAUREANDO: Roberto Piva

A.A. 2011-2012


UNIVERSITÀ DEGLI STUDI DI PADOVA
DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE
TESI DI LAUREA

# COMBINING TIMBRIC AND RHYTHMIC FEATURES FOR SEMANTIC MUSIC TAGGING

RELATORE: Ch.mo Prof. Nicola Orio
Dipartimento dei Beni Culturali: Archeologia,
Storia dell'Arte, del Cinema e della Musica

LAUREANDO: *Roberto Piva*

Padova, 10 marzo 2012

*A Chiara.*
*Agli amici.*
*Agli anni più belli della mia vita.*

*Alla musica.*

**Abstract**

The recent explosion of online streaming services is a reflection of the high importance the multimedia has in our everyday life. These systems deal with large collections of media pieces and their usability is tightly related to the meta-data associated with content. As long as the meta-data is correctly assigned users can easily reach what they were looking for. The presence of poorly annotated data and continuous addition of new items can lead to poor recommendation performances and loss of earning because a large part of the collection will remain unexplored . Automatic tagging can be a step toward the solution for this problem, however state of the art system are not yet ready to substitute human annotators, expecially for music.

In this thesis we propose a novel approach to semantic music tagging. The project takes inspiration from Hidden Markov Models and exploits a statistical framework to semantically link two acoustic features. We make the assumption that acoustically similar songs have similar tags. We model our collection of known songs as a graph where the states represent the songs and the transition probabilities are related to the timbric similarity between songs. Observations and observation probabilities was modeled by rhythm descriptors and rhythm similarity. To query the model we simulate the insertion of the query song in the graph by calculating timbric and rhythmic similarity with the collection. A modified Viterbi algorithm was developed to extract semantically meaningful paths in this songs graph, from the query song. We infer the tags for the query by a weighted sum of tags from the songs in the extracted paths.

We tested our model using the CAL500 dataset, a well known dataset for music information retrieval evaluation, and results are promising. Performance measures are good for an first implementation of a new model.

This thesis is intended to show the results of our work between august, 2012 and march, 2013 on this project.

# Contents

# Introduction

In the cloud computing era lots of people manage their files and media on online services, often relying on the availability guaranteed by this kind of systems. Content has to be somehow indexed, as users want their data to be also easier and faster to seek. Services providers usually offer multiple ways to handle this. Data could be organized in a tree-like fashion, as desktop computer's OS got us used to, in a data type specific scheme (e.g. movies may be organized by director, year, cast, genre and so on, whereas books may be organized by author and year), or in a tag system, like music streaming services usually do.

Tags are (often) short texts, usually of a single word, that suggest and describe the kind of content the data represent, they are the are so-called meta-data. Putting them along with the data is usually a task for a human being, not only for the difficulty to gather synthetic description of the data, but also because tagging involves some subjective judgements and logical connections between concepts. Another important thing to note is that metadata has a very big role in every media that is not based in text or graphic symbols (i.e. books, pictures) and/or it is not static: we can build a program that summarize a text but you cannot write a summary of a movie without actually seeing it before, and the same applies to music.

Speaking about music, tagging has always been a feature of online streaming services like LastFM[1], Apple Genius, Pandora[2] or Grooveshark[3]. Their tags have different origins though: while Pandora pays music experts to put correct and well expressive tags on music[4], LastFm and Apple Genius rely on user generated tags and playlists and use statistics tools like collaborative filtering (see [15]) for annotating new items. For these services having good tags for their music is essential, since they rely on this to deliver the right songs to the right user, selecting the ones that suit the user's tastes and the mood of previously listened songs.

[1] http://www.last.fm
[2] http://www.pandora.com
[3] http://www.grooveshark.com
[4] As part of the Music Genome Project http://www.pandora.com/about/mgp

They also have to deal with newly added material and a whole range of music, spacing from new broadly popular hits to very little known productions from independent labels or individual bands. For the popular music getting proper tagging may not be an issue, at least if there are advertiser which push songs in the market. But what about the rest of the tracks users ignore? Will they ever be properly described? Will that obscure but very interesting track be considered if it is poorly tagged? The short answer is "no", we cannot reach a song without proper tagging . Without an automatic tagging system popular songs will become more popular and newly written songs (and not-so-popular ones) will not be annotated and thus progressively ignored.

There is also a niche (actually not so little) in which proper tagging is requested but little or not at all provided: there are large production libraries that contains millions of songs. These songs are ready to be used into a variety of situations, such as commercial jingles, background music of television shows, or even elevator music (muzak), but they are ignored because of poor tagging. If library holders are asked to select a song for a particular purpose they may have to listen randomly to hundreds of tunes to get a proper one, or, worse than that, they may rely only on the already tagged songs, leaving the others in the oblivion.

Another big problem is the long tail distribution: if we look on the listening charts of a music streaming user we can see that the distribution of play counts over artists resemble a power law. This means that there is a restricted number of artists which get a high play count, while there is a long tail of artists with a little play count each (at least compared to the top artists). We could think that those artists which are in the long tail are being rather unnoticed, or even not appreciated. The problem here is that if we sum up (integrate) the play counts of the long tail artists we outnumber the total play counts of the first ones. This situation is replicated for every user (even those who listen to "alternative music", or to "every kind of music") and, more generally, to the global charts of a music streaming service. The way the listening distribution has a reflection on services which make money on music, and there has been some investigation on how this long tail affects the market. Quoting a work by Anita Elberse (see [12]) "The goods in the long tail include former hits as well as true niche content. Most of the latter was never released through traditional distribution channels or consists of orphans of unbundling activity (as with individual tracks in the music industry)". The problem is related to tagging because in most of the times the long tail consists of poorly tagged music pieces. It goes without saying that a song with no metadata cannot be found, and this is true expecially for music we do not know about (i.e. new pieces).

Into this scenario lots of work has been focused into automatic tagging of music, in order to speed up the evaluation process for searching music and get a proper

model to label new one. Early works were toward the recognition of the music genre (see [20, 21, 46]), often pointing out that "genre" classification is very subjective and different values have high tendency of overlapping. This problem led many researchers to propose a new taxonomy for genre classification (see example in [31]). However, music tagging is a more broad concept that includes other type of information, such as whether the track is slow paced or it is fast, or whether you can dance on it or not, that's why some publications call it "music semantic annotation" (e.g. [17, 44]). This subject is challenging, and lots of researchers tried different approaches to improve music annotation performance. Notable works include the association between related tags (i.e. co-occurrence) while judging some track in [26], or the inclusion of mined social tags to infer some relation between tags and audio features in [2].

A variety of features were considered in search of the right combination to correctly annotate novel songs. First attempts started with Mel Frequency Cepstral Coefficients (see section 1.3), which were proven valid in speech recognition (see [34]) and were promptly tried for music classification (see the work by Beth Logan in [23]). MFCCs turned out to be quite good but in the end researchers hit what it was then called a "glass ceiling" (see [30] for an extensive work on this), and they saw that exploiting the sole timbral features was not enough. Recently better results were shown in the MIREX 2008 evaluation campaign (see [9]) with the aid of other sources, such as social tags in [11], or temporal features like beat, tempo and rhythm complexity like in [24]. More recent results (MIREX 2012) were pointed toward the improvement of MFCC features by using different way to compute them: using Principal Component Analysis like in [22] or using different time scales like [16]. There are also works on using visual representation of music features for genre and mood classification: Wu *et al.* did it with timbre and beat features using Gabor filter banks in [49]. Some complementary work has been made on making frameworks for audio music similarity and retrieval (see [7, 47]) where researchers seek for new features, different audio representations and better computing performances. Recent works usually make heavy use of the Support Vector Machines for different purposes: for example, Panda *et al.* [33] and Bourguigne *et al.* [3] use them for classification, while Lim *et al.* [22] uses them for feature selection. Other learning frameworks are used as well, for example there are works that makes use of Artificial Neural Networks for music classification like [16] and works that use variants of them like Restricted Boltzmann Machines in [7].

In this thesis we will discuss a method for semantically tagging music clips, by exploiting rhytmical features (i.e. rhythm histograms as described by Rauber and Lidy here [21]) on top of a previously defined retrieval model by Miotto and Orio [28]. The model itself uses a modified Hidden Markovian Model (see [35]) to combine tags

and acoustic similarity and thus providing a semantic link between them: the results were promising on the retrieval task, and we will show a way use the same system for tagging new clips.

The thesis is organized as follows. On the first part (Chapter 1) we will describe shortly the mathematical symbols , structures and methods. Then we will present the model described by Miotto and Orio (Chapter 2), combined with our features, followed by how we tag new clips and methods we would develop or explore but we did not. In the end of the document you will find results (Chapter 3) from our tests, our conclusions and suggestions for future works on this topic.

# Chapter 1

# Background

In this section we will describe the mathematical symbols, structures and methods involved in this thesis, to better understand the concepts and as a recap for letting everybody share the same knowledge before getting to the core content.

## 1.1 Hidden Markovian Models

Hidden Markovian Models are mathematical structures and methods which let us infer the statistical properties of an unknown discrete Markov processes. Using some particular mathematical method we can infer the structure and the transition probabilities by only knowing (a sample of) an observation sequence. First introduced by Leonard E. Baum in 1966 [1], HMMs became popular because of their practical and compact representation of unknown sequences. HMMs permit some form of prediction of the next emission under certain circumstances, making them a form of machine learning framework. To better comprehend the HMM structure we need to introduce the discrete Markov processes, which are the basic frame of HMMs.

### 1.1.1 Markov chains

A discrete Markov process is a stochastic process which has a finite (or countable) number of states and probabilistic transitions between them. The main property of these systems is the Markov property, that is, the conditional probability of switching between states depends only on the current state, not on the ones from previous steps. In other words, Markov property makes the systems memoryless.

More formally: think of a system which may be described by being, at any time, in one state from a finite set of $N$ distinct states[1] named $S_1, \ldots, S_N$. At regularly spaced

[1] We take it finite because we do not need further complication but the theory let the set be up to

times the system undergoes a state change (possibly to the same state) according to a set of probabilities associated with the current state. If we number the time instants as $t = 1, 2, \ldots$, and the current state as $q_t$, we may specify the transition probability is described as follow:

$$P(q_t = S_j | q_{t-1} = S_i, q_{t-2} = S_k, \ldots, q_0 = S_l) \tag{1.1}$$

that is, we would have to specify every single previous conditional transition probability. Because of the Markov property the whole description is truncated to just the current and previous state, and the state transition probabilities are indipendent of time:

$$P(q_t = S_j | q_{t-1} = S_i) = a_{ij} \qquad 1 \leq i, j \leq N. \tag{1.2}$$

The transition coefficients $a_{ij}$ satisfy the following standard constraints:

$$a_{ij} \geq 0 \qquad 1 \leq i, j \leq N \tag{1.3}$$

$$\sum_{j=1}^{N} a_{ij} = 1. \tag{1.4}$$

We also have to define some initial probability, named $\pi_i$, for each state: of course these prior probabilities must satisfy some basic constraints:

$$\pi_i \geq 0 \qquad 1 \leq i \leq N \tag{1.5}$$

$$\sum_{i=1}^{N} \pi_i = 1. \tag{1.6}$$

The above system is called an observable Markov model, since at each time we can inspect the system state as it is the output of the process.

To fix ideas, think of the following game: you have 2 boxes and some colored balls, in box 1 there are 3 blue and 7 red balls, in box 2 there are 1 blue and 9 red balls. A genie does the following steps: first it chooses a box, with 0.5 probability each, and then it picks a random ball from that box, with uniform probabilty. Finally, it puts the ball in the same box as before. At the end of the process he choses a random box to pick balls from, however, it has some bias: if the genie had picked a blue ball it is more likely to choose box 1 the next time, while picking a red ball lead to equal box probability. We may model (we may call this model $\lambda_1$) this situation with a 4 state Markov chain:

State 1: The genie has chosen box 1
State 2: The genie has chosen box 2
State 3: The genie has picked a blue ball
State 4: The genie has picked a red ball.

countable infinite

The process starts at a random box state, and we can write the transition probability matrix $A$ as follows:

$$\boldsymbol{A} = \{a_{ij}\} = \begin{vmatrix} 0 & 0 & \frac{3}{10} & \frac{7}{10} \\ 0 & 0 & \frac{1}{10} & \frac{9}{10} \\ \frac{3}{5} & \frac{2}{5} & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \end{vmatrix} \tag{1.7}$$

We can see that getting to state 3 from state one has $\frac{3}{10}$ probability, while going to state 1 from state 2 is impossible ($P(S_t = 1|S_{t-1} = 2) = 0$). Also, the chances to pick a blue ball are higher if the genie had chosen the first box (i.e. if the machine was in state 1 before coming to state 3). To complete the model we must also specify the prior probability vector $\pi$:

$$\boldsymbol{\pi} = \{\pi\} = [0.5, 0.5, 0, 0] \tag{1.8}$$

The model $\lambda_1 = (A, \boldsymbol{\pi})$ itself describe well the whole process, however we are interested only in the ball color, so we can simplify the model by retaining only 2 states:

State B: The genie has picked a blue ball
State R: The genie has picked a red ball

And summarize the transition probabilities by summing each possible path that can lead from B to B, B to R and so on:

$$\begin{aligned} P(q_t = B|q_{t-1} = B, \lambda_2) &= P(q_t = S_3|q_{t-2} = S_3, \lambda_1) = \frac{3}{5}\frac{3}{10} + \frac{2}{5}\frac{1}{10} = \frac{11}{50} \\ P(q_t = B|q_{t-1} = R, \lambda_2) &= P(q_t = S_3|q_{t-2} = S_4, \lambda_1) = \frac{1}{2}\frac{3}{10} + \frac{1}{2}\frac{1}{10} = \frac{4}{20} \\ P(q_t = R|q_{t-1} = R, \lambda_2) &= P(q_t = S_4|q_{t-2} = S_4, \lambda_1) = \frac{1}{2}\frac{7}{10} + \frac{1}{2}\frac{9}{10} = \frac{16}{10} \\ P(q_t = R|q_{t-1} = B, \lambda_2) &= P(q_t = S_4|q_{t-2} = S_3, \lambda_1) = \frac{3}{5}\frac{7}{10} + \frac{2}{5}\frac{9}{10} = \frac{39}{50} \end{aligned} \tag{1.9}$$

We need also to adjust the prior probabilities:

$$\begin{aligned} \pi_1^{\lambda_2} &= P(q_2 = S_3|q_1 = S_1, \lambda_1) \cdot \pi_1^{\lambda_1} + P(q_2 = S_3|q_1 = S_2, \lambda_1) \cdot \pi_2^{\lambda_1} = \frac{1}{2} \\ \pi_2^{\lambda_2} &= P(q_2 = S_4|q_1 = S_1, \lambda_1) \cdot \pi_1^{\lambda_1} + P(q_2 = S_4|q_1 = S_2, \lambda_1) \cdot \pi_2^{\lambda_1} = \frac{1}{2} \end{aligned} \tag{1.10}$$

where with the superscripts $\lambda_1$ and $\lambda_2$ we denoted the prior probabilites of first and second model. At this point we have a simplified model of this process, and we can answer simple questions like:

1. What is the probability for we to see the sequence "blue-blue-red-blue-red" or $O = \{S_1, S_1, S_2, S_1, S_2\}$?

2. Given the system is in state $S_i$, what is the probability that system stays on it for $d$ time slices?

The answer to question 1 is as simple as multiplying the conditional probabilities of the transitions plus the prior probability:

$$\begin{aligned} P(O|\lambda_2) &= \pi_1 \cdot P(S_1|S_1) \cdot P(S_2|S_1) \cdot P(S_1|S_2) \cdot P(S_2|S_1) \\ &= \frac{1}{2} \cdot \frac{11}{50} \cdot \frac{39}{50} \cdot \frac{1}{5} \cdot \frac{39}{50} \\ &= 1.338 \times 10^{-2} \end{aligned} \tag{1.11}$$

The answer to question 2 is not harder to tell, we only need to specify the meaning of it in terms of observation sequence:

$$O = \{ \underbrace{S_i, S_i, \ldots, S_i}_{\text{d observations}} , S_j \neq S_i \}$$

and calculate the probability as above:

$$P(O|q_1 = S_i, \lambda_2) = (a_{ii})^{d-1}(1 - a_{ii}) = p_i(d) \tag{1.12}$$

where $p_i(d)$ is the discrete probability density function of duration $d$ for state $i$. The fact that $p_i(d)$ is exponential in terms of $d$ is a characteristic of Markov chain's state duration.

The above mentioned structure is what is called an observable stochastic process, that is, we can see exactly "what is going on", we know the states the system can be on, and we know all the transition probabilities. Unfortunately this model is too restrictive for some particular problems (like our, as we will see). We may need the ability to guess or infer the system structure by only looking at the observations, like the system is a black box and we can see only it's output. In the next subsection we will have a look into Hidden Markovian Models, which are an extension of this mathematical structure.

### 1.1.2 Hidden Markovian Models

Once we have seen the potentialities of a Markov chain, we may extend this definition by introducing emissions as probability functions of the states. The result is called Hidden Markov Model (HMM), which is a doubly embedded stochastic process: one process is hidden and it is observable via a set of stochastic processes that produce the observation sequences.

To fix ideas consider the previous urn and balls model: the genie is behind a wall, hiding from you, and performs the experiment by only letting you know the result

of the ball picking (i.e. "I've extracted a blue ball"). Giving the ball color you can not really tell from which urn the ball came from: the only thing we would see is a sequence of "blue" and "red". We want to build an HMM to model the event.

The first issue is to decide what a state represents, and how many states our model should have: in this case we may choose to represent one state per ball color, thus having two states, or maybe two states, each representing an urn, with different ball color emission probabilities per state. If we do not know what's behind the wall (in a real situation this would not be an unlikely event) we may also guess there are more urns, and the model could include 3 states or more.

The first situation is, in fact, an observable Markov process, and we only need to specify the transition probabilities. If we want a state to represent an urn we are in a different situation: each state has a different stochastic process which picks a ball color with a probability distribution, and the state transitions is itself a stochastic process with its own probabilities. As we see the complexity of the model grows more than linearly with the number of states (we need 1 parameter for the observable process, 2 emission probabilities + 2 transition probabilities for the 2 states model) while we gain more degrees of freedom for modeling complex events.

Choosing the right representation and number of states is a crucial point for an HMM: if we take too few states we may be not able to get a full representation of the event, not covering some situation that a correctly designed HMM would. If we set up too many states we may be using an underspecified model (e.g. we may model a 1 coin toss with a 3 states HMM). Moreover, we will see some computational complexity implications that strongly limit the size of the models we may consider. Before we see how to use these models, we must introduce an example and some formalism.

Following the very good Rabiner's tutorial (as before, see [35]) we introduce the *Urn and Ball model*, a generalization of the examples seen before: We have $N$ (finite) urns, and each urn has a lot of colored balls in it. Balls can be of one of $M$ (finite) distinct colors. Like before a genie initially chooses an urn and extract a ball from it, the ball's color is recored as an *observation* and replaced in the original urn. Then the genie chooses another urn according to some stochastic process associated with the current urn, and repeats the ball selection process (and so on). What we want to do is to model the entire event using the finite observation sequence as the observable output of an HMM. As you can guess the simplest way to model this is the same as before: each urn corresponds to a single state, and the extracted ball's color is a probabilistic function of the current state. State transition is ruled by the transition matrix.

More formally, an HMM is characterize by 5 parts:

1. A set $S$ of $N$ states, where $S = \{S_1, S_2, \ldots, S_N\}$. For many application there is a

physical correspondence between the event and the modeled states, such as for the urns before. The state at time $t$ is referred as $q_t$.

2. A set $V$ of $M$ different observation simbols. For the urn and balls model these symbols are the balls' colors. We denote the symbols as $V_i$ with $1 \leq i \leq M$.

3. The state transition probability matrix $\mathbf{A} = \{a_{ij}\}$ where

$$a_{ij} = P(q_{t+1} = S_j | q_t = S_i), \qquad 1 \leq i, j \leq N \tag{1.13}$$

4. The observation symbols probability distribution for state $j$, $\mathbf{B} = \{b_j(k)\}$, where

$$b_j(k) = P(V_k \text{ at } t | q_t = S_j) \qquad 1 \leq j \leq N,\ 1 \leq k \leq M \tag{1.14}$$

5. The initial state distribution $\pi = \{\pi_i\}$, where:

$$\pi_i = P(q_1 = S_i), \qquad 1 \leq i \leq N \tag{1.15}$$

We can now say that to completely describe an HMM we need two parameters $(N, M)$, a set of observation symbols $V$, and the specification of three probability measures $\mathbf{A}, \mathbf{B}$ and $\pi$. We now refer to the model with the following compact notation:

$$\lambda = (A, B, \pi) \tag{1.16}$$

### 1.1.3 Three basic problems

Now we can define the three basic problems associated with HMMs:

1. Given the observation sequence $\mathbf{O} = O_1, O_2, \ldots, O_T$ and model $\lambda = (A, B, \pi)$ how do we efficiently compute the observation probability $P(O|\lambda)$ ?

2. Given the observation sequence $\mathbf{O} = O_1, O_2, \ldots, O_T$ and model $\lambda = (A, B, \pi)$ how do we choose a state sequence which most likely produce the aforementioned sequence (i.e. it is optimal in some sense)?

3. How do we choose the model parameters $\lambda = (A, B, \pi)$ to maximize a (serie of) $P(O|\lambda)$?

In this thesis we are interested in the solution for the second problem, but in this section we will have a look at all three.

The solution to problem 1 gives us an idea on how a model is adherent with some specification based on observation. Moreover, if we have more models, we can compare them by the probability of emitting a particular sequence. A *naïve* but correct

solution to this problem is to enumerate each possible path sequence of lenght $T$ and sum the emission probability for the given observation sequence for each path. That is, for the aforementioned observation sequence $O$ we can write, for each state sequence $\mathbf{Q} = q_1, \ldots, q_T$, the transition probability $P(Q|\lambda)$ and the emission probability $P(O|Q, \lambda)$ as:

$$P(Q|\lambda) = \pi_{q_1} a_{q_1 q_2} a_{q_2 q_3} \cdots a_{q_{T-1} q_T} \tag{1.17}$$

$$P(O|Q, \lambda) = b_{q_1}(O_1) b_{q_2}(O_2) \cdots b_{q_T}(O_T). \tag{1.18}$$

To compute the solution to problem 1 the joint probability of O and Q (i.e. their product) must be summed for all possible state sequences Q giving:

$$P(O|\lambda) = \sum_{\text{all } Q} P(O|Q, \lambda) P(Q|\lambda) \tag{1.19}$$

$$= \sum_{Q^i \text{ from all } Q} \left[ \sum_{q_1^i, q_2^i, \ldots, q_T^i} \pi_{q_1} b_{q_1}(O_1) a_{q_1 q_2} b_{q_2}(O_2) \cdots a_{q_{T-1} q_T} b_{q_T}(O_T) \right]. \tag{1.20}$$

We could easily see that this way the computational cost is exponential with the value of $T$.

A wiser way to calculate the sequence probability is to use the *forward-backward algorithm*. This procedure is split in two parts: the forward computation, which solves problem 1, and the backward computation, which will be presented as it will be useful for problem3.

In the forward computation we calculate the *forward variable* $\alpha_i(t)$ as here:

$$\alpha_t(i) = P(O_1\, O_2 \cdots O_t, q_t = S_i|\lambda) \tag{1.21}$$

This variable express the probability of being in state $i$ after seeing the partial observation from $O_1$ to $O_t$. We compute $\alpha_i(t)$ inductively:

$$\alpha_1(i) = \pi_i b_i(O_1), \qquad 1 \le i \le N \tag{1.22}$$

$$\alpha_{t+1}(j) = \left[ \sum_{i=1}^{N} \alpha_t(i) a_{ij} \right] b_j(O_{t+1}) \qquad 1 \le t \le T-1,\ 1 \le j \le N. \tag{1.23}$$

And finally we can calculate the observation probability by summing the last $\alpha$:

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i). \tag{1.24}$$

What we have done is to set the initial $\alpha_1(i)$ to the joint probability of state $S_i$ and observation $O_1$, then we can calculate $\alpha_{t>1}(j)$ for state $S_j$. Because $S_j$ can be reached from

(possibly) every other state we must calculate the joint probability of getting to state $S_j$ from previous states (and observations) and observing the current observation in here: this is what induction calculate in equation 1.23. Once we have inductively calculated every $\alpha_T(i)$ from $i = 1$ to $N$, equation 1.24 sums them up to get the final probability of the observation for model $\lambda$. This algorithm complete the task by performing $O(N^2T)$ calculations, which is better than the *naïve* version. The forward computation already solves problem 1, but as we stated before, the *backward computation* will be interesting for problem 3, so on the next paragraph we will present it.

The backward computation defines a backward variable $\beta_t(i)$ which is similar to the $\alpha$ from the forward computation. The difference is that it considers the observations starting from $O_{t+1}$ to $O_T$:

$$\beta_t(i) = P(O_{t+1}\, O_{t+2} \cdots O_T, q_t = S_i|\lambda) \tag{1.25}$$

i.e. it is the probability of observing the last part of the observation sequence giving we are on state $S_i$ at time $t$. As before we can compute $\beta$ in an inductive way:

$$\beta_T(i) = 1, \qquad 1 \leq i \leq N \tag{1.26}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_{ij} b_j(O_{t+1}) \beta_{t+1}(j), \qquad 1 \leq i \leq N,\ t = T-1, T-2, \ldots, 1. \tag{1.27}$$

The first step initialises all $\beta_T(i)$ to 1, arbitrarily. Then the induction shows that in order to be in state $S_i$ at time $t$ and subsequently see observations from $O_{t+1}$ and on, you have to consider all states at time $t + 1$, with the respective transition and observation probabilities. The backward computation takes again $O(N^2T)$ calculations, and we will see how it will help solving problems 2 and 3.

Problem 2 is about finding a state sequence, for a given observation sequence, which is optimal in some sense. We say "optimal" because we cannot find a single right state sequence, matter of a fact the state transition process is hidden. One possible way to define an optimal state sequence is to determine the states which are individually optimal for the given observation. That way we can exploit the forward-backward procedure, and select the states according to the variable:

$$\gamma_t(i) = P(q_t = S_i|O, \lambda). \tag{1.28}$$

which can be written in terms of the forward and backward variables. This variable express the probability of being in state $S_i$ at time $t$ given the observation sequence $O$ and the model $\lambda$. The problem with this definition is that it doesn't take in account the resulting state sequence probabilities, thus leading to sequences which may not be even feasible (e.g. they may have 0 valued transitions between them).

A solution to this is to consider a single best state sequence by maximizing both the probabilities of state and observation sequences. The algorithm which does this is called Viterbi algorithm and it is based on dynamic programming methods. The algorithm keeps trace, for each possible state and for each observation, of the best partial path via the following variable:

$$\delta_t(i) = \max_{q_1,q_2,\ldots,q_{t-1}} P(q_1 q_2 \cdots q_t = i, O_1 O_2 \ldots | \lambda). \tag{1.29}$$

The $\delta$ variable holds the best partial score for a single path at time $t$ (the first $t$ observations) that ends at state $S_i$. We need also to keep track of the chosen path, so we introduce a new variable $\psi_t(j)$ that holds the state values for each observation $t$ and state $S_i$. By induction we can define $\delta$ and $\psi$ as:

$$\delta_1(i) = \pi_i b_i(O_1), \qquad 1 \le i \le N \tag{1.30}$$

$$\psi_1(i) = 0, \qquad 1 \le i \le N \tag{1.31}$$

$$\delta_t(j) = \max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}]b_j(O_t) \qquad 1 \le i \le N,\ 2 \le i \le T \tag{1.32}$$

$$\psi_t(j) = \arg\max_{1\le i\le N}[\delta_{t-1}(i)a_{ij}] \qquad 1 \le i \le N,\ 2 \le i \le T \tag{1.33}$$

As we can see the initialization of $\delta_1(i)$ (equation 1.30) keeps track of the prior probabilities and the first observation probabilities; the $\psi_1(i)$ are set to 0, which has no meaning, but it doesn't affect the result. The algorithm is quite similar to the forward computation as seen before, the only difference is that equation 1.32 makes use of the maximization in place of the summing, matter of a fact we are looking for a single best path. The other significant difference is that we take in account the arg max of the maximizations, because we need to trace back the single best path. Next equations explain how we can get the best path's probability and backtrack the path itself via $\psi$ variables:

$$P^* = \max_{1\le i\le N}[\delta_T(i)] \tag{1.34}$$

$$q_T^* = \arg\max_{1\le i\le N}[\delta_T(i)] \tag{1.35}$$

$$q_t^* = \psi_{t+1}(q_{t-1}^*), \qquad t = T-1, T-2, \cdots, 1. \tag{1.36}$$

As we see, for $P^*$ the calculation is quite easy, since we are already carrying the conditional probability (a consequence of the Markov property, equation 1.2). For $q^*$ we need to iterate over the $\psi$ values to infer the resulting path (equation 1.36). Computational comlpexity is similar to the forward computation from problem 1.

Problem 3 is about inferring or modifying the inner probabilities $(A, B, \pi)$ of the model by only observing an (a set of) observation sequence. The goal is to maximize

the observation sequence's probability. This is the most challenging problem for the HMM structure, and there are no known way to analitically solve this task.

The Baum-Welch method is an iterative procedure that maximizes $P(O, \lambda)$ locally, i.e. it obtains a local maxima for the observation sequence, not necessarily a global one. This method makes use of a particular forward-backward procedure that keeps track of the single transition. The probability that the best path for $O$ touches states $S_i$ and $S_j$ sequentially is called $\xi_t(i, j)$ and the probability of being in state $S_i$ at time $t$ is $\gamma_t(i)$, as seen equation 1.28. Baum-welch method defines a new model $\overline{\lambda}$ in terms of $\xi$ and $\gamma$ so that:

$$P(O|\overline{\lambda}) > P(O|\lambda) \tag{1.37}$$

that is, in the new model the observation sequence is more likely to be produced. We won't cover the whole procedure here, since we do not need it for this thesis. However you can find a very well written explanation of Baum-Welch algorithm in Rabiner's paper [35].

## 1.2 Rhythm Histograms

According to Oxford Dictionary[2], rhythm is "a strong, regular repeated pattern of movement or sound". If we think of rhythm as a superimposition of amplitude modulation frequencies, humans perceive as rhythm only modulation frequencies between about 0 and 10 Hz. Rhythm Histograms (RH) are an audio feature which holds information regarding the rhythm of a given piece of audio. RH where introduces in 2001 by Pampalk [32] and Rauber [36], that were in search of *psychoacoustically motivated Rhythm Patterns*. Rhythm Patterns themselves are a variation of Fluctuation Pattern, an audio feature developed by Fastl [14] in 1982 for a purpose similar to RHs. To better understand Rhythm Histograms we will first introduce Rhythm Patterns.

### 1.2.1 Rhythm Patterns

Rhythm Patterns are an audio feature which represents rhythm in a piece of audio. Following Rauber's paper [21] we can first define how they are calculated, and then we can comment on what they represent. The first step is to preprocess the audio, forcing it to be in mono format at 44 kHz sampling resolution. Then we iterate the following steps for each 6 seconds excerpts of audio taken from the original audio piece (with lead in and fade out options):

[2] http://oxforddictionaries.com/definition/english/rhythm

**S1** transform the audio segment into frequency representation via Fast Fourier Transform with Hanning window of 23 ms and 50% overlap between windows: this will obtain roughly 520 samples, which is very close to a power of 2, allowing the FFT calculation to be efficient[3].

**S2** apply Bark scale (see [50]) by grouping frequency bands in 24 critical bands: this accounts for the human perception of frequency, which is not linear (i.e., we cannot perceive well the same frequency differences at different frequency ranges).

**S3** apply spreading function to account for spectral masking effect (see [39]): again this is done because of the human perception mechanism.

**S4** transform the resulting energy values in deciBel scale [dB]: this is a standard step that is motivated by the better representation of energy ratio, which is a thing humans perceive better than absolute energy value.

**S5** transform the energy values in loudness levels, making use of equal-loudness contours [Phon]: this is done to account for the human way of loudness perception (i.e. the psychological perception of amplitude) as a function of amplitude level.

**S6** compute specific loudness sensation per critical band [Sone]: this transformation account for loudness perception as a function of the frequency.

**R1** apply Fast Fourier Transform to the Sone representation (treat every critical band as a signal, calculate FFT for each band): this will produce a time-invariant representation of the critical bands. This procedure highlights the rhythmic structure of the critical bands in the form of amplitude modulation up to 43 Hz[4]. However, as said before, humans do not recognize as rhythm the frequencies above 10 Hz, so we take the first 60 of 520 modulation frequencies.

**R2** weight modulation amplitudes according to human fluctuation strength sensation: humans perceive well amplitude modulation at around 4 Hz, and decreasing toward 15 Hz.

**R3** apply a gradient filter and perform Gaussian smoothing: this will emphasize distinctive beats and reduce noise from un-noticeable variations.

After these steps we will have a $24 \times 60$ matrix that holds information on rhythm patterns of the 6 seconds excerpt. To calculate Rhythm Patterns of the whole piece of

[3] Remember the Cooley-Tukey $O(N \log(N))$ optimization, expecially the radix-2 decimation in time

[4] 520 samples for 6 seconds audio means we have 86 Hz sampling frequency, then Nyquist-Shannon theorem comes in action and halves the value to get the right frequency resolution

audio we have to calculate the median of Rhythm Patterns for every 6 second of music. In the original paper Rauber states that steps S3, S4, S5, S6, R2 and R3 are considered optional, although they are useful and make the representation more appropriate.

### 1.2.2 Rhythm Histograms

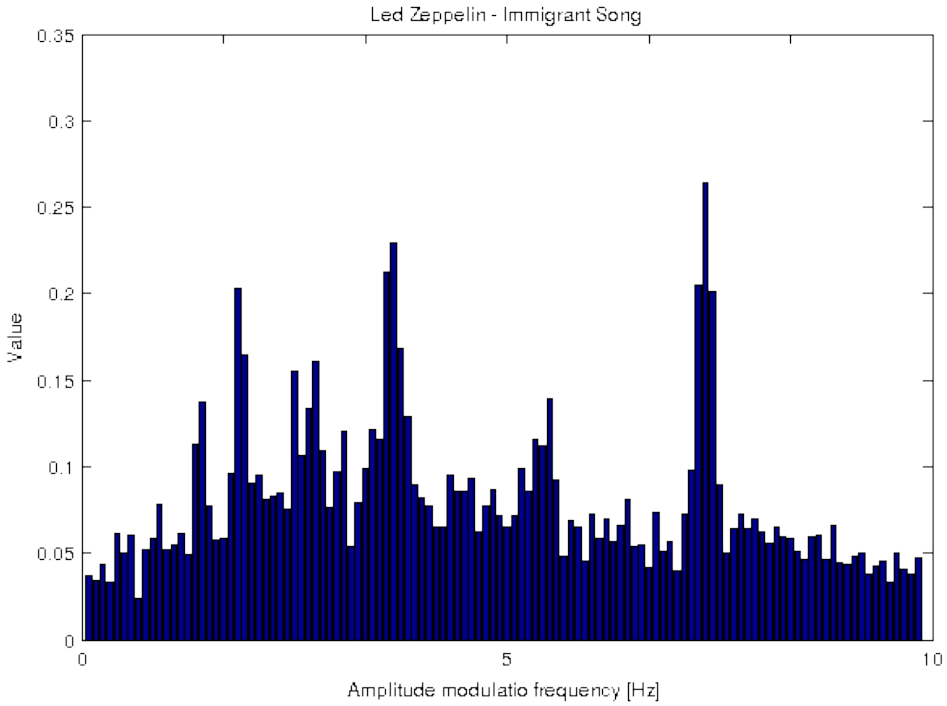
Rhythm Histograms are an audio feature that sums up the rhythmics of an audio document. Rhythm Histograms store information regarding rhythm without referring to a critical band, thus keeping only a single value for each amplitude modulation frequency bin. To calculate the Rhythm Histogram of a given piece of audio we need to compute the Rhythm Pattern first and then sum the values for each frequency bin. The result is a 60 elements vector representing the "rhythmic energy" of the corresponding modulation frequencies. The calculation of the Rhyhtm Histogram of a complete piece of audio takes the same steps as Rhythm Patterns: we consider the Rhythm Histograms of each 6 second excerpt and we take the median of all Histograms as the final result. In figure 1.1 we can see Rhythm Histograms from some selected songs.

## 1.3 Mel Frequency Cepstral Coefficients

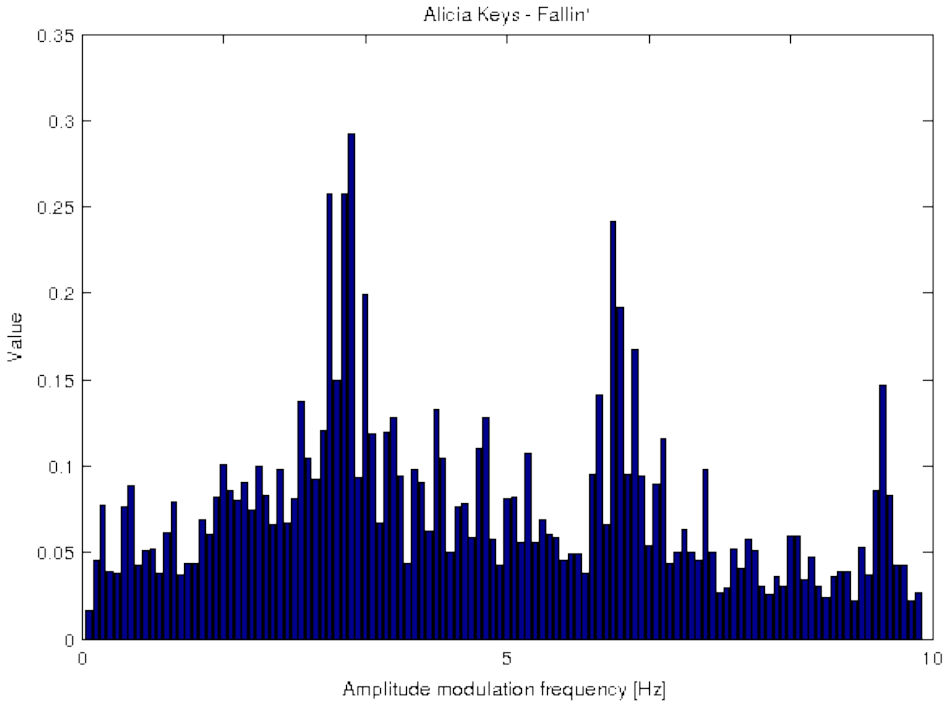
Mel Frequency Cepstral Coefficients, or MFCCs are timbric features. The timbre can be defined as the "sound characteristics that allow listeners to perceive as different two sounds with the same pitch (i.e., the perceived fundamental frequency of a sound) and intensity (i.e., the energy of the music vibration) but played by different instruments" (taken from [29]). They are short-term spectra-based features that are widely used in speech recognition because of their power to highlight the timbre of an audio piece. The "Mel" part of the name comes from the Mel scale, which is the psychoacoustic base for MFCCs. Beth Logan [23] made an interesting work on MFCCs, showing that they can be effectively used in music modeling. Figure 1.3 shows two examples of MFCCs plots: we can clearly see that the timbric signature makes the two songs distinguishable.

### 1.3.1 Mel scale

Mel scale is a perceptual pitch scale originally developed by Stevens *et al.* [41]. The original task was to determine how human perceive pitch and pitch ratio. Based on multiple tests, Stevens and his group discovered that from 1000Hz upwards humans tend to perceive less variation in pitch. To explain how big is the difference, think




(a) Rhythm Histogram plot for an excerpt of Led Zeppelin's "Immigrant Song"




(b) Rhythm Histogram plot for an excerpt of Alicia Keys' "Fallin' "

Figure 1.1: Some examples of Rhythm Histograms from selected songs (taken from a single 6 second excerpt)

that a 4 real octave interval can be perceived as a 2 octaves interval. The full inspection of frequencies in $[0, 10000]$ Hz shows a log-like relation between perceived and real frequency. Because of the subjectiveness of the data, there is no single formula for converting frequency to mel scale. Figure 1.2 shows a plot of a commonly used

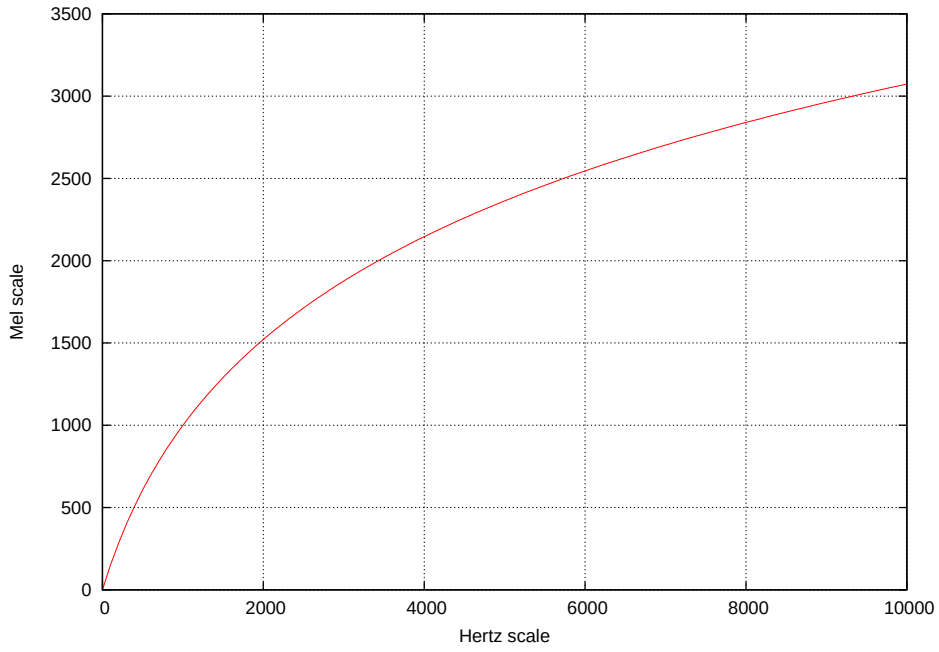



Figure 1.2: The mel-hertz equivalence, calculated with formula 1.38

formula, that is:

$$m = 1127 \log_e \left(1 + \frac{f}{700}\right) \tag{1.38}$$

where $f$ is the frequency in Hertz, and $m$ is the frequency in mel scale.

### 1.3.2 MFCC calculation

MFCCs are easy to compute, and the procedure consists in 5 steps:

**S1** Divide the source signal in little frames, usually around 20 ms, to ensure that signal is statistically stationary at each frame. This step includes some windowing function like Hamming window.

**S2** Fourier-transform each frame separately, stripping down the phase data. This can be done because perceptual studies have shown that amplitude is much more important than phase.

**S3** Convert each amplitude in log scale (i.e. in deciBel), because loudness perception has been found to be approximately logarithmic.

**S4** Smooth the spectrum to emphasize meaningful frequencies. This is done by grouping previously calculated amplitudes into frequency bins. Bin frequency grouping is not done by regular frequency spacing. Mel scale is used instead, thus grouping small ranges at lower frequencies, and big ranges at high frequency. At this step we have low number of bins, usually less than 40.

**S5** Now the mel-spectral vector on each frame contains highly correlated values. The typical use for MFCC vectors is to process them via Gaussian Mixture Models (GMM), and correlated data makes the GMM training harder to do and the resulting model to be less effective. A good way to decorrelate is to use the Discrete Cosine Transform (you can find a better explanation of this step in Logan's paper [23]) to the vector. The result is a 13 sized vector of MFCCs, with the first one representing the signal's energy.

### 1.3.3 Delta-MFCC

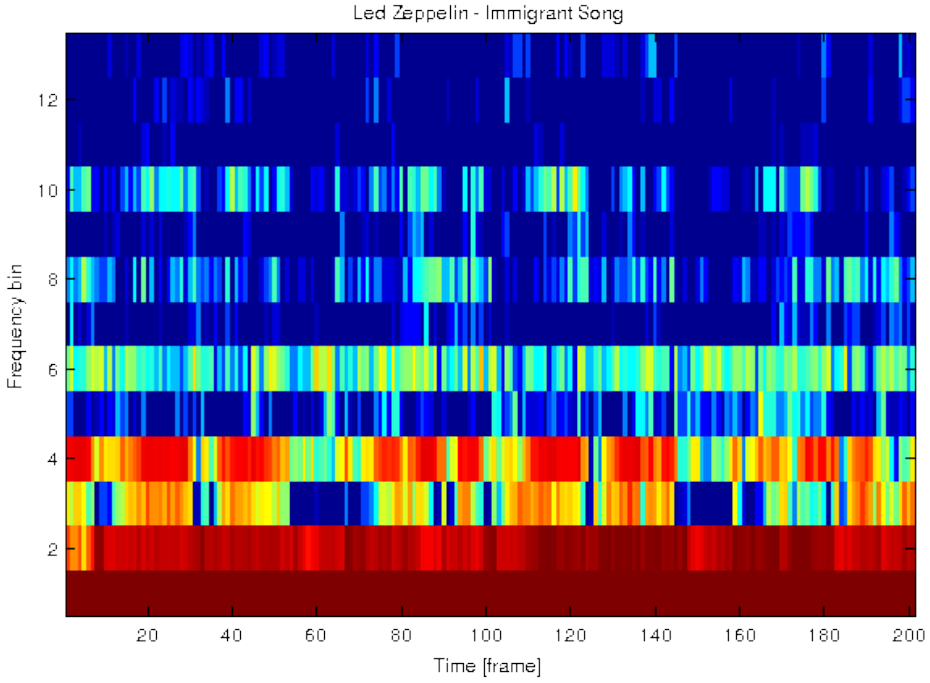
If we consider MFCCs as described before we can see they do not keep temporal information. That is, we cannot get information regarding variations in time by only looking at a single vector. To overcome this limitation we can use the first and second time derivatives of MFCCs: the delta- and delta-delta-MFCCs (we can simply call them "delta" and "delta-delta"). The delta MFCCs are computed via linear regression:

$$\Delta c[m] = \frac{\sum_{i=1}^{k} i(c[m+i] - c[m-i])}{2\sum_{i=1}^{k} i^2} \tag{1.39}$$

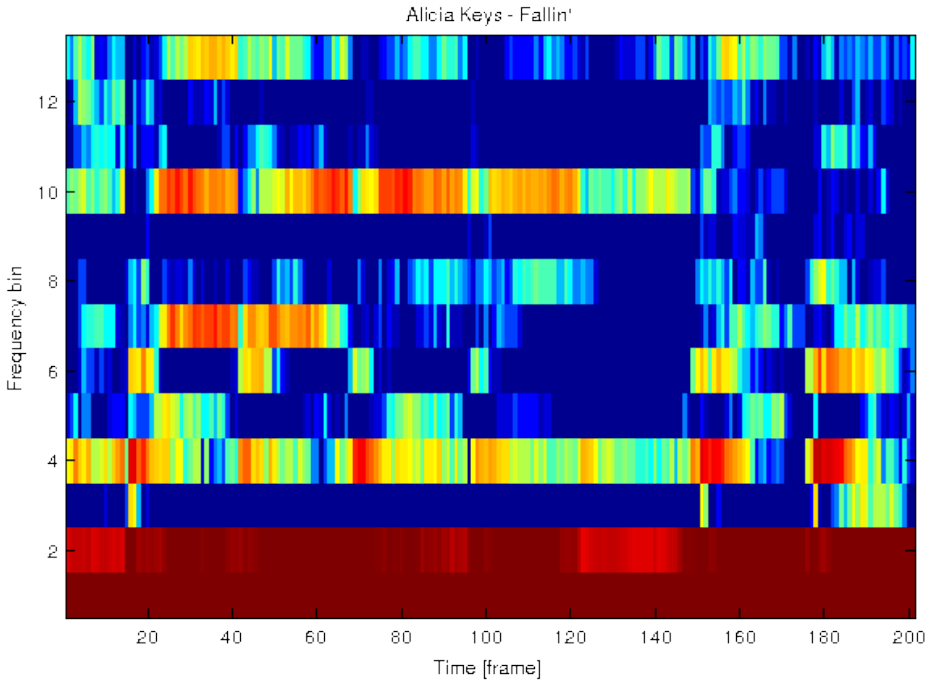
where c[m] is the m-th MFCC coefficient, and $2k + 1$ is the regression window size. Delta-delta feature is calculated by applying linear regression to the delta feature. This way we can account for MFCC variation over time and thus expanding the feature expressivity.

## 1.4 Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is a parametric statistical tool to approximate a probability density function. The model consists in a weighted sum of Gaussian distributions. The parameters of the models are the medians and the covariance matrices



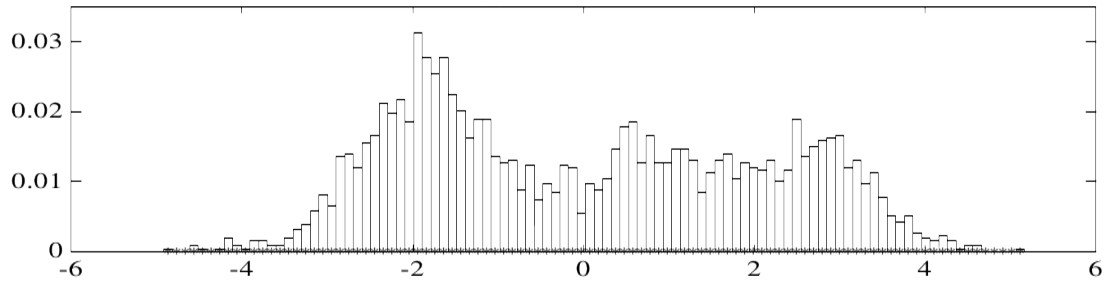

(a) MFCC plot for an except of Led Zeppelin's "Immigrant Song"



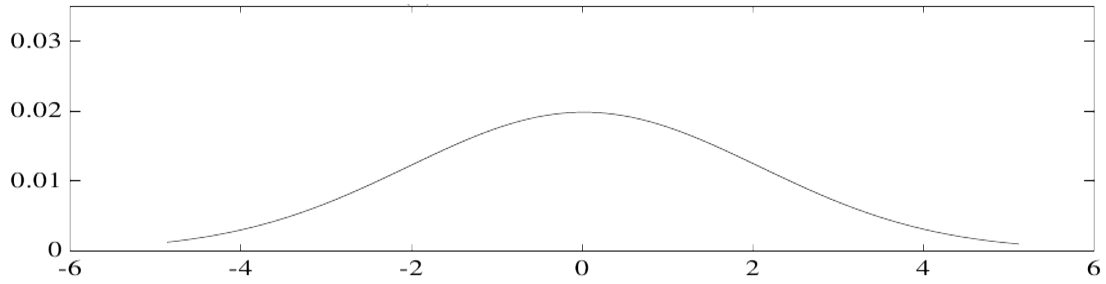

(b) MFCC plot for an except of Alicia Keys' "Fallin' "

Figure 1.3: Two examples of MFCC plots: we can see that different music pieces have different timbric signatures

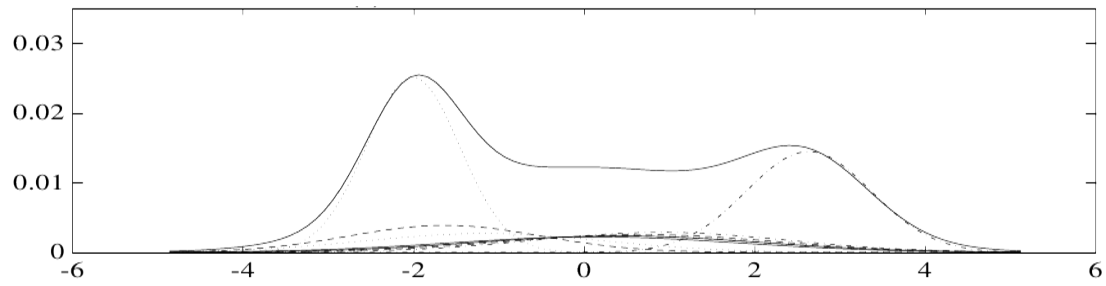
of the Gaussians, and the weights of the sum. GMMs are used to approximate a density function which is known to be composed of other known density functions.For example, it is known that humans' height (the feature) is normally distributed, but it is also known that males are generally taller than females: the overall density function is a gaussian, but it is composed by a mixture of two gaussian distributions. Training a GMM over some population's data could help guessing the sex of a given subject by knowing only its height. Another interesting feature of GMMs is that their output density is usually smooth, and its components are clearly highlighted by the weigths. You can see an example of the smoothing performance of GMMs in figure 1.4, which shows the smoothing of a single MFCC coefficient over time.




(a) Distribution of original data




(b) Unimodal Gaussian




(c) Resulting mixture

Figure 1.4: An example of Gaussian smoothing of a single cepstral coefficient from a 25 second utterance by a male speaker (courtesy of Douglas Reynolds, see [37])

Following a very good work by Douglas Reynolds (see [37]) we will introduce this useful tool that will be used in our work.

### 1.4.1 Definitions

The model's output probability density is defined as follows:

$$p(\mathbf{x} \mid \lambda) = \sum_{i=1}^{M} w_i \, g(\mathbf{x} \mid \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}), \tag{1.40}$$

where $\mathbf{x}$ is a $D-$dimensional data vector (the features), $M$ is the number of mixed Gaussians densities, each with its weight $w_i$. Each Gaussian density is a $D-$variate Gaussian density with mean $\boldsymbol{\mu_i}$,covariance matrix $\boldsymbol{\Sigma_i}$, and density function calculated with the standard formula:

$$g(\mathbf{x} \mid \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i}) = \frac{1}{(2\pi)^{D/2}|\boldsymbol{\Sigma_i}|^{1/2}} \exp\left\{-\frac{1}{2}(\mathbf{x} - \boldsymbol{\mu_i})' \, \boldsymbol{\Sigma_i}^{-1}(\mathbf{x} - \boldsymbol{\mu_i})\right\}. \tag{1.41}$$

Mixture weigths $w_i$ must of course sum to one.

### 1.4.2 Constrained versions

Depending on the amount of available training data and on the particular application, model's parameters can be constrained in different ways. For example, the covariance matrices $\boldsymbol{\Sigma_i}$ can be constrained to be diagonal (i.e. forcing the $D$ components to be independent), or to be all equal, or we may define some function that ties the parameters together. It is worth saying that forcing the covariance matrices to be diagonal does not really reduce the expressiveness of the model: the linear combination between diagonal covariance Gaussians can model correlation between feature vectors. This means you can model $M$ full covariance matrix Gaussians by using a larger number of diagonal covariance matrix Gaussians.

### 1.4.3 Training

Gaussian Mixture Models are used for classification based on multiple features. GMM training is the process that determines the model $\lambda = (w_i, \boldsymbol{\mu_i}, \boldsymbol{\Sigma_i})$ that better describe the probability distribution of the training dataset.

The most popular method to find the model parameters is the Maximum Likelyhood (ML) estimation. The method uses a sequence of training vectors $X = \{\mathbf{x_1}, \ldots, \mathbf{x_T}\}$

that are assumed statistically independents (obviously this is not true in general, but it makes problem tractable). What we need to maximize is the following:

$$p(X \mid \lambda) = \prod_{t=1}^{T} p(\mathbf{x_t} \mid \lambda). \tag{1.42}$$

Of course we cannot use linear programming to maximize this probability, this is a non-linear function. ML parameters can instead be estimated iteratively via Expectation Maximization (EM) algorithm. EM algorithm starts from an already known model and iteratively and monotonically increases the model's likelihood value at each step. The following re-estimation formulas are used at each step:

**Mixture Weights**

$$\overline{w}_i = \frac{1}{T} \sum_{t=1}^{T} P(i \mid \mathbf{x_t}, \lambda). \tag{1.43}$$

**Means**

$$\overline{\boldsymbol{\mu}}_i = \frac{\sum_{t=1}^{T} P(i \mid \mathbf{x_t}, \lambda) \mathbf{x_t}}{\sum_{t=1}^{T} P(i \mid \mathbf{x_t}, \lambda)}. \tag{1.44}$$

**Variances (covariance matrices' diagonals)**

$$\overline{\sigma}_i^2 = \frac{\sum_{t=1}^{T} P(i \mid \mathbf{x_t}, \lambda) x_t^2}{\sum_{t=1}^{T} P(i \mid \mathbf{x_t}, \lambda)} - \mu_i^2, \tag{1.45}$$

where $\sigma_i^2$, $x_t$ and $\mu_i$ refer to arbitrary elements of vectors $\boldsymbol{\sigma_i^2}$, $\mathbf{x_t}$ and $\boldsymbol{\mu_i}$ respectively. The *a posteriori* probability for component i is given by

$$P(i \mid \mathbf{x_t}, \lambda) = \frac{w_i \, g(\mathbf{x_t} \mid \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)}{\sum_{k=1}^{M} w_k \, g(\mathbf{x_t} \mid \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)}. \tag{1.46}$$

The initial model parameters can also be derived from an estimation using other methods (not covered here).

In this thesis we will use a modified version of EM algorithm that can learn mixture hierarchies from the training data. The whole process is not covered here, but you can look in the original paper from Vasconcelos *et al.* in [48] for further information.

# Chapter 2

# The model

In this chapter we will describe the model we use to semantically tag songs. The model itself is based on a work by Miotto and Orio ( [28] and [27]) which considered the relation between tags and acoustic similarity to build a retrieval framework. We have been inspired by the model and the way it combines two sources of information (tags and trimbric features), and we wanted to replicate the framework for a tagging system. The model is still similar but we covered another characteristic of music by adding a rhythmic feature. We have then developed a query procedure that exploits all the features together to guess the tags for a novel song. On the next section we will see the original model, complete with the retrieval process. Then we will see how we added a rhythm feature and how we modified the query procedure to tag songs. The last part will cover methods we wanted to try and method who had proven to be not useful.

## 2.1 Retrieval model

The goal for the retrieval model proposed by Miotto and Orio was to retrieve a list of meaningful songs in response of a user query. The user may ask for a certain type of songs, that is, naming the semantic tags of what he needs. Or he may ask for songs similar to a given one: this way we could infer the needed tags by looking at the one from the query song. The retrieval model of course needs a collection of known songs, from which it can learn acoustic and semantic characteristics. The database is also the source of the response songs: the system cannot suggest songs it does not "know" (well, actually it could output general suggestion like "you may like this artist", but usually it's not done).

What the user expects to *observe* is a list of meaningful songs, in a sense that

each proposed song has a positive relation with the query tags. This relation can be expressed as the adherence to a sequence of observation in time, that is, $\mathbf{O} = (\mathbf{o_1}, \ldots, \mathbf{o_T})$. In these terms $T$ is the number of requested songs, and $\mathbf{o_t}$ express what the user expect to see at time $t$. The query process consists in ranking the songs from the collection in order to maximize the probability to observe the given sequence over time.

In order to correctly rank the songs the model represents the collection as a graph where each state is a song. The retrieval process is modeled as a doubly embedded stochastic process: one process is hidden and can only be observed through a set of other processes which emit the observation sequence. In particular: each connection edge is weighted using the acoustic similarity between the songs (the states), and the emission symbols are the semantic tags.

The model as described so far resembles a Hidden Markovian Model (see section 1.1): at each time step the system undergoes a state (song) change according to the acoustic similarity, and emits a symbol (tag) according to the tag relevance for the given state. The path probability is a measure of the relevance of the path's songs for the given observation sequence (query tags). On the following paragraphs we will see more formally the model definition, and the retrieval process with details regarding acoustic similarity and tag management.

### 2.1.1 Definitions

In this section we will describe more formally the graph-based retrieval model at the base of this work. Following the analogy with HMMs the model $\lambda$ can be described as follows:

**States** Giving the collection has $N$ songs, we model each song as a state. States are named as in the set $\mathcal{S} = \{S_1, \ldots, S_N\}$, and state at time $t$ is denoted as $s(t)$.

**Observations** We model each tag as an observation symbol and because of the limitation of the HMMs the tag set must be fixed and finite. We name the observation vocabolary as $\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$.

**Transition probabilities** The transition probability $a_{ij}$ of moving from state $S_i$ to $S_j$ is related to the acoustic similarity between the song. The similarity has not been forced to be symmetric, so $a_{ij}$ may be different from $a_{ji}$. We will see this part in details on the next sections.

**Emission probabilities** The emission probability for each observation in each state is set according to the tag relevance for the given song. That is, each $b_i(w)$

represents the tag $w$ is associated to song $i$. This is set according to the ground truth and a further study on top of it (see next sections for a detailed view on this).

**Prior probabilities** The prior probabilities are not specified since they are set to different values depending on the query type (i.e. query-by-example or query-by-tag). Again, on the next sections we will see details on this.

A graphical representation of the model can be found in figure 2.1.

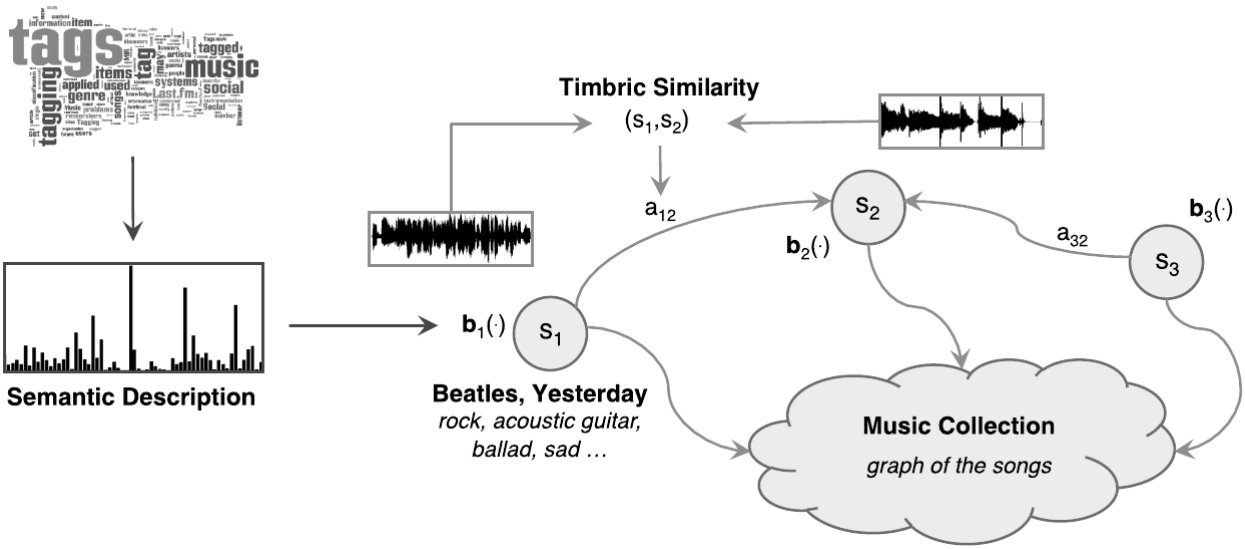



Figure 2.1: A graphical representation of the retrieval model

If we consider acoustic similarity as is we get a matrix composed of $a_{ij} > 0$ for each $(i, j)$ pair. With the aim to improve scalability this situation is modified by artificially keeping, for each song $S_i$, only a subset $\mathcal{R}(S_i)$ of connected states containing the first $P$ most similar song. That is, we keep the top $P$ highest transition probability for each state, and we put 0 (or an artificially low probability) on the remaining transition. The value of $P$ has been set to 10% of the global number of the songs. Investigation on this value has been made in [28] but we will not cover this here. At the end of this pruning operation the transition probabilities are normalized, so the statistical properties remain verified (see equation 1.4). Another approach would be capping the maximum number of reachable songs, which may be better for very large collections[1]. Because of this

[1]Lowering to 10% connections improves scalability but does not prevent linear growing of connections with the number of songs in the model, hence a fixed capping may be better from a certain (big) model size

modification the similarity matrix (i.e. the transition matrix) may be not symmetric. This asymmetry can be interpreted as follows: there are songs which had inspired a lot of other songs, being covered by other artists or by using the same musical structure. Each cover version may be very similar (and thus related) to the original song, but the latter may be only loosely related.

### 2.1.2 Acoustic similarity

The acoustic similarity is defined in terms of timbral features. We make use of MFCCs (see section 1.3) to capture the timbral signature of each song. These features has been successfully used for speech recognition and a work by Beth Logan (see [23]) showed that they can be used successfully in music too.

We specifically capture MFCCs using a 23 ms, half-overlapping windows over at most 4 minutes of music. We take the music from the middle of the song after stripping down the first and last 20 seconds of music in order to exclude the characterization of the lead-in and lead-out phase of the song. The output is a 13-sized vector for each window, but since we include the first and second derivatives, we end up with a matrix of 39 elements multiplied by the number of windows.

What we need is a way to compute the similarity between two songs, that is, to produce a real number in the $[0, 1]$ interval that is closer to 1 the more similar the songs are. There are a lot of well known similarity measures, but these are usually able to compare at most two distributions. Since our song feature is a (rather big) matrix and we need to first extract a representative distribution from it, and then use this distribution to compare two songs.

For our model we use an approach based on a work by Mandel and Ellis (see [25]) which treats MFCCs from each frame as words on a dictionary. We calculate mean and covariance of the MFCCs over the length of the song and the result is a representative multivariate Gaussian distribution of the song. To compare two distibution we use the Kullback-Leibler divergence (KL divergence):

$$KL(p \,\|\, q) \equiv \int p(x) \log \frac{p(x)}{q(x)} \, \mathrm{d}x, \tag{2.1}$$

where $p(x)$ and $q(x)$ are the two distributions. The KL divergence cannot be used as-is as a metric (and a similarity measure), it has to be transformed to a $[0, 1]$ value, with 1 representing two identical songs. To do this we exponentiate the divergence to obtain the similarity :

$$sim(p, q) = e^{-\gamma KL(p \,\|\, q)}, \tag{2.2}$$

where $\gamma$ is a parameter to be tuned (for this work it has been set to 0.009). The fact that KL divergence is asymmetric reflects the music scenario as described in the previous section.

As said in section 2.1.1 we need to limit the number of connected songs in order to improve scalability of the system. To do so we limit transition probabilities for each song $S_i$ as follows:

$$a_{ij} = \begin{cases} a_{ij} & \text{if } j \in \mathcal{R}(S_i) \\ 0 & \text{elsewhere} \end{cases} \forall i, j = 1, \ldots, N, \tag{2.3}$$

where $\mathcal{R}(S_i)$ is the number of connected songs as defined in section 2.1.1. Then we normalize the remaining transition probabilities so they sum to one.

### 2.1.3 Tag model

We describe a song in terms of semantic tags from a finite vocabulary $\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$. We take these song-tags association from a set of already annotated songs, the ground truth. The latter usually marks a tag for a song as either present or not, which lead to a sort of boolean matrix of song-tag association. The problem is that this representation does not express the tag relevance for a given song and also present some issues while comparing two songs' tags. What we need is to measure the relevance of these tags for each song, obtaining what we call a Semantic MultiNomial (SMN), a vector of weights which is a smoothed version of the original ground truth.

In this work we calculate the SMN for each song using an autotagging approach. Autotagging itself is obviously the objective of this work, but here we just need a smoothed version of the SMN. Following a work by Turnbull *et al.* (see [44]) we try to infer some relation between a tag and music features in songs that are positively related with that tag. To do this, Turnbull proposes a model which relates, for each tag $w_i \in \mathcal{V}$, a probability distribution on the space of the music features. The model is based on Gaussian Mixture Models (see section 1.4) and the probability distribution can be written as follows:

$$p(\mathbf{x} \mid w_i) = \sum_{r=1}^{R} a_r^{w_i} \mathcal{N}(x \mid \boldsymbol{\mu}_r^{w_i}, \boldsymbol{\Sigma}_r^{w_i}), \tag{2.4}$$

where $\mathbf{x}$ are the audio features, $R$ is the number of mixture components, $\mathcal{N}(\cdot \mid \boldsymbol{\mu}, \boldsymbol{\Sigma})$ a multivariate Gaussian distribution with mean $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and $a_r^{w_i}$ the mixture weights for mixture component $r$ and tag $w_i$. The model's parameters $\{a_r^{w_i}, \boldsymbol{\mu}_r^{w_i}, \boldsymbol{\Sigma}_r^{w_i}\}_{r=1}^{R}$, are estimated from the timbral features (MFCC) using Hierarchical Expectation Maximization using the songs positively associated for each tag $w_i$.

Once we have trained the GMM we can compute the relevance of each tags based on the music features $\mathcal{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ using the Bayes' rule:

$$P(w_i \mid \mathcal{X}) = \frac{p(\mathcal{X} \mid w_i)P(w_i)}{p(\mathcal{X})}, \tag{2.5}$$

where $P(w_i)$ is the tag prior probabilty (which is assumed uniform) and $p(\mathcal{X})$ is the song prior, that is, $p(\mathcal{X}) = \sum_{j=1}^{|\mathcal{V}|} p(\mathcal{X} \mid w_j)P(w_j)$. After this process, normalizing to 1 if needed, we have the SMN for each song.

In the query-by-example process we use the query song's tags as the observation sequence, so we do not actually see only one tag per observation. We have a full SMN instead, so we treat the SMN as an observation and we need to emulate the observation probability (i.e. the SMN) $b_i(o_t)$ by comparing two songs' SMNs. To do this we can make use of the discrete version of Kullback-Leibler divergence, since it expresses a distance for probability distributions:

$$D_{KL}(\mathbf{b}_x(\cdot) \parallel \mathbf{b}_y(\cdot)) = \sum_{k=1}^{|\mathcal{V}|} \mathbf{b}_x(k) \log \frac{\mathbf{b}_x(k)}{\mathbf{b}_y(k)}, \tag{2.6}$$

where $\mathbf{b}_x(\cdot)$ and $\mathbf{b}_y(\cdot)$ are the two SMNs for songs $x$ and $y$ and $\mathcal{V}$ is the tag vocabulary. Of course we need a similarity measure, not a distance, so we use the inverse of the KL divergence as observation relevance $\phi_x(\mathbf{b}_y(\cdot))$ for song $x$ with song $y$:

$$\phi_x(\mathbf{b}_y(\cdot)) = \frac{1}{D_{KL}(\mathbf{b}_x(\cdot) \parallel \mathbf{b}_y(\cdot))}. \tag{2.7}$$

### 2.1.4 Retrieval process

As we have seen in the previous sections, the retrieval problem is similar to Problem 2 for HMMs (see section 1.1.3): what we need is a state (song) sequence which better explains an observation sequence. The query can be formally expressed as a sequence of observations $\mathbf{O} = (\mathbf{o}_1, \ldots, \mathbf{o}_T)$, where $T$ is the expected number of response songs. The output song list must of course be ranked according to the relevance with the query sequence.

The way to solve this problem is to use a modified Viterbi algorithm. This algorithm is composed of two procedures called forward and backward (see section 1.1.3) that calculate first the probability of the most likely state sequence, and then decode the path which gives this probability. To understand seamlessly the procedure we will use a naming convention which resembles the one we used for HMMs.

Input for the problem must be described in both query-by-tag (also called query-by-description) and query-by-example cases. The observation sequence $\mathbf{O} = (\mathbf{o}_1, \ldots, \mathbf{o}_T)$,

is defined such that for each $\mathbf{o}_t$:

$$\mathbf{o}_t(\cdot) = \begin{cases} \{(o_t^1, \ldots, o_t^k) \mid o_t^y = w \in \mathcal{V},\ y = 1, \ldots, k\} & \text{if k-tag query-by-description} \\ \mathbf{b}_q(\cdot) & \text{if query-by-example with song } q \end{cases} \tag{2.8}$$

where $\mathbf{b}_q(\cdot)$ is the tag relevance (SMN) for this song. In the query by description equation we assume that each tag mentioned by $o_t^y$ is considered as requested, and that the query does not specify any weight on tags (it is like query is a uniform SMN).

Looking at section 1.1.3 we also see that we need to define a way to account for prior probabilities $\pi_i$ and observation probabilities $b_i(\cdot)$. Because we must handle different situations (query by tag and query by example) we defined a function $\phi_i(t)$ that expresses the observation probability for song $i$ and observation at time $t$ and the prior probabilities at the same time as:

$$\phi_i(t) = b_i(o_t^1) \cdot b_i(o_t^2) \cdot \ldots \cdot b_i(o_t^k) \ \ \forall t > 0 \qquad \text{if query by description} \tag{2.9}$$

$$\phi_i(t) = \begin{cases} 1 & \text{if } t = 1 \wedge i = q \\ 0 & \text{if } t = 1 \wedge i \neq q \\ \frac{1}{D_{KL}(\mathbf{b}_x(\cdot) \,\|\, \mathbf{b}_y(\cdot))} & \text{if } t > 1 \end{cases} \qquad \text{if query by example,} \tag{2.10}$$

where $q$ is the query song in the query by example situation.

In the general query by description situation the query consists in a sequence of observations, each one composed of a k-tag observation (see equation 2.8). We can see that equation 2.9 accounts for each observation vector by multiplying each tag relevance from its SMN description. On the other hand in the query by example scenario we have a reference (seed) song with its already known tags[2]. In this situation we firstly force all paths to start from the seed song by setting $\phi_{\text{seed}}(1) = 1$. The next step is to define the observation probability: since we are seeking for songs similar to the seed song, we want to be sure that the observation probability $\phi_i(t)$ reflects the tag similarity of song $i$ and tags from $\mathbf{o}_t$. Equation 2.10 does this by exploiting the inverse of the KL divergence as defined in section 2.1.3.

### 2.1.5 Querying the model

Once we have defined the naming convention we can describe the querying algorithm. To query the model we use a modified Viterbi algorithm (see the original version in section 1.1.3) which is composed of three parts:

[2]We assume that we are querying the model using a song from the ground truth collection

**Initialization:** for all $i = 1, \ldots, N$:

$$\delta_1(i) = \phi_i(1) \tag{2.11}$$

$$\psi_1(i) = 0 \tag{2.12}$$

**Recursion:** for $t = 2, \ldots, T$, and $i = 1, \ldots, N$:

$$\delta_t(i) = \max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ji}]\, \phi_i(t) \tag{2.13}$$

$$\psi_t(i) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ji}] \tag{2.14}$$

$$a_{ri} = \frac{a_{ri}}{\eta} \quad \text{for} \quad r = \psi_t(i),\ \eta = 10 \tag{2.15}$$

**Decoding (backtracking):**

$$s(t)^* = \begin{cases} \arg\max_{1 \le i \le N} [\, \delta_t(i)\, ] & \text{if } t = T \\ \psi_{t+1}(s(t+1)^*) & \text{if } t = T-1, T-2, \ldots, 1, \end{cases} \tag{2.16}$$

Where $N$ is the number of songs in the database, $\delta_i(\cdot)$ and $\psi_t(\cdot)$ are the forward variables in the Viterbi algorithm and $s(t)^*$ are the output songs. The result of the algorithm is a ranked list of songs $\{s(1)^*, s(2)^*, \ldots, s(T)^*\}$.

As we can see equations 2.11 and 2.13 make use of the previously defined observation probabilities (see equations 2.9 and 2.9) to account for relation between query tags and songs ones. Another notable difference from original Viterbi algorithm is equation 2.15: when we search for an optimal path in the songs graph it could happen (it actually happens a lot) that the path loops between two or more songs. This is a consequence of the transition probabilities and the query. If the combination between the two induces a looping nothing can prevent it because Viterbi algorithm searches for maximum path probability, and it does not care about loops. As a solution to this we temporarily lowered the transition probability of most probable path (i.e. the chosen transition) by dividing it by a fixed amount $\eta = 10$. This decreases the probability of choosing the same transition on the path for the query, thus preventing looping. Of course this modification is temporary and the original transition probabilities are restored after querying the model. This solution happens to be effective but one could argue that dividing by a fixed amount do not actually cover all possibile situations: it could happen that the second maximum transition probability is actually more than $\eta$ times lower than the maximum (e.g. maximum value is 0.80, second maximum value may be 0.07) thus making equation 2.15 useless. A solution may be to consider some statistic over the transition probability (e.g. the variance or the max value) and use

that for parametrically reduce the value of the chosen transition's probability. We will discuss later on this topic in the conclusions. It should be pointed out that this kind of modification actually breaks guarantee of path's optimality, but it does not affect the relevance of the output songs to the query.

There is also another difference from Viterbi's algorithm: Miotto and Orio noted that the output precision of the original algorithm was rather low expecially when dealing with long paths through the graph. They observed that long paths have the tendency of losing relation with both query tags and first selected songs. They resolved this issue by splitting the retrieval process in little substeps: each step does retrieve a little number of songs (e.g. 5) and the following step resets the forward variables and forces the first retrieved song to be the last retrieved from the previuous step. At the same time the steps after the first ignore the already retrieved songs while forming the output ranked list. The aim of this solution is to keep some form of locality between results, thus mantaining correlation with both query and first songs.

## 2.2 Semantic music tagging

In this thesis we want to present a method for semantic tagging of music pieces. As we said in the introduction, the role of semantic music tagging is to provide some meaningful description (tags) to a novel music piece. We want to do this by exploiting the existing Markovian-like retrieval model by Miotto and Orio, by adapting it to tag new songs. The retrieval model has been developed on the assumption that tags and timbre are in a positive correlation. This assumption was not so weak, as they observed that exploiting tag similarity can increase the performance of a timbre-based retrieval. We also wanted to add a feature to cover another characteristic of music pieces: the rhythm description. To account for this we have introduced Rhythm Histograms, a feature proposed by Rauber and Lidy (see [21]) which was been proven valid for genre classification. The querying process was also modified, since the input of the model is different (i.e. it is a new song) and the expected output is different too (i.e. tags instead of songs).

The first issue we meet is how we define the input and the output of the problem: the input is a song which is not in our collection, and the output is a weighted set of tags related to the query song. The first thought about how we could tag is to use the acoustic similarity: the retrieval model suggests that acoustically similar songs have similar tags. So we thought about "placing" the query song in the collection graph in some meaningful way and then exploring the "best" paths (in some sense) around the query to guess the tags by the ones from neigbour songs. This is basically what we

ended up doing, but we had to define that "place", "path", and "best" mean.

### 2.2.1 Definitions

The model itself is based on the retrieval model, so some terms remains the same as before, while others are re-defined to reflect the tagging task. At any time the system rely on a set of known songs which serves as both the source of the tags and the similarity model. This means that at the boostrapping phase we assume we have some previous knowledge, which is composed of a set of known songs, with proper tags on them.

Analogy with Hidden Markovian Models remains, so the model itself is named $\lambda$ as before, and can be described as follows:

**States** A song is represented as a state $S_i$ from a set $\mathcal{S} = \{S_1, \ldots, S_N\}$, and, as in the retrieval model, state at time $t$ is denoted as $s(t)$.

**Transition probabilities** As before, the transition probabilities are related to the acoustic similarity between the songs. We have chosen to use the timbric similarity as it is defined in the retrieval model. Again, the similarity is not forced to be symmetric for the reasons mentioned in subsection 2.1.2. They are named as before, and $a_{ij}$ stands for the acoustic similarity of song $S_i$ with song $S_j$.

**Observations** The notion of observation is different from the previous model: since we need tags as output and not as input, we cannot use them as observation. We have chosen to use an audio feature that captures a different characteristic of a music piece: the rhythm. However, we could use any other characteristic, the main reason for choosing this was that rhythm (or, more specifically, Rhythm Histograms) is an effective feature for genre classification. We also wanted to see if the use of two features together gives better results than the single features used individually. We will cover better this aspect on next sections.

**Observation probabilities** As said before, we do not have conventional observations. Since we simulate observation with rhythmic description of a song, we model observation probability as rhythmic similarity. That is, instead of calculating the probability of a state to emit a particular symbol, we replace this probability with the rhythm similarity between the query song and the state song. This means that in a random walk in the graph the path probability is influenced by both rhythmic and timbric similarity. As a consequence of the analogy with HMM, rhythm plays a bigger role than the timbre in discriminating the songs. This

happens because rhythm is compared to query at each song (it is an observation), while timbric similarity is progressively multiplicated from previous steps, so it decades as the path grows longer. However, this does not necessarily mean that songs along the path have low timbric similarity with the query. We name these "observation" probabilities $\phi_i(j)$ for song $S_i$ observing the rhythm of song $S_j$.

**Prior Probabilities** Prior probabilities play a big role in the query process: they help choosing the first states for the paths. With the idea of starting from a song which is acoustically similar to the query we first simulate the insertion of the query in the model, and then we force the paths to start from the query. That is, we calculate the timbric similarity between each song and the query, we treat it as a song in the database, and then we set the prior probability of the query to 1 and the others to 0. In this way we force the model to calculate the similarity of both timbre and rhythm, and we also have a suggestion for how the model could grow.

**Tags** The foundamental thing that is not included in the Hidden Markovian model are the tags: at every instant the models has information regarding the tags of the known songs. These are stored on a separate structure and are needed to tag novel songs. We name that structure $b_i(j)$, which stands for tags for song $S_i$ in the collection. We assume that the vocabulary of tags consist in $M$ different symbols $\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$ where $M = |\mathcal{V}|$. The vector is in binary form, it contains a 1 in position $j$ if tag $j$ is present, 0 if not.

A graphical representation of the model can be found in figure 2.2.

It should be noted that every time we mentioned some similarity measure with the query or between known songs we also imply that statistical properties must be satisfied, so every value has to be normalized in order to treat the model as an HMM.

Regarding the transition probabilities: we mantain the pruning of the transition probabilities for scalability purposes, as seen in the end of section2.1.1.

As a difference with the retrieval model we make use of binary tags and not a semantic multinomial as seen in section 2.1.3, that is, a tag can either be present or not for a specific song . We will also see that having the tags in a separate structure let us grow the model easily if there is a novel tag (e.g. the song is partly tagged).

### 2.2.2 Adding rhyhtm features

We have chosen to use Rhythm Histograms (see section 1.2.2) as a rhythm descriptor for a given song. This feature has been developed by Rauber *et al.* in [21] and has

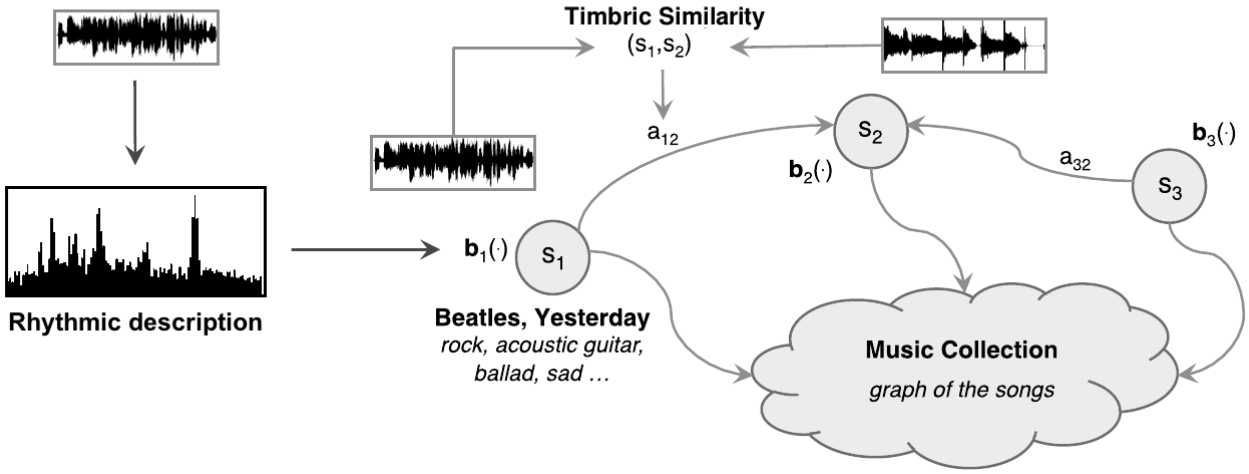



Figure 2.2: A graphical representation of the tagging model

been successfully used for genre classification. With the reference from section 1.2.1 we have calculated RH using steps S1, S2, S3, S4, R1 for rhythm patterns calculations, and then we summed the values from each frequency bin to obtain the Rhythm Histograms. Since we have done a slightly different calculation we obtain a 120 values vector representing the amplitude modulation frequencies between 0 and 10 Hz. We have calculated this feature over only a 6 second segment of music after the first 30 seconds. This was initially done for testing and then kept in view of future experiments with Amazon APIs, which provide short music samples taken after roughly 30 seconds of the songs[3].

### 2.2.3 Similarity measures

Since we use RH as observation we must define the observations vocabulary, but we can clearly see that this time the vocabulary is not finite: it is in fact a vector of floating numbers. In order to compare two songs (namely the query and the state) we need to define some similarity measure between their RH, which will be used in lieu of the observation probability.

What we want is a measure that expresses the similarity of two Rhythm Histograms with a number in the [0, 1] range. We expect that the closer to 1 the similarity value is,

[3]We have also thought about scraping the Amazon catalogue to test our system, but we did not try it yet

the more similar the two rhythm are. We denote the similarity of RH $x$ (the one from query song) with RH $y$ as $\phi_x(y)$, and the $i$-th value from a RH vector as $x_i$. We can now see which similarity measures we have tried:

**Cosine** The standard metric for information retrieval:

$$\phi_x(y) = \frac{\sum_{i=1}^{|x|} x_i y_i}{\sqrt{\sum_{i=1}^{|x|} x_i^2}\sqrt{\sum_{i=1}^{|y|} y_i^2}}. \tag{2.17}$$

**Euclidean distance** This is based on the standard euclidean distance:

$$\phi_x(y) = 1 - \|\mathbf{x} - \mathbf{y}\| = 1 - \sqrt{\sum_{i=1}^{|x|} |x_i - y_i|^2} \tag{2.18}$$

**Bray-Curtis** This similarity was named after the two biologists J. Roger Bray and J.T. Curtis that had invented it in [4].

$$\phi_x(y) = \frac{2\ \sum_{i=1}^{|x|} \min(x_i, y_i)}{\sum_{i=1}^{|x|}(x_i + y_i)} \tag{2.19}$$

**Modified Chord** This is a variation of the cosine similarity that compresses the low and highlight the high similarity values.

$$\phi_x(y) = 1 - \sqrt{1 - \phi_x^{\text{cosine}}(y)} = 1 - \sqrt{1 - \frac{\sum_{i=1}^{|x|} x_i y_i}{\sqrt{\sum_{i=1}^{|x|} x_i^2}\sqrt{\sum_{i=1}^{|y|} y_i^2}}}. \tag{2.20}$$

**Ruzicka similarity index** Ruzicka similarity was developed by the Croatian scientist Lavoslav Ružička in a work about geobotany (see [38]).

$$\phi_x(y) = \frac{\sum_{i=1}^{|x|} \min(x_i, y_i)}{\sum_{i=1}^{|x|} \max(x_i, y_i)} \tag{2.21}$$

**Similarity ratio**

$$\phi_x(y) = \frac{\sum_{i=1}^{|x|} x_i y_i}{\sum_{i=1}^{|x|} x_i^2 + \sum_{i=1}^{|y|} y_i^2 - \sum_{i=1}^{|x|} x_i y_i} \tag{2.22}$$

**Kulczynski similarity**

$$\phi_x(y) = 1 - \frac{\sum_{i=1}^{|x|} |x_i - y_i|}{\sum_{i=1}^{|x|} \min(x_i, y_i)} \tag{2.23}$$

**Kullback-Leibler distance** This is based on the already presented KL divergence.

$$\phi_x(y) = e^{-D_{KL}(\mathbf{x}\|\mathbf{y})} = \exp\left(-\sum_{i=1}^{|x|} x_i \log \frac{x_i}{y_i}\right) \tag{2.24}$$

It is worth saying that some of them suffer some technical problems in specific situations, thus leading to some caveat in the implementation. For example, KL divergence in some situations can lead to a $0 \log 0$ multiplication, and Kulczynski similarity is prone to the division by zero. These caveats are often resolved in a way that interferes with the measure itself: for example a divison by zero may be avoided by adding a (little) constant value to the denominator.

It should be also noted that different similarities have different computational cost: those which include calculation of square roots (cosine, modified chord) or logarithms (KL), tend to cost more than a simple minimum or maximum (Ruzicka). This has almost no influence with a low number of song in the model, like in our tests, but can be quite costly as the collection grows bigger.

### 2.2.4 Querying the tagging model

Now that we have defined the model we can see how we can tag a novel song. As before we use a modified version of the Viterbi algorithm (see the original in section 1.1.3 and the retrieval version in section 2.1.5) and we will also make use of the tags which are held in a separate structure (see previous sections). At any time, when we want to refer to the query song, we use the subscript $q$ (e.g. $S_q$ for the song, $b_q()$ for the tags, and so on) which of course stands for "query". We also assume that the vocabulary of tags consists in $M$ different symbols $\mathcal{V} = \{w_1, w_2, \ldots, w_{|\mathcal{V}|}\}$ where $M = |\mathcal{V}|$.

**Initialization** As we said in section 2.2.1 we simulate the insertion of the query song in the model in order to tag it. So if we assume that at this time there are $N$ song in the collection the initializazion of the foward variables is, for all $i = 1, \ldots, N$ :

$$\delta_1(i) = \begin{cases} 1 & i = q \\ 0 & i \neq q \end{cases} \tag{2.25}$$

$$\psi_1(i) = 0. \tag{2.26}$$

Where $q$ denotes the query song.

**Recursion** Here we take the same steps as the retrieval model. For $t = 2, \ldots, T$, and $i = 1, \ldots, N$:

$$\delta_t(i) = \max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ji}]\, \phi_q(i) \tag{2.27}$$

$$\psi_t(i) = \arg\max_{1 \le i \le N} [\delta_{t-1}(i)\, a_{ji}] \tag{2.28}$$

$$a_{ri} = \frac{a_{ri}}{\eta} \quad \text{for} \quad r = \psi_t(i),\ \eta = 10. \tag{2.29}$$

Where $\phi_q(i)$ is the rhythm similarity between the query and the given song, calculated using the current similarity measure. We can see that we do the same as in the retrieval to avoid looping in equation 2.29. Some consideration regarding this type of probability cutting are made in section 2.1.5.

**Decoding (backtracking)** Decoding is the same as in the retrieval model, but we also keep the path probability at each step.

$$s(t)^* = \begin{cases} \arg\max_{1 \le i \le N} [\, \delta_t(i)\, ] & \text{if } t = T \\ \psi_{t+1}(s(t+1)^*) & \text{if } t = T-1, T-2, \ldots, 1, \end{cases} \tag{2.30}$$

$$p(t)^* = \begin{cases} \max_{1 \le i \le N} [\, \delta_t(i)\, ] & \text{if } t = T \\ \delta_t(s(t)^*) & \text{if } t = T-1, T-2, \ldots, 1. \end{cases} \tag{2.31}$$

Here $p(t)^*$ represent the probability at each step of the best path. This time $T$ is a parameter, which has to be tuned: the length of the path affect the similarity between the first and the last song in the path, so we may want to keep the value low.

**Tagging** Now we will see how we tag new songs: we assume that a song that is similar to the query in terms of both timbre and rhythm has very similar tags. This assumption lets us infer the tags for the new songs by looking at the tags from songs in the best path in the model (in a Markovian sense). In fact, a song in the best path has is very likely to have a high timbric and rhythmic similarity with the query. Since we have included the song in the model in order to tag it, it is possible that the path includes one or more times the query song. Of course we can not take in account any contribution from the query since it does not have any tag, so we have to skip it. So to tag the novel song we take the $T$ songs extracted with equation 2.30 and we calculate a vector of tag weights $b_q(j)$ for all tags in $j = 1, \ldots, M$ as in:

$$b_q(j) = \sum_{i=1, i \ne q}^{T} b_i(j)\omega(i). \tag{2.32}$$

In this equation $\omega(i)$ is the weighting function of the tags: this is a weakly decreasing monotonic function of the path position $i$. That is, tags from songs acoustically similar (low $i$) to the query should have more importance than songs that are not that similar (i.e. that are later in the path). The $i \neq q$ is needed because the song has initially no tags, and then to avoid any self-contribution, which has no semantic meaning.

**Iterating** As the retrieval model in section 2.1.5 the tagging model also take action against the probability decay of long paths. Instead of splitting the path research in little substeps, making the model restart from the last visited state as in a "chain", we just restart the Viterbi procedure. We keep $T$, the maximum length of a path, to a low number (e.g. 4) and we launch the Viterbi-like procedure to tag the song. Then we restart from the initialization step (thus restarting from the query song as before) without resetting the modified transitions (i.e. keeping the effects from equation 2.29) and keeping the tags weights calculated so far. The calculated tag weights are summed at each iteration. We repeat this process for a number of iterations (e.g. 10) to be sure we inspect enough similar songs to correctly infer the query's tags. This variation has been developed because we got poor results from the "chained" version of the Viterbi procedure.

We have chosen the Viterbi algorithm because it gives us a method to link the influence of two features at once. We could argue that we can calculate query's tag using all the neighbour songs, the transition and the observation probabilities, that is, we can just do a weighted sum of the tags from all the songs using a forward exploration[4] from the query. The advantage we get from using Viterbi is the fact that it finds an acoustically meaningful "music path" from the query, which helps us avoiding non-related songs. Another important advantage is that we can also decide which weight we want to assign to each song in the path, which could lead to better results, as we will see in the results chapter.

It has to be noted that there is no real contol of the paths extracted at each iteration, in a sense that there may be songs that appear multiple times on different iterations or maybe multiple times on the same iteration (see limitations on avoiding looping in section 2.1.5). This as an desired behaviour: if a song is chosen multiple times we assume that it is somehow "very similar" to the query, and so its tags may be better related to the query. In fact, we sum the tags contribution of these songs every time they are chosen, regardless of the number of times and the position in the paths they appear.

[4]In graph theory the forward star of a state is defined as its outgoing edges

For the special case of paths with loops involving the query multiple times (thus making the path useless for tagging) we do the following workaround: we calculate a path of length $T$ for each iteration, but we retain only $K$ songs. This way we can expect that equation 2.29 does its effects and let the path deviate towards other songs while still offering at least $K$ songs different from query. An alternative workaround may be thought for this, but we will discuss it after the conclusions.

As of the weighting function in equation 2.32 we have different options, and we have tried four different functions so far:

**Path probability** We can use the path probability at step $i$ as the weight:

$$\omega(i) = p(i)^*. \tag{2.33}$$

This has the advantage that tag weighting reflects the acoustic similarity in a direct way.

**Linear decay** We have defined a linear decading function of the path step:

$$m = -c/T, \qquad c < 1 \tag{2.34}$$

$$\omega(i) = 1 + m(i-1), \tag{2.35}$$

where $T$ is the path length and the coefficient $c < 1$ is needed to give the last weight a value above 0 and the $(i-1)$ lets the first weight to be 1. We set $c$ to 0.9 in our experiments.

**Exponential decay** We have defined an exponentially decading function of the path step:

$$\omega(i) = a^{(i-1)}, \tag{2.36}$$

where the $(i-1)$ lets the first weight to be 1. The parameter $a$ should be $0 < a < 1$, we have set it at $1/2$.

**Hyperbolic decay** We have defined an hyperbolic decay function:

$$\omega(i) = 1/i \tag{2.37}$$

# Chapter 3

# Results

An automatic tagging system is expected to put meaningful tags on novel songs in a reliable way. We have already seen the importance of automatic tagging in the introduction and we wanted to test how our model performs in this difficult task. In this chapter we will show some results obtained with the model presented on this thesis. We will first describe the test setup, including the source data and the evaluation measures. Then we will show the results of the tests with a comment on what we obtained in respect of what we expected.

## 3.1 Test setup

To test our model we need to first find a source of already tagged music as the model infers tags from already known songs. This prior knowledge constitutes also the ground truth on which we can test the results of the tagging process. We will see now the data source we have chosen, the evaluation measures we used, the tests we have made and a little comment on the tuning of the model's parameters.

### 3.1.1 Data source

We have different options regarding the initial data source, but we thought about some restrictions. We put an accent on the quality of the source data, as we need to rely on it to put the right tags to songs. For these reasons we have chosen the CAL500 dataset as the source of our data. This dataset consists of 502 popular songs of Western music my different artists, and has been collected by Turnbull *et al.* in [45] for a work on query-by-tag.

Songs from this dataset have been tagged through a contolled survey, where every song has been examinated by at least three human annotators. The semantic vocabu-

lary consists of 149 tags spacing from genre classification (e.g. “rock”, “pop”) to vocal and acoustic characteristics (e.g. “female lead vocals), as well as emotions (e.g. “aggressive”) and song usages (e.g. “studying”). The survey results is a binary annotation of each song, which marks a 1 for each tag applies to that song, 0 otherwise. It should be noted that in order to mark a tag for a song at least 2 annotators must vote that tag, so we can be quite confident about this data.

It is also important to note that we have full acces to all the audio clips from the dataset, but for copyright reasons the clips are a degraded version of the original[1]. We need this because it permits us to calculate every feature we may need, MFCCs and Rhythm Histograms in our case.

There are of course other useful datasets, but some of them does not meet our needs. For example, the CAL10k database (see [43]) provides around 10000 songs with manually annotated tags. These tags are mined from Pandora service as part of the Music Genome Project and are considered objective. However, this database does not give access to the audio clips: instead, it gives an EchoNest TrackID for each song. Echo Nest is a music data service company, which supports other music Companies such as streaming services with their data. It provides acoustic data on songs via some public API. However, this data does not include the audio clip of the songs, so we cannot get features besides the ones they expose.

The Magnatagatune database is another example of set which permits access to the the music clips, and it can do this because it uses music released under Creative Commons. It also provides human annotated tags to the songs, but they did not employed music experts to do this: instead they have written a social music game where users can tag the songs. This makes the database useful but unreliable, so we skipped it.

Another option would be using some large source of music and relying on social music services like LastFM to provide tags. But then again, we would face the reliability problem: we need expert evaluation to get a good prior knowledge.

### 3.1.2 Evaluation measures

To measure the performance of our model we need to define what is important for an automatic music tagging system to do. What the users expect from this system is, naturally, that proposed tags are relevant, in a sense that they truly describe the content of the song, and the tags are complete, so there is no lack of information. This also means that any extra tag that the system propose for a song is counted as an error, or

[1] We have access to the MP3 version of the songs, however the encoding quality is high enough for our purposes

at least as noise. We also expect that a system for automatic tagging has to be concise, that is, if the output of the system is a ranked list of tags, where the rank is a relevance measure, we may want to keep only the top most tags as the query result. In other words, we need to measure whether the system is able to propose the most relevant tags at the top of the ranked list, while other tags (not-so-relevant ones and wrong ones) should have lower ranks. With this aim in mind we choosed to use the precision metric, which measures the fraction of relevant tags on a list: since it is exactly what we wanted to do we have implemented all the metrics as variations of the precision to measure the performance of our system:

**Precision at** $k$ This metric measures how many good results are in the top $k$ results from the ranked list of tags, reporting the fraction of relevant tags. If we think that a real system may keep the first $k$ tags it is important to measure which fraction of the first $k$ tags is expected to be right. We measured this precision at $k = 1, 3, 5, 10$ for each song and we then average the value between all songs to have an idea on the general performances.

**Precision at number of total tags** This metric measure the precision (as defined before) at a precise point. Since we tested our system on a well tagged dataset we know how many real tags every song have. This means we can measure whether the system is able to guess these tags and put them on top of the ranked list. To do this we count how many tags the song really have (say $R$), and we measure the precision at $R$, then we report the average value of this precision between every song.

**Precision at 100% recall** The recall measures the fraction of all relevant tags retrieved by the ranked list. Since our ranked list includes all the tags the recall is always 100% at the end of the list, so the interesting statistic is the recall at a given position or the precision at a given recall value. Here we measure the precision when the recall reach 100%, that is, when the ranked list of tags includes all the tags from the ground truth.

**Mean Average Precision** The MAP is calculated by averaging the precision at each point of the ranked list of tag. The mean value of the average precision of each song is the MAP of the system. This is a measure of the quality of the whole ranked list.

### 3.1.3 Tests

Since the role of this model is to tag a new song we can test it using the ground truth (the CAL500 dataset) in a leave-one-out fashion. The model can be "trained"[2] with all songs minus one, and tested with the removed one. What we have done is calculating the timbric similarity for each song in the dataset, and, in turns, we simulated the querying of each song against the rest of the dataset. We have also calculated in advance the rhythm similarity between all songs.

The tagging procedure helps us, since it ignores the tag contribution from the query song as it assumes that it does not have tags on it. This is essential since we know in advance the tags from the query as they come from the ground truth.

Because we have done the timbric similarity calculation ahead of time, we do not need to "waste time" calculating it in the initialization phase, and we can speed up the test phase.

After we have queried the model we can test the results of the tagging by comparing the ranked tags against the tags from the ground truth using the metrics described in section 3.1.2.

### 3.1.4 Parameters Tuning

The tagging model as proposed in this thesis has some parameters which have to be tuned. For the timbric similarity we rely on the work by Miotto and Orio in [28], since they had good results in their model. They have defined the timbric similarity as an exponentiation KL divergence of the distribution of the MFCCs of the songs. The parameters of this similarity are the same as their work ($\gamma = 0.009$ and number of subsampled MFCCs).

Other parameters which have been kept as before are the one from the pruning of transitions probabilities (i.e. $\mathcal{R}(S_i)$). In this case we have kept the 10% of the total transitions.

On the modified Viterbi algorithm we use $\eta = 10$ as defined in the retrieval model, however we think that an investigation on this could lead to a better strategy for loop prevention.

We also tested some combinations of the values of $T$ (length of the random path) in relation to the number of effectively retrieved songs per path and the total number of iterations (see section 2.2.4). What we have seen is that short paths give better results, and the number of effectively retrieved songs should be as close as possible to

[2]We may not call it training, because we do not adjust any parameter on the basis of some data analysis

the length of the path: this could mean that the conservation of acoustic similarity is preferred over the number of retrieved results. We ended up choosing a path length of 4 with 3 retrieved song per iteration. This approach can produce iterations where as little as 1 song are retrieved, that is, there can be a loop of length 3 with the query and one song. Other combinations of path length and retrieved songs led us to worse results MAP-wise.

Regarding the total number of iterations of the Viterbi algorithm we have seen that, for our data, the best result were obtained with 9 iteration. Any number below that value has led to worse results. Iterations above 10 were not tested because we wanted to limit the total number of retrieved songs, and also because we got better results with 9 than with 10, as confirmed in figure 3.1 (tested with Cosine similarity).

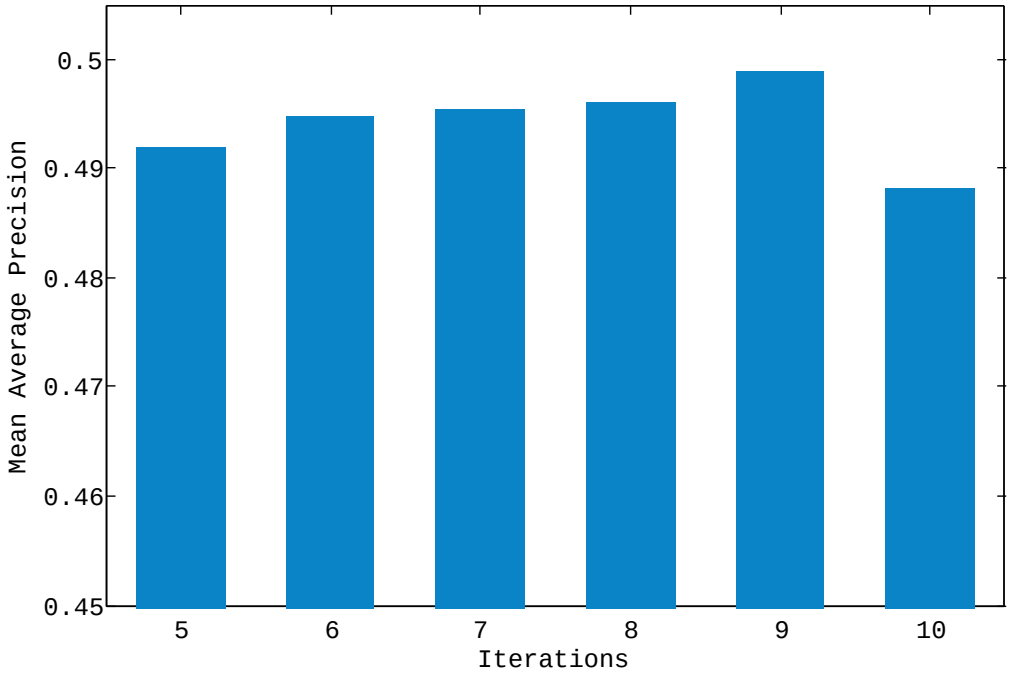



Figure 3.1: Mean Average Precision plot for different numbers of Viterbi iterations, using Cosine similarity (see section 3.1.4)

Of course the influence of these parameters should be further discussed and optimal values may change for different datasets or as the model grows integrating the tagged songs.

## 3.2 Results

The first question we face is which similarity measure best describes the difference between rhythm of the songs. We can measure this with the Mean Average Precision over the tagging of all tested songs. We also have to compare the results of each weighting strategy in order to find the best overall performance.

Table 3.1, and the corresponding plot in figure 3.2 show the summary of the results we obtained from our tests. We can see that Ruzicka similarity gives the best results among the similarity measures. We also see that linear weighting outperforms the path probability weighting with every similarity metric: this suggests that path distance from the query is more important than the path probability.

| | Path Prob. | Linear | Exponential | Hyperbolic |
|---|---|---|---|---|
| Cosine | 0.5036 | 0.5430 | 0.5418 | 0.5415 |
| Bray Curtis | 0.5096 | 0.5574 | **0.5558** | **0.5558** |
| Mod. chord | 0.5054 | 0.5535 | 0.5497 | 0.5497 |
| Ruzicka | 0.5108 | **0.5594** | **0.5558** | **0.5559** |
| Sim. ratio | 0.5016 | 0.5390 | 0.5400 | 0.5393 |
| KL | 0.4689 | 0.4802 | 0.4833 | 0.4822 |

Table 3.1: Mean Average Precision values for Rhythm Histograms similarity measures and weighting functions. Best results are highlighted in bold.

Since we said that a real system may keep only the top most relevant tags after the tagging process, it is interesting to see the performance of our system in terms of precision at 10. This measure tells us how well we can tag a novel song if we keep only the top 10 tags in the ranking list. We also wanted to see how well our system perform in respect to the baseline, that is, if our system does better than a (truly) random walk in the graph. To measure this we tried three strategies: first, we measured the influence of the rhythmic similarity by uniforming the transition probabilities (which are the timbric similarities) to $1/N$, where $N$ is the number of songs in the dataset. Then we measured the influence of the timbric similarity by uniforming the observation probabilities (which are the rhythmic similarities) to $1/N$. The last test simulated a completely random walk by uniforming both observation and transition probabilities, and this last test represents the true baseline.

Figure 3.3 shows our results using the best performing rhythmic similarity measure, the Ruzicka, and we can clearly see that our results are better than all baseline tests. The value of the precision tells us that if we pick a random song and we keep

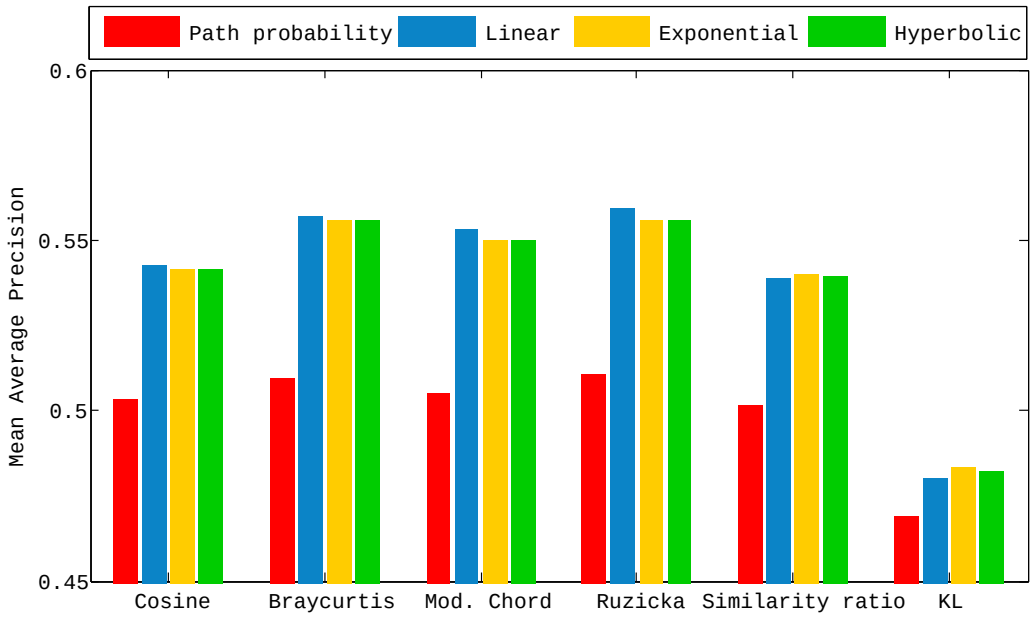



Figure 3.2: Mean Average Precision plot for table 3.1: results for different rhythm similarity measures and weighting functions.

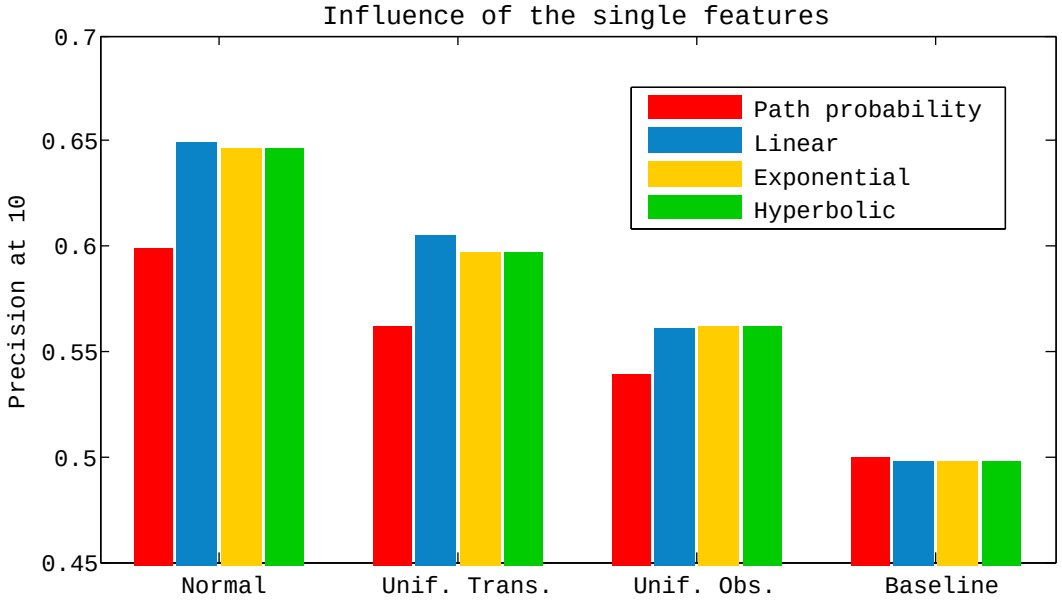



Figure 3.3: Comparison of mean precision at 10 calculated with Ruzicka rhythm similarity and different baseline test cases.

only the top 10 proposed tags, we can expect 6 of them to be correct on average, which is a good result for the first implementation of a novel approach. We can also see that uniforming transition probabilities leads to better results than uniforming observation probabilities: this does not necessarily mean that rhythmic features are better for discriminating songs. This could be the effect of using rhythmic similarity as the observation probabilities, and this should be further investigated by swapping the role of timbre and rhythm in the model (we will see this on the conclusions). Another interesting fact is that even using one feature (i.e. only rhythm or only timbre) we obtain better results than the baseline's completely random walk.

One particular test that is usually done for retrieval systems is the precision at 1: this measures the ability of a system to propose a meaningful document at the top of the ranked list of retrieved document. Results for this test, computed using Ruzicka similarity measure, are exposed in table 3.2. We can see that results are all very good, with the worse preformance by path probability weighting function. However, we can see that the baseline outperforms the normal procedure, meaning we can seek for errors in this direction.

| | Normal | Unif. Trans. | Unif. Obs. | Baseline |
|---|---|---|---|---|
| Path prob. | 0.8267 | 0.7888 | 0.7072 | **0.8785** |
| Linear | **0.8685** | **0.8705** | 0.8207 | **0.8845** |
| Exponential | 0.8625 | 0.8645 | 0.8207 | **0.8845** |
| Hyperbolic | 0.8625 | 0.8645 | 0.8207 | **0.8845** |

Table 3.2: Precision at 1 values for different weighting functions. Best results are highlighted in bold.

Another way to measure if the system is performing well is to measure the precision at the number of tags from the ground truth (see section 3.1.2). Table 3.3 shows a summary of the performances for different rhythmic similarities and weighting function. We can clearly see that Ruzicka and Bray Curtis are the top performing similarities. The overall results are low but this was expected since we are measuring the precision on a large part of the ranked list.

An interesting measure is the precision value when recall reach 100%, that is, the precision value calculated when the ranked list includes all tags from the ground truth. We do not expect high values, in fact table 3.4 shows poor results, however it is interesting to see that again results calculated with Ruzicka similarity measure are the top performing ones together with the ones calculated with Bray Curtis similarity.

| | Path Prob. | Linear | Exponential | Hyperbolic |
|---|---|---|---|---|
| Cosine | 0.4505 | 0.4995 | 0.4997 | 0.4985 |
| Bray Curtis | 0.4533 | **0.5141** | **0.5114** | **0.5110** |
| Mod. chord | 0.4539 | 0.5093 | 0.5062 | 0.5058 |
| Ruzicka | 0.4549 | **0.5138** | **0.5103** | **0.5102** |
| Sim. ratio | 0.4531 | 0.4947 | 0.4956 | 0.4949 |
| KL | 0.4313 | 0.4449 | 0.4487 | 0.4465 |

Table 3.3: Mean precision at number of tags from ground truth; calculated for different Rhythm Histograms similarity measures and weighting functions. Best results are highlighted in bold.

| | Path Prob. | Linear | Exponential | Hyperbolic |
|---|---|---|---|---|
| Cosine | 0.2119 | 0.2277 | 0.2274 | 0.2273 |
| Bray Curtis | 0.2149 | **0.2327** | **0.2325** | **0.2324** |
| Mod. chord | 0.2119 | 0.2281 | 0.2275 | 0.2275 |
| Ruzicka | 0.2140 | **0.2316** | **0.2311** | **0.2311** |
| Sim. ratio | 0.2105 | 0.2249 | 0.2247 | 0.2244 |
| KL | 0.2070 | 0.2151 | 0.2150 | 0.2149 |

Table 3.4: Mean precision at 100% recall for Rhythm Histograms similarity measures and weighting functions. Best results are highlighted in bold.

# Conclusions and future work

Automatic music tagging is a challenging task in the music information retrieval world. In this thesis we have presented a novel approach for semantic music tagging on a graph-based framework. The work was based on the assumption that acoustically similar songs have similar tags, and thus we can infer the tag of a novel song by seeking for its acoustical neighbours. We have been inspired by Hidden Markov Models and we have developed a model which combines timbric and rhythmic features to tag new songs. Each state in the model represents a song, while transition probabilities are modeled as timbric similarity between songs and observation probability as rhythmic similarity with the query song. The model assumes that at any time there is a set of known songs with proper tags on them that serves as prior knowledge for tagging and acoustic similarity modeling. A modified Viterbi algorithm was developed and a criterion for repeated exploration of the graph was introduced to extract tags from similar songs. The output of the querying process is a set of ranked tags, where higher rank indicates higher tag relevance. We have found that using Viterbi for producing short "acoustic paths" for multiple times gives better results than extracting a single long path.

The model itself does not need much training, however different initial datasets may require some tuning on the Viterbi parameters (number of iterations to perform, path length for each iteration, tags weighting strategy) to reach optimal results.

We found out that the model performs better with a tags' weighting that is independent of the path probability, which is linked to the similarity metrics. This can be an advantage since this let us be more robust against similarity's poor discrimination performances, and we can use tags weighting strategy as a tuning parameter.

The framework described here was tested using the CAL500 dataset, a collection of western songs tagged by expert annotators. Experiments have shown good results in terms of Mean Average Precision of the ranked lists of tags. We have also measured the performance contribution of each feature separately: we have found that both features perform better than random guessing, while combining the features in the model produces better results than using single features. To the best of our knowledge, we

think that our results are to be considered respectable for an initial work on a new model. Performances are in line with other approaches at an early stage, so we can be quite satisfied. It should be noted that usually automatic music tagging systems are specialized in one kind of tags (e.g. only in genre or mood classification) while this approach aims to be as general as possibile. We take our initial performance as an encouragement for further developing of the system.

As the model exploits acoustic similarity to produce a list of similar song we can exploit this framework as a retrieval model. For example we could build a system for query-by-example retrieval, which could take the query song as input and produce a playlist of similar songs as output. The output playlist would have not only a high acoustic similarity with the query but also a high probability of being semantically linked to the query's tags.

The performances obtained on this novel approach let us be confident about the evolution and extension of the model. We believe that the enhancing of this model could lead to great improvements in music information retrieval. We also believe that artificial intelligence could take advantage of this framework too, as this model tries to infer meaningful meta-data (tags) from the raw data (audio clips) by exploiting multiple features: this could be an good step toward semantic classification of generic data.

# Future work

Future works can take several directions from this initial approach. First we can search for different similarity metrics for the features, expecially the timbric ones. We have trusted the earlier work by Miotto and Orio for the similarity measure for MFCCs but there could be other significant metrics available. Another interesting thing to inspect is the role of each feature: future works may test if exchanging the roles of timbric and rhythmic features can lead to better results. It may also be worth trying using other features as well: for example melodic features like n-grams or harmonic like chromas.

As of the modified Viterbi algorithm, we have noticed that the workarounds to avoid loops may not be effective under some circumstances. Expecially the $\eta$ parameter in equation 2.29 has been subject of discussion in our group. We believe that a different approach on this, for example a data-dependent variation of the transition probability, could lead to better loop avoidance.

Another possible improvement on looping avoidance may be researched in the way we insert the novel song in the graph for the purpose of tagging. In the current procedure we calculate the timbric similarity and we treat the query as a song in the col-

lection (we just avoid it when we infer tags): this way there could be a loop between the query and a single other song, potentially leading to a poor tagging performance. In this case the vulnerability against looping is caused by the timbric similarity of the known songs with the query. This could be avoided by just setting transition probabilities to zero on all the transitions to the query, thus permitting the paths to start with the query but never come back to it.

The current tagging procedure suggests a way to grow the model by adding the newly tagged song to the graph. This could be a good starting point for a research on how the newly inserted song influences the tagging of next songs. Since the tagging is not perfect we can expect that the performances will decrease as new songs are added in the graph. One possibile solution would be to avoid the adding of the newly tagged songs and keep tagging next audio clips using a single large well tagged dataset. However, a better approach would be trying to validate the tags of new songs: we could think on a music streaming service which tags unknown songs, leaving them on a separate collection. As the users listen to these songs the system may ask whether they agree or not with the proposed tags. This way the system can mark the songs as correctly tagged, and then add them to the tagging graph, thus refining the results of next novel songs.

*"Said the straight man to the late man*
*Where have you been?*
*I've been here and I've been there*
*And I've been in between."*
(King Crimson - I talk to the wind)

# Bibliography


[1] L.E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.

[2] T. Bertin-Mahieux, D. Eck, F. Maillet, and P. Lamere. Autotagger: A model for predicting social tags from acoustic features on large music databases. *Journal of New Music Research*, 37(2):115–135, 2008.

[3] Simon Bourguigne and Pablo Daniel Agüero. Mirex 2011: Audio tag classification using feature trimming and grid search for svm.

[4] J Roger Bray and John T Curtis. An ordination of the upland forest communities of southern wisconsin. *Ecological monographs*, 27(4):325–349, 1957.

[5] Sung-Hyuk Cha. Comprehensive survey on distance/similarity measures between probability density functions. *City*, 1(2):1, 2007.

[6] E. Coviello, L. Barrington, A.B. Chan, and G.R.G. Lanckriet. Automatic music tagging with time series models. In *11th International Society for Music Information Retrieval (ISMIR) Conference*, 2010.

[7] Franz de Leon and Kirk Martinez. Using timbre, rhythm and tempo models for audio music similarity estimation. 2012.

[8] E. Di Buccio, N. Montecchio, and N. Orio. Falcon: Fast lucene-based cover song identification. In *Proceedings of the international conference on Multimedia*, pages 1477–1480. ACM, 2010.

[9] 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.

[10] S. Downie and M. Nelson. Evaluation of a simple and effective music information retrieval method. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 73–80. ACM, 2000.

[11] D. Eck, P. Lamere, T. Bertin-Mahieux, and S. Green. Automatic generation of social tags for music recommendation. *Advances in neural information processing systems*, 20(20):1–8, 2007.

[12] Anita Elberse. Should you invest in the long tail? *Harvard business review*, 86(7/8):88, 2008.

[13] K. Ellis, E. Coviello, and G.R.G. Lanckriet. Semantic annotation and retrieval of music using a bag of systems representation. In *proc. ISMIR*, volume 1, page 7, 2011.

[14] H. Fastl. Fluctuation strength and temporal masking patterns of amplitude-modulated broadband noise. *Hearing Research*, 8(1):59–69, 1982.

[15] D. Goldberg, D. Nichols, B.M. Oki, and D. Terry. Using collaborative filtering to weave an information tapestry. *Communications of the ACM*, 35(12):61–70, 1992.

[16] Philippe Hamel. Multi-timescale pmscs for music audio classification.

[17] M. Hoffman, D. Blei, and P. Cook. Easy as cba: A simple probabilistic model for tagging music. In *Proc. of ISMIR*, pages 369–374, 2009.

[18] Jeffrey R Johansen, Adchara Javakul, and Samuel R Rushforth. Effects of burning on the algal communities of a high desert soil near wallsburg, utah. *Journal of Range Management*, pages 598–600, 1982.

[19] P. Knees, T. Pohle, M. Schedl, D. Schnitzer, K. Seyerlehner, and G. Widmer. Augmenting text-based music retrieval with audio similarity. *Proceedings of the International Society for Music Information Retrieval (ISMIR'09)*, pages 579–584, 2009.

[20] T. Li, M. Ogihara, and Q. Li. A comparative study on content-based music genre classification. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 282–289. ACM, 2003.

[21] T. Lidy and A. Rauber. Evaluation of feature extractors and psycho-acoustic transformations for music genre classification. In *Proc. ISMIR*, pages 34–41, 2005.

[22] Shin-Cheol Lim, Karam Byun, Jong-Soel Lee, Sei-Jin Jang, and Moo Young Kim. Music genre/mood classification: Mirex 2012.

[23] B. Logan et al. Mel frequency cepstral coefficients for music modeling. In *International Symposium on Music Information Retrieval*, volume 28, page 5, 2000.

[24] M.I. Mandel and D.P.W. Ellis. Multiple-instance learning for music information retrieval. In *ISMIR 2008: Proceedings of the 9th International Conference of Music Information Retrieval*, pages 577–582. Drexel University, 2008.

[25] Michael I Mandel and Daniel PW Ellis. Song-level features and support vector machines for music classification. In *ISMIR 2005: 6th International Conference on Music Information Retrieval: Proceedings: Variation 2: Queen Mary, University of London & Goldsmiths College, University of London, 11-15 September, 2005*, pages 594–599. Queen Mary, University of London, 2005.

[26] R. Miotto and G. Lanckriet. A generative context model for semantic music annotation and retrieval. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(4):1096–1108, 2012.

[27] R. Miotto and N. Orio. A probabilistic approach to merge context and content information for music retrieval. *Proceedings of the International Society for Music Information Retrieval (ISMIR'10)*, pages 15–20, 2010.

[28] R. Miotto and N. Orio. A probabilistic model to combine tags and acoustic similarity for music retrieval. *ACM Transactions on Information Systems (TOIS)*, 30(2):8, 2012.

[29] N. Orio. *Music retrieval: A tutorial and review*, volume 1. Now Pub, 2006.

[30] F. Pachet and J.J. Aucouturier. Improving timbre similarity: How high is the sky? *Journal of negative results in speech and audio sciences*, 1(1):1–13, 2004.

[31] F. Pachet, D. Cazaly, et al. A taxonomy of musical genres. In *Proc. Content-Based Multimedia Information Access (RIAO)*, pages 1238–1245, 2000.

[32] E. Pampalk. Islands of music: Analysis, organization, and visualization of music archives. *Master's thesis, Vienna University of Technology, Vienna, Austria*, 2001.

[33] Renato Panda and Rui Pedro Paiva. Mirex 2012: Mood classification tasks submission. *Machine Learning*, 53(1-2):23–69, 2003.

[34] L. Rabiner and B.H. Juang. Fundamentals of speech recognition. 1993.

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

[36] A. Rauber, E. Pampalk, and D. Merkl. The som-enhanced jukebox: Organization and visualization of music collections based on perceptual models. *Journal of New Music Research*, 32(2):193–210, 2003.

[37] D. Reynolds. Gaussian mixture models. *Encyclopedia of Biometric Recognition*, pages 12–17, 2008.

[38] M Ruzicka. Anwendung mathematisch-statistischer methoden in der geobotanik (synthetische bearbeitung von aufnahmen), 1958. Cited in [18].

[39] M.R. Schroeder, B.S. Atal, and JL Hall. Optimizing digital speech coders by exploiting masking properties of the human ear. *The Journal of the Acoustical Society of America*, 66:1647, 1979.

[40] J. Shifrin, B. Pardo, C. Meek, and W. Birmingham. Hmm-based musical query retrieval. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 295–300. ACM, 2002.

[41] S.S. Stevens, J. Volkmann, and EB Newman. A scale for the measurement of the psychological magnitude pitch. *The Journal of the Acoustical Society of America*, 8(3):185–190, 1937.

[42] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to data mining*. Pearson Education India, 2007.

[43] 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.

[44] D. Turnbull, L. Barrington, D. Torres, and G. Lanckriet. Semantic annotation and retrieval of music and sound effects. *Audio, Speech, and Language Processing, IEEE Transactions on*, 16(2):467–476, 2008.

[45] Douglas Turnbull, Luke Barrington, David Torres, and Gert Lanckriet. Towards musical query-by-semantic-description using the cal500 data set. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 439–446. ACM, 2007.

[46] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *Speech and Audio Processing, IEEE transactions on*, 10(5):293–302, 2002.

[47] George Tzanetakis. Marsyas submissions to mirex 2012.

[48] Nuno Vasconcelos and Andrew Lippman. Learning mixture hierarchies. *Advances in Neural Information Processing Systems*, pages 606–612, 1999.

[49] Ming-Ju Wu and Jyh-Shing Roger Jang. Mirex 2012 submissions-combining acoustic and multi-level visual features for music genre classification.

[50] E. Zwicker, H. Fastl, and H. Frater. Psychoacoustics, facts and models, volume 22 of springer series of information sciences, 1999.