An Interactive Personal Audio Database for Musicians

Dawen Liang

Submitted in partial fulfillment of the requirements for the degree of
Master of Science in Music and Technology

School of Music
Carnegie Mellon University

May, 2012

Thesis Committee:
Roger B. Dannenberg, Chair
Richard M. Stern
Richard Randall

Abstract

As the high-capacity data storage devices (e.g. portable hard drives, and USB flash drives) become available to everyone, musicians are able to record all of their rehearsals and save them digitally. Given large amount of unlabeled and unorganized rehearsal recordings, manually organizing them can be a huge task. Therefore, managing music audio databases for practicing musicians automatically is a new and interesting challenge.

This thesis describes a systematic investigation to provide useful capabilities to musicians both in rehearsal and when practicing alone. The goal is to allow musicians to automatically record, organize, and retrieve rehearsal (and other) audio to facilitate review and practice (for example, playing along with difficult passages). In order to accomplish this task, three separate problems should be solved:

- Given the original recordings, extract the music segments and get rid of the non-music parts. A novel music classification system based on Eigenmusic and AdaBoost classifier to separate rehearsal recordings into segments is proposed.
- Given the music segments from the previous step, the system should be able to group the segments which belong to the same composition together. An unsupervised clustering and alignment process to organize segments is introduced.
- Finally, an interactive user interface is required for musicians to easily access and compare the previous rehearsals. The thesis provides a digital music display interface that provides both graphical input and output in terms of conventional music notation.

The thesis will address each of these 3 problems and provide an implementation that works in practice. Some future work is also described.

Contents

1 Introduction

Music Information Retrieval (MIR) promises new capabilities and new applications in the domain of music. Consider the following scenario: you are a prestigious violinist and your assistant records all of your rehearsals and saves them into a huge ***5 TB*** portable hard drive. Now you want to listen to all the rehearsals where you played the Tchaikovsky's *D Major Violin Concerto* with the orchestras to seek for further improvement. However, what you are facing (assume the recordings are stereo with CD quality) is about 220 days of audio. To make it worse, you did not play the entire concerto in some of the rehearsals, and you still want to find these segments of the whole piece. Organizing it by yourself will take forever. Therefore you would really like the computer to do this job for you automatically. This problem is exactly what this thesis is going to solve.

To define the specification more formally, we need to build a system that processes a personal music database composed of rehearsal recordings. Music is captured by continuously recording a series of rehearsals, where the music is often played in fragments and may be played by different subsets of the full ensemble. These recordings can become a valuable resource for musicians, but accessing and organizing recordings by hand is time consuming (as shown in the example above).

To make rehearsal recordings more useful, there are three main processing tasks that can be automated (See Figure 1). The first is to separate the sound into music and non-music segments. The music segments will consist of many repetitions of the same material. Many if not most of the segments will be fragments of an entire composition. We want to organize the segments, clustering them by composition, and aligning them to one another (and possibly to other recordings of the music). Finally, we want to coordinate the clustered and aligned music with an interface to allow convenient access. The thesis will present a display for music notation that provides a two-way interface to the underlying music database. The display can be used as *input* to select music (show me the recordings made at this location in the score) and also as *output* to show the current location in the score while music is played.

We see these capabilities as the foundation for an integrated system in which musi-

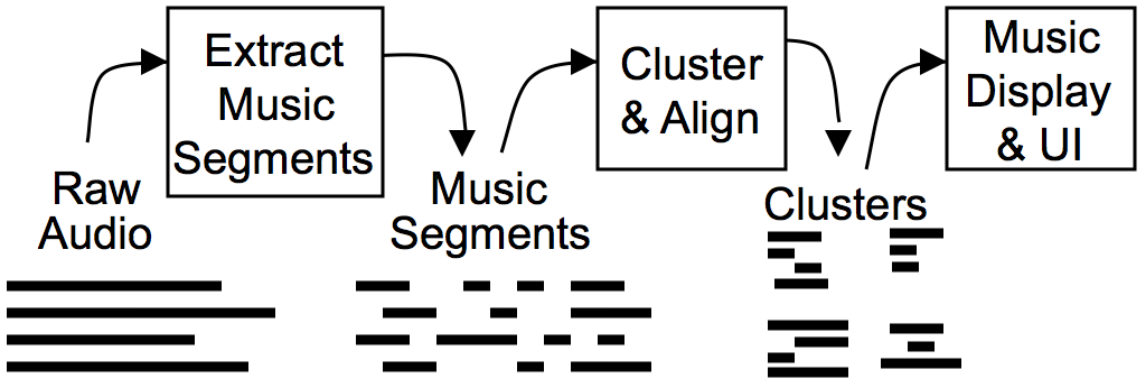

Figure 1: System diagram for a musician's personal audio database. Rehearsal recordings are automatically processed for simple search, analysis, and playback using a music notation-based user interface.

cians can practice and compare their intonation, tempo, and phrasing to existing recordings or to rehearsal data from others. By performing alignment in real time, the display could also turn pages automatically.

The organization of the rest of the thesis is as follows: In Section 2, related work will be briefly presented. Section 3 provides a system overview for the implementation. Section 4 and 5 describe how to organize the segments (namely, segment the raw audio into music and non-music and cluster the segments from the same composition together). Section 6 describes a two-way interface from music notation to the audio. The experiments and results analysis are in Section 7. Many of the possible applications for the system are presented in Section 8. The main contributions of the thesis are presented in Section 9. Finally, the thesis concludes in Section 10 and future work is presented.

2 Related Work

The problem proposed by the thesis is new to the fields of music understanding, as there is little previous work on rehearsal recordings segmentation and clustering. But much work has been done on other scenarios.

For different scenarios, people extract different features. A lot of work dating back to early days was focused on segmenting music and speech from broadcast streams, or

videos. Saunders (1996) described a technique using zero-crossing rate (ZCR) and the dominant frequency principle which is similar to spectral centroid, successful at discriminating speech from music on broadcast FM radio. Lu and Hankinson (1998) turned to work on automatically classifying audio into categories such as music, noise and speech. They used silence ratio, spectral centroid, harmonicity and pitch as features, a rule-based classification algorithm, and found out that silence ratio worked best among the three features in the task of video classification. There was also work on background music detection (Izumitani et al., 2008), that is, detecting music used in background of main sounds, such as TV programs. Besides some classical features as mentioned above, Mel-frequency cepstrum coefficients (MFCCs), which are widely used in speech recognition, were tried and it turned out MFCCs help improve the performance.

Later, instead of exploring new features (which is hard), researchers turned to the broad world of machine learning, specifically designing and adapting algorithms for the tasks they were facing. One of the most widely used statistical methods is Bayesian Information Criteria (BIC). Strictly speaking, BIC is not a statistical classifier, but a model selection technique. Chen and Gopalakrishnan (1998) first used BIC in the task of acoustic change detection and they reformulated this task as equivalent to model selection. Then Zhou and Hansen (2000) adapted the idea of BIC in Chen and Gopalakrishnan (1998) for unsupervised audio stream segmentation and clustering. The basic idea of this paper is that given a sequence of audio frames, it is assumed that each point before a certain position is drawn from a multivariate Gaussian which represents music, then the points after that position are drawn from another multivariate Gaussian which represents non-music. To find this boundary, BIC will treat this as a model selection problem, searching for the optimal position.

Another widely used model is the Hidden Markov Model (HMM). Ajmera et al. (2003) proposed a 2-state HMM (music and non-music) for segmentation, which makes very intuitive sense given the mathematical property underlying HMM. Rhodes et al. (2006) used HMM for music structure segmentation, and showed that this method can produce accurately labelled segmentations for popular music.

Other approaches include neural networks (Bugatti et al., 2002), Bayesian networks

(Pikrakis et al., 2008) and so on. In general, a lot of past work shows that a good combination of feature and model will produce the best results. However, the particular problem of variations in the sound source seems to be largely ignored. In reality, sound is not standardized in volume or bandwidth and may even contain different kinds of noise. In these cases, more robust features and methods are needed. In this thesis, Eigenmusic, similar to the famous Eigenface approach (Turk and Pentland, 1991), is used to capture and refine information from the frequency domain, and AdaBoost (Freund and Schapire, 1995) is used as a powerful non-linear classifier. Based on the probabilistic interpretation of AdaBoost, HMM is adopted to further smooth the results. Section 4 will talk about this in detail.

There is no prior work on systems for organizing large scale collections of rehearsal recordings. However, similar work in other fields is not difficult to find. The *Informedia Digital Video Library project* (Wactlar et al., 1996) creates large online digital video libraries featuring full-content and knowledge-based search and retrieval. In order to accomplish this task, both audio and visual features are utilized.

3 System Overview

The thesis provides a systematic implementation for the proposed solution in Figure 1. An overview of the whole system is introduced first in this section. It will serve as the simplified description for all the components talked about in the following sections, where more detailed implementation issues will be presented.

3.1 System Workflow

Defining a concise system architecture is the most important step for implementing a working system. Figure 2 illustrates the detailed architecture based on the system diagram shown in Figure 1. It dynamically describes the flow of the data from the very beginning, as the raw rehearsal recordings, to the last component, interactive music display. The three large boxes represent the three components (segmentation, clustering, and

display) respectively.

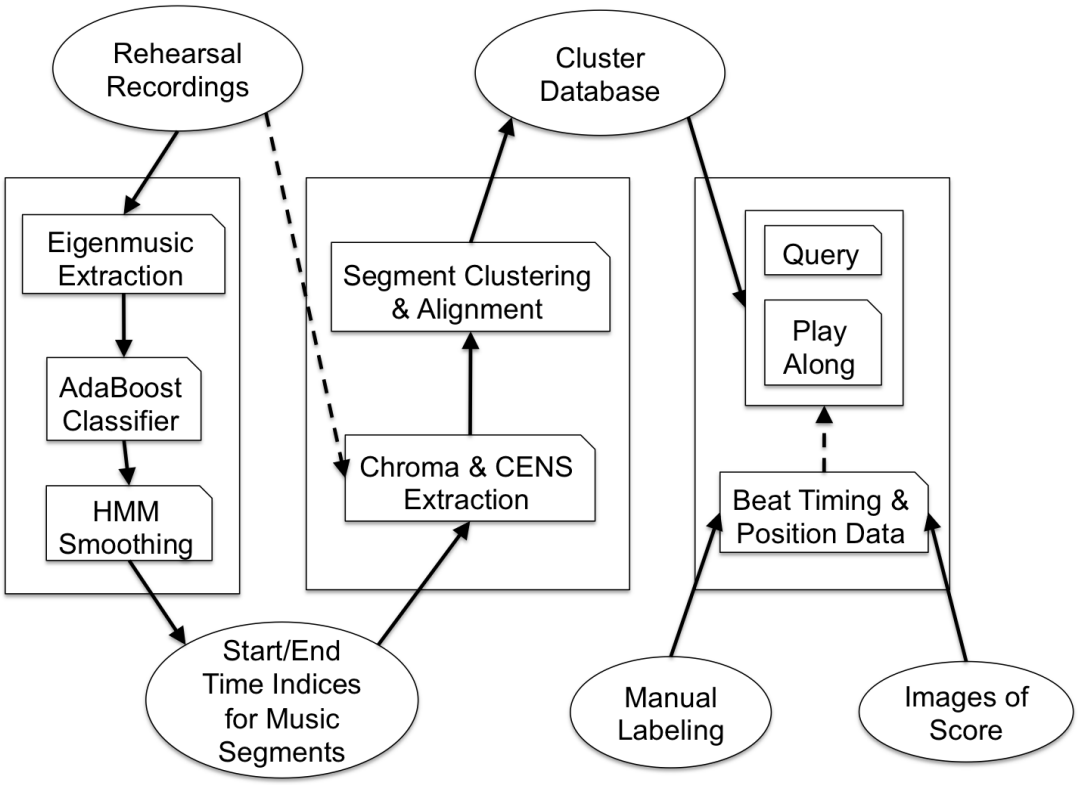

Figure 2: The detailed structure for the whole implemented system. Boxes represent processes (e.g. feature extraction, data acquisition), and ovals represent the data source, some of which are the output from the previous step.

In this picture, boxes represent processes (e.g. large boxes as functional components, and small boxes as data acquisition, feature extraction and processing). Ovals represent the data sources, some of which are the output from the previous step, for example, "*Start/End Time Indices for Music Segments*" oval from the segmentation is the input to the clustering component.

3.2 Components Description

Segmentation First of all, the rehearsal recordings are presented to the system as in the top left of Figure 2. For the segmentation component, Eigenmusic features (Section 4.1) are extracted first for every frame, then an AdaBoost classifier (Section 4.2) is used to predict if the frame is music or non-music. Finally, HMM is used to further smooth the results above the frame level (Section 4.3).

Here the term *segment* is used to represent a series of frames that are all determined by the system as music and the adjacent frames are determined by the system as non-music after the processing of the AdaBoost classifier and HMM smoothing. The length of one segment should be greater or equal to 30 seconds. Note that the results from this part are not actual audio files. Instead, only the start and end time indices of music segments are saved, so that a significant amount of space is saved. For future access, the original file can be read beginning with the sample corresponding to the start time instead of the beginning of the audio file.

In this component, considering that the length of the rehearsal recordings can be arbitrary, the spectrum of the recording audio, which is used to compute the Eigenmusic features, may be too big to fit into the memory. Therefore, the processing here is designed to be stream-based. Every time, only one frame of the recordings are read in to compute the feature, then AdaBoost makes prediction for this particular frame. This design makes sure the running time for segmentation grows almost linearly with the length of the recordings. As for the HMM smoothing step, only one number for each frame is stored, thus the memory issue is no longer a concern[1]. More details can be found in Section 4.4.

Clustering For the clustering component, there are two main tasks: cluster the segments according to the composition (Section 5.2), then align the segments within a cluster together (Section 5.3). Two related features are extracted from the rehearsal recordings (as the dashed line to the left shown): CENS (**C**hroma **E**nergy distribution **N**ormalized

[1]According to the long frame length used in Section 4, for 5 hours rehearsal recordings, only 14400 float numbers are stored, which can easily fit into the memory for a modern computer.

Statistics) for rough clustering and chroma vectors for accurate alignment (Section 5.1).

The term *cluster* is used here to represent a set of *segments* that (the system thinks) belong to the same composition. There is no 100% accurate learning/clustering algorithm, thus the system can make mistakes by either misclassifying non-music as music, or grouping the segments from different compositions. To address this problem, a manual correction mechanism is adopted in Section 5.4.

The output for this part is stored as the database, where for each segment, the necessary information for future access is listed. Once the database is built, every time the system starts, the information will be loaded into memory. The information is about 100 bytes per segment (more than 90% are used to store the name of the original recordings) so it is reasonable to keep it all in main memory.

The nature of the clustering algorithm (Algorithm 1) indicates that even with a huge amount of recordings already in the database, adding new ones is still efficient (linearly grows with the number of existing clusters, see the derivation in Section 7.3).

After clustering, the segments within a cluster are still isolated without an explicit connection. To make the display functionality as described below easier, these segments in each cluster are aligned, so that the time information for one segment can be easily converted to the time information for another segment within the same cluster. More details can be found in Section 5.4.

Display The database constructed in the previous step enables the display interface to access the recordings in a bi-directional (both as input and output) fashion. By manually labeling the score and audio, the mapping between score and audio can be established (as the "*Beat Timing & Position Data*" box indicates), as the foundation of the interactive interface. For each cluster, only one segment is necessary to be labeled, as the alignment step in the process above enables the time conversion between different segments within one cluster. Two functions are illustrated in Figure 2:

- *Query* (display as input): The query can be done in two ways:

    1. Given a score where the mappings with recordings have already been established as well as a certain position on the score, the available *segments* which

cover that position can be found.

2. Given a cluster (composition) from a recording audio file, the other recordings/segments within the same cluster can be found.

- *Play Along* (display as output): One of the most useful features of the system is that a player can play along with the rehearsal recordings for a better practice experience. Consider a flute player practicing a duet. It would be very helpful to practice with a recording of the other voice. As long as the mapping between the score and audio is properly established for the recordings being played, the location of the beat/measure can be indicated in the score in real-time. Furthermore, automatic page turning can be achieved based on the score location and page information.

The three components interactively communicate with each other, and each of them will be introduced in detail in the following sections, both in terms of the theoretical foundation and the system implementation.

4 AdaBoost & HMM-based Segmentation

This section and the next section include a description of Xia et al. (2011). In this section, a solution to the segmentation problem is presented. The input of this part of the system is simply the set of raw audio recordings, and the output is a set of music segments. Eigenmusic is first introduced as a robust feature. For music/non-music classification, AdaBoost is used to give frame-level results, from which HMM is used to further smooth them.

4.1 Eigenmusic Feature Extraction

The concept of Eigenmusic is derived from the well-known representation of images in terms of Eigenfaces (Turk and Pentland, 1991). The process of generating Eigenmusic can be performed in both the time and frequency domains, and in either case, simply refers to the result of the application of Principal Component Analysis (PCA) to the audio data,

though it is found that Eigenmusic in the frequency domain (Beyerbach and Nawab, 1991) produces better results.

One could consider Eigenmusic as the eigenvectors of an empirical covariance matrix associated with an array of music data. In the frequency domain, the array of music data is structured as a spectrogram matrix, with columns as the time frames and rows as a number of Fast Fourier Transform (FFT) bins. The array contains the spectral information of the audio in those time intervals. Expressing each frame of music data in terms of Eigenmusic can be considered as projecting the data point in the original high-dimensional space (in this case, the number of dimensions is the number of the FFT bins, typically hundreds or thousands) to a low-dimensional Eigenmusic space, while still keeping the statistically important information. On the other hand, when expressing the frames of non-music data in terms of Eigenmusic, the coefficients for the projected data point are generally expected to be outlying based on the fundamentally different characteristics of music and non-music.

In this thesis, the recordings are first resampled to 16000 *Hz*. Then about 2.5 hours of pure music in the training data collection is used to extract the Eigenmusic in the frequency domain:

- Let $\mathbf{X} = [\mathbf{x_1}, \mathbf{x_2}, \cdots, \mathbf{x_N}]$ be a spectrogram matrix. Each $\mathbf{x_i}$ represents the averaged magnitude spectra among 5 FFT frames, each of which corresponds to 0.25 seconds with no overlap. Thus each $\mathbf{x_i}$ represents 1.25 seconds of audio.

- Compute the corresponding empirical covariance matrix $\mathbf{C_x} = \overline{\mathbf{X}} \cdot \overline{\mathbf{X}}^{\mathbf{T}}$, where $\overline{\mathbf{X}} = \mathbf{X} - \mathbf{E}[\mathbf{X}]$ has 0 mean at each row.

- Eigenmusic $\mathbf{V} = [\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_M}]$ are the eigenvectors of the empirical covariance matrix $\mathbf{C_x}$ where $\mathbf{M}$ is the number of FFT bins. In practice, only the first 10 eigenvectors are retained corresponding to 10 largest eigenvalues, as the remaining eigenvalues are too small to be taken into account. Therefore, Eigenmusic $\mathbb{V} = [\mathbf{v_1}, \mathbf{v_2}, \cdots, \mathbf{v_{10}}]$.

Given a new magnitude spectrum column vector $\mathbf{x_{new}}$, its Eigenmusic coefficients can be represented by $\mathbb{V}^{\mathbf{T}}\mathbf{x_{new}}$ which is a 10-dimensional vector.

4.2 AdaBoost

AdaBoost (Freund and Schapire, 1995), short for Adaptive Boosting, is a meta-learning-algorithm. Strictly speaking, AdaBoost itself is not a classifier. Instead, it consists of many simple classifiers, and the prediction from AdaBoost is a weighted linear combination of the results of the classifiers. It is adaptive in the sense that subsequent classifiers built are tweaked in favor of those instances misclassified by previous classifiers[2]. Each simple classifier is individually considered as a *weak classifier*, in this case, a classifier on the Eigenmusic coefficients $\mathrm{h_t}(\mathbb{V}^{\mathbf{T}}\mathbf{x})$. The combined final classifier is considered to be a *strong classifier*. In the training step, each weak classifier focuses on instances with the higher weights, where the previous classifier failed. Then it will obtain a weight α_t for this classifier, and update the weight of individual training data based on the performance, to make misclassified instances gain more weights. In the prediction step, the strong classifier is taken to be the sign of the weighted sum of weak classifier predictions:

$$\mathrm{H}(\mathbf{x}) = \mathrm{sign}(\sum_t \alpha_t \mathrm{h_t}(\mathbb{V}^{\mathbf{T}}\mathbf{x})) \tag{1}$$

By training a sequence of linear classifiers $\mathrm{h_t}$, each one of which is simply a decision tree with depth 1, constructed by choosing an individual Eigenmusic dimension out of 10 and a root threshold that minimizes the weighted error, AdaBoost is able to accomplish a non-linear decision surface in the 10-dimensional Eigenmusic space. More importantly, the time complexity for training an AdaBoost grows linearly with the number of training data, which exceeds some other non-linear classifiers, e.g. Support Vector Machine, and naïve implementation of k-Nearest Neighbor, in terms of scaling. The experimental result for AdaBoost will be presented in Section 7.1.1.

4.3 HMM Smoothing for Segmentation

The AdaBoost prediction of music/non-music is made at the frame level. However, there is some prior knowledge about the recordings that may help in the high-level decision

[2] http://en.wikipedia.org/wiki/AdaBoost

making procedure: Most of the segments should be music, and the length of the segments should be much larger than that of frames, which means transitions from music to non-music and vice versa are uncommon. In order to take such information into account, in this section, a probabilistic interpretation of AdaBoost is discussed first. Then, based on this interpretation, a Hidden Markov Model (HMM) is used to smooth the decision boundary.

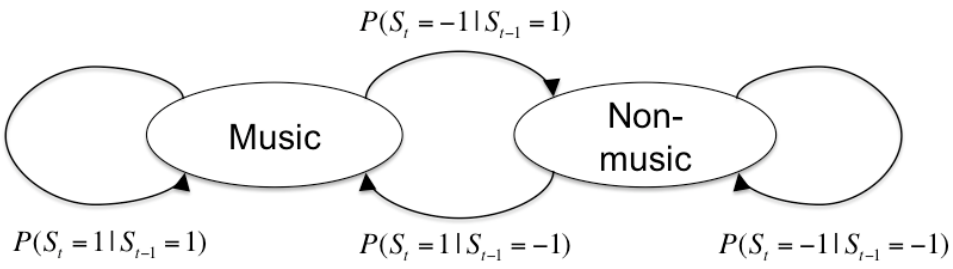

Figure 3: A two-state HMM (music and non-music) with all the transition probabilities. The emission probabilities are omitted, as they are neither discrete nor Gaussian, which makes visualization hard.

4.3.1 Probabilistic Interpretation

By taking into account the fact that state changes between music and non-music do not occur rapidly, the rehearsal recordings can be modeled as a two-state (music and non-music) HMM as shown in Figure 3. Formally, given an observation vector $\mathbf{x}$, let $\mathcal{S}_t = y \in \mathcal{Y} = \{-1, 1\}$ represent its true label. Here, -1 stands for non-music and 1 stands for music. And let $\mathrm{W}(\mathbf{x})$ represent the weighted sum of weak classifiers:

$$\mathrm{W}(\mathbf{x}) = \sum_t \alpha_t \mathrm{h_t}(\mathbb{V}^{\mathbf{T}}\mathbf{x}) \tag{2}$$

In Equation 1, the sign of $\mathrm{W}(\mathbf{x})$ is taken as the decision, but this approach could be modified to compute the *a posteriori* probability for $\mathcal{Y}$, given the weighted sum, which is denoted as the function $\mathcal{F}$:

$$
\begin{aligned}
P(y=1|\mathrm{W}(\mathbf{x})) =& \mathcal{F}(\mathrm{W}(\mathbf{x})) \\
P(y=-1|\mathrm{W}(\mathbf{x})) =& 1-\mathcal{F}(\mathrm{W}(\mathbf{x}))
\end{aligned}
\tag{3}
$$

According to the discussion in Friedman et al. (2000), $\mathcal{F}(\mathrm{W}(\mathbf{x}))$ could be a logistic function:

$$
\mathcal{F}(\mathrm{W}(\mathbf{x})) = \frac{1}{1+\exp(-2\cdot \mathrm{W}(\mathbf{x}))} \tag{4}
$$

To verify this hypothesis, in Figure 4, the small circles show $P(y=1|\mathrm{W}(\mathbf{x}))$ estimated from training data sorted into bins according to $\mathrm{W}(\mathbf{x})$. The logistic function is shown as the solid curve. It can be seen that the empirical data matches the theoretical probability quite well.

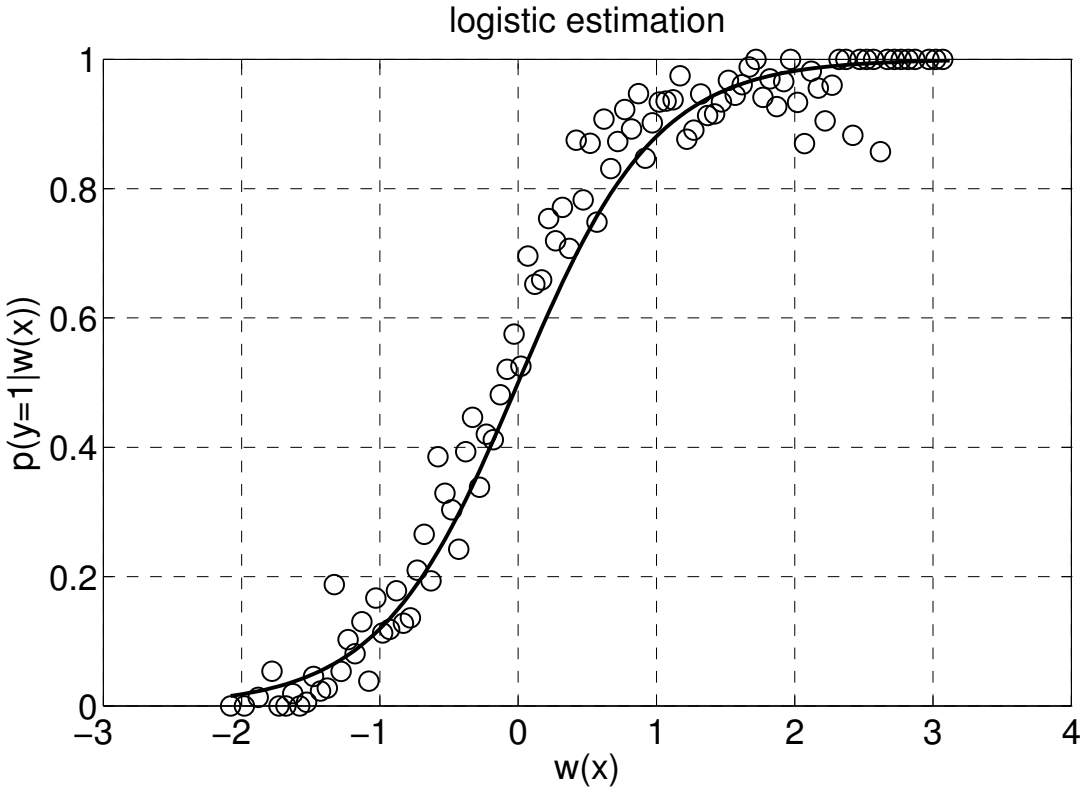

Figure 4: The logistic function estimation on training data.

It is worth noticing that the idea of linking AdaBoost with HMM is not new, but very little work has implemented it in the field of speech and music (Schwenk, 1999; Dimitrakakis and Bengio, 2004). To the authors' knowledge (Xia et al., 2011), this is the first

attempt of a probabilistic interpretation of AdaBoost when linked with HMM.

4.3.2 HMM Training and Smoothing

Even if pure AdaBoost could achieve very high accuracy, a very low error rate at the frame level cannot guarantee a satisfying segmentation result at the piece level. For example, suppose a relatively low 5% error rate is obtained at the frame level. If the segmentation rule is to separate the target audio at every non-music frame, a 10 minute long pure music piece would be cut into about $10(\text{minutes}) \times 60(\text{seconds/minute}) \ / \ 1.25(\text{seconds/frame}) \times 5\% + 1 = 25$ segments in this case. This is obviously undesirable. Therefore, based on typical characteristics of rehearsal audio data, it is assumed that: (1) music and non-music frames cannot alternate frequently, and (2) short duration music and non-music intervals are less likely than longer ones. By utilizing these assumptions in conjunction with the HMM, low (but possibly deleterious) frame-level error rates can be further reduced. The HMM observation corresponding to every frame $\mathbf{x}$ is a real number $\mathrm{W}(\mathbf{x})$, as in Equation 2, given by the AdaBoost classifier.

To smooth the results from AdaBoost using HMM, let $\mathcal{S} = [\mathcal{S}_0, \mathcal{S}_1, \cdots, \mathcal{S}_T]$ be the state sequence and let $\mathcal{O} = [\mathcal{O}_0, \mathcal{O}_1, \cdots, \mathcal{O}_T]$ be the observation sequence. To estimate the initial probabilities $P(\mathcal{S}_0)$ and transition probabilities $P(\mathcal{S}_{i+1}|\mathcal{S}_i)$, maximum likelihood estimation (MLE) is used by simply counting the fractions of each possible transitions $\{\text{music}, \text{non-music}\} \rightarrow \{\text{music}, \text{non-music}\}$ over the different previous states. For emission probabilities $P(\mathcal{O}_t|\mathcal{S}_t)$, according to Bayes rule:

$$P(\mathcal{O}_t|\mathcal{S}_t = 1) = \frac{P(\mathcal{S}_t = 1|\mathcal{O}_t)P(\mathcal{O}_t)}{P(\mathcal{S}_t = 1)} \tag{5}$$

Here the emission probability for music is derived first. $\mathcal{O}_t$ is given by $\mathrm{W}(\mathbf{x_t})$, and $P(\mathcal{O}_t)$ is a constant. According to Equation 3, $P(\mathcal{S}_t = 1|\mathcal{O}_t) = \mathcal{F}(\mathrm{W}(\mathbf{x_t}))$. Therefore, the emission probability for music can be further reduced to:

$$P(\mathcal{O}_t|\mathcal{S}_t = 1) = \mathcal{C} \ \frac{\mathcal{F}(\mathrm{W}(\mathbf{x_t}))}{P(\mathcal{S}_t = 1)} \tag{6}$$

$\mathcal{C}$ is the constant scalar multiplier. Similarly, the emission probability for non-music is:

$$P(\mathcal{O}_t|\mathcal{S}_t = -1) = \mathcal{C}\,\frac{1 - \mathcal{F}(\mathrm{W}(\mathbf{x_t}))}{P(\mathcal{S}_t = -1)} \tag{7}$$

A priori probabilities for both music and non-music are set to 0.5. Then the *Viterbi algorithm* (Viterbi, 1967) can be applied to efficiently find the most likely state sequence for a given observation sequence. The experimental result for HMM smoothing is presented in Section 7.1.2.

The smoothing is visualized in Figure 5. The sequences of transitions between music (white) and non-music (black) is illustrated. At each time point t_i, there are 2 choices for the state: music or non-music. There are 4 possible transitions from t_i to t_{i+1}: $\mathcal{S}_i^1 \rightarrow \mathcal{S}_{i+1}^1, \mathcal{S}_i^1 \rightarrow \mathcal{S}_{i+1}^{-1}, \mathcal{S}_i^{-1} \rightarrow \mathcal{S}_{i+1}^1$, and $\mathcal{S}_i^{-1} \rightarrow \mathcal{S}_{i+1}^{-1}$. For simplicity, here $\mathcal{S}_i^j = \{\mathcal{S}_i = j\}$. Each transition has a "cost value" assigned to it, which is computed from transition and emission probabilities. What the *Viterbi algorithm* does is simply to find the transition sequence that is most likely. As in Figure 5, dashed lines represent all the possible transitions (one out of four is covered by the solid line). The solid lines represent the path *Viterbi algorithm* chooses. The segments can then be found by checking the labels for each time point.

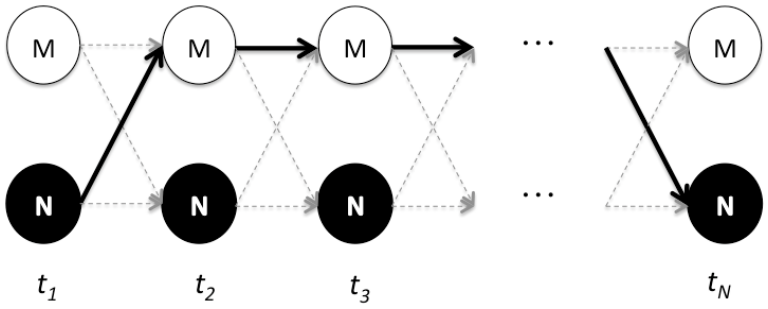

Figure 5: A visualization for HMM smoothing.

4.4 Segmentation Implementation

Many of the implementation issues for this part of the system have been mentioned in Section 3. More details will be provided here as they are important for an efficient working system.

4.4.1 Eigenmusic & AdaBoost Implementation

The class diagram for Eigenmusic & AdaBoost sub-component is shown in Figure 6. First of all, the notations in this diagram are explained.

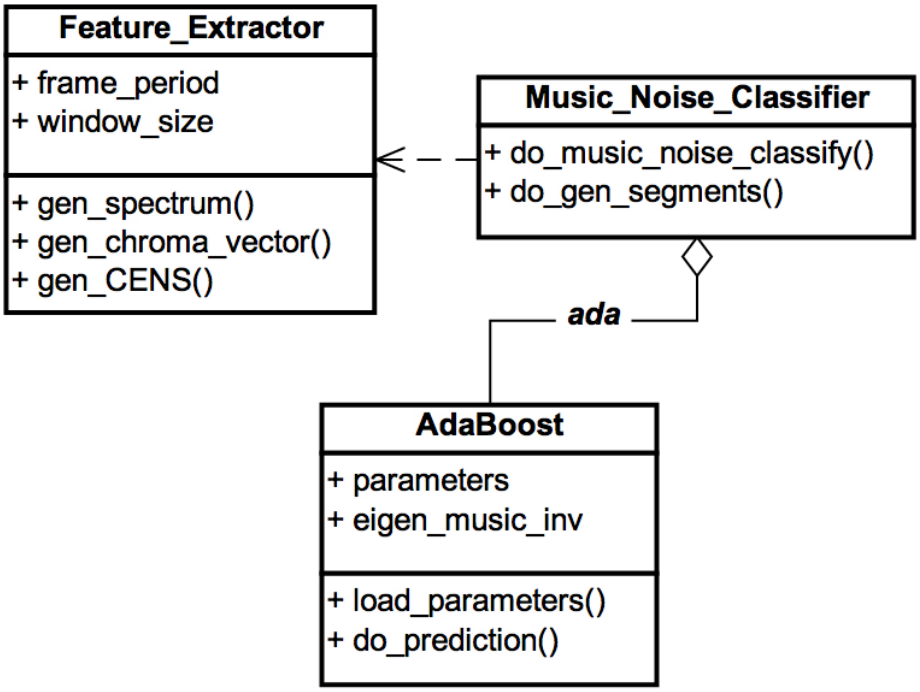

Figure 6: The class diagram for Eigenmusic & AdaBoost sub-component.

Each rectangular box represents a class, thus in this part, it has **Feature_Extractor**, **Music_Noise_Classifier**, and **AdaBoost** classes. For each class box, the middle part represents the class parameters (e.g. *frame_period* and *window_size* in class **Feature_Extractor**), while the bottom part represents the class methods (e.g. *gen_spectrum()*, *gen_chroma_vector()*, and *gen_CENS()* in class **Feature_Extractor**). For simplicity, only important class parameters and methods are shown here.

The dashed line between class boxes represents *dependency*. In the case of the system, class **Music_Noise_Classifier** will need to compute the Eigenmusic features for an audio file so that they can be predicted as music/non-music by AdaBoost, thus it depends on the class **Feature_Extractor** where method *get_spectrum()* computes the spectrum. The spectrum can then be further processed to obtain the Eigenmusic features.

The line with diamond at one end represents *aggregation*, where the diamond points to the base. In the case of the system, the segmentation component will first use the AdaBoost classifier to make a prediction on the frame level. Thus class **Music_Noise_Classifier** has the object *ada* for class **AdaBoost** so that it has access to all the public functionalities in class **AdaBoost**.

To segment the raw rehearsal recordings, a file name is passed as the parameter to the method *do_music_noise_classify()* in class **Music_Noise_Classifier**. Then the parameters (frame period and window size) are passed as constants. As mentioned earlier in Section 3, the processing for this part of the system is stream-based. One frame of the audio (0.25 seconds) is read in every time and an FFT is computed. The spectrum is stored in an array of spectra vectors. Then the spectrum is averaged over 5 frames (1.25 seconds). The class **AdaBoost** is more generic, thus method *do_prediction()* in class **AdaBoost** is not aware of the type of the input data, as long as it fits the parameters for the classifier. The predictions for all the long frames (with 1.25 seconds) are stored in a vector, which will be used in the HMM smoothing part described below.

4.4.2 HMM Smoothing Implementation

The class diagram for the HMM smoothing sub-component is shown in Figure 7. Following the notation introduced in Section 4.4.1, there is one object *hmms* of class **HMM_Smoother** in class **Music_Noise_Classifier** so that, similarly, it has the access to the smoothing method in class **HMM_Smoother**.

As in Section 4.4.1, the prediction results by the AdaBoost classifier are returned in a vector, which is further passed to the method *do_smooth()* here. The implementation is pretty straightforward, following the *Viterbi algorithm*: construct a table of "cost value" (Section 4.3.2) for 4 possible transitions at each time point and then find the path that costs the least. The naïve implementation is about 50 lines of C++ code.

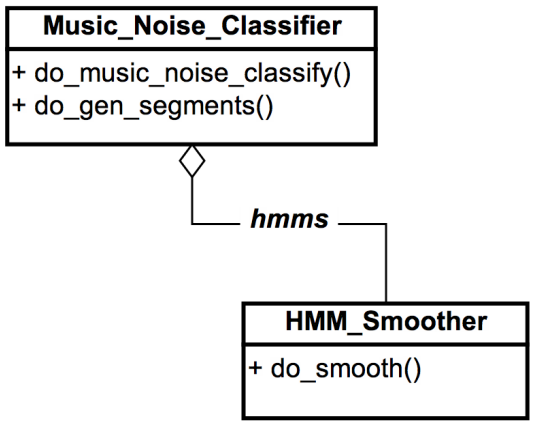

Figure 7: The class diagram for HMM smoothing sub-component.

5 Chroma-based Segments Clustering and Alignment

After the processes in Section 4, the raw recordings are segmented into short segments. The next step is to cluster these segments by composition. A feature based on the widely used chroma vectors will be presented first, then a novel unsupervised clustering algorithm is proposed. Finally, the algorithm to align segments is presented.

5.1 CENS Feature Extraction

Chroma vectors (Bartsch and Wakefield, 2001) have been widely used as a robust harmonic feature in all kinds of MIR tasks. The chroma vector represents the spectral energy distribution in each of the 12 pitch classes (C, C#, D,$\cdots$, A#, B), ignoring the specific octave, as shown in Figure 8. Such features strongly correlate to the harmonic progression of the audio.

Considering the objective that the system should be robust to external factors (e.g. audience cheering and applause for live performance recordings), the feature cannot be too sensitive to minor variations. Therefore, as suggested by Müller et al. (2005), 2 post-processing steps are taken to address the problem:

1. After obtaining the normalized energy distribution for the 12-dimensional chroma vector $\overrightarrow{v}$ (as virtualized in Figure 8) with about a 200 ms Hamming window and 50% overlap, $\overrightarrow{v}$ is quantized by creating 4 buckets between 0 and 1, and assigning

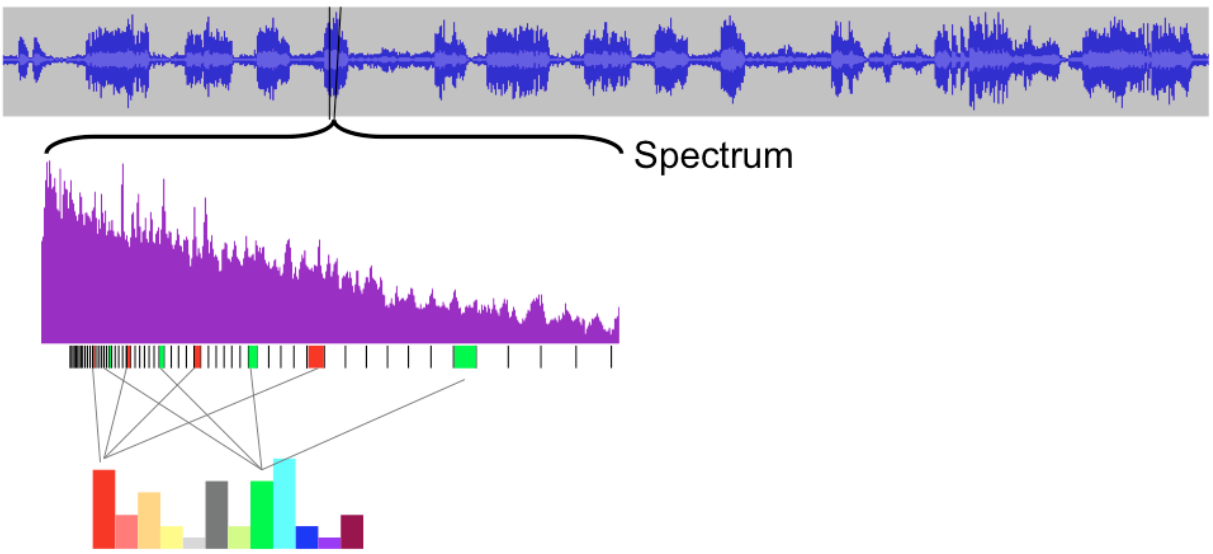

Figure 8: A visualization of the concept of chroma vectors.

values to each chroma component based on which bucket its value falls into. If the chroma component $v_i \geq 0.4$, the quantization value is assigned as 1. If $0.2 \leq v_i < 0.4$, the value is 0.75. If $0.1 \leq v_i < 0.2$, the value is 0.5. If $0.05 \leq v_i < 0.1$, the value is 0.25. 0 otherwise.

2. To calculate a longer-term summary, the sequence of quantized chroma vectors are convolved with a Hann window with length 41. The hop-size of 10 vectors (1 second) is used. After each convolution, downsample the result by a factor of 10 and normalize it to have norm 1 for each 12 dimensional vector. Thus, one vector per second is obtained, each of which spans roughly 4100 ms of audio.

These long-term feature vectors are described as *CENS* features (**C**hroma **E**nergy distribution **N**ormalized **S**tatistics) (Müller et al., 2005). The length of the long-term window and hop size in Step 2 can be changed to take global tempo differences into account. If using the default settings with windowing over 41 consecutive short-term vectors and summarizing with a 10-vector (1 second) hop-size, all the minor variations within 1 second will be ignored, which is desirable.

5.2 Segment Matching and Clustering

Segment matching happens in the following scenario. One segment (usually shorter) will be compared with another segment (usually longer) to see if they are close enough to be clustered together. The former one is refereed to as the "query segment", while the other is refereed to as the "template segment". Since the CENS itself is robust to minor variations, segment matching can be achieved by simply calculating the correlation between the query segment $\mathbf{Q} = [\mathbf{q_1}, \mathbf{q_2}, \cdots, \mathbf{q_M}]$ and the subsequence (of length M) of template segment $\mathbf{T} = [\mathbf{t_1}, \mathbf{t_2}, \cdots, \mathbf{t_N}]$ ($\mathrm{N} > \mathrm{M}$). This correlation can be considered as the distance between the 2 segments. Here, all the lower case $\mathbf{q_i}$ and $\mathbf{t_i}$ represent 12-dimensional CENS vectors. Thus, $\mathbf{Q}$ and $\mathbf{T}$ are both sequences of CENS vectors over time. As in Müller et al. (2005), the distance between query segment $\mathbf{Q}$ and the subsequence of template segment $\mathbf{T^{(i)}} = [\mathbf{t_i}, \mathbf{t_{i+1}}, \cdots, \mathbf{t_{i+M-1}}]$ is:

$$\mathrm{dist}(\mathbf{Q}, \mathbf{T^{(i)}}) = 1 - \frac{1}{M}\sum_{k=1}^{M}\langle \mathbf{q_k}, \mathbf{t_{i+k-1}} \rangle \tag{8}$$

Here $\langle \mathbf{q_k}, \mathbf{t_{i+k-1}} \rangle$ denotes the dot product between these 2 CENS vectors. All of the distances for $i = 1, 2, \cdots, N - M + 1$ together can be considered a distance function Δ between query segment $\mathbf{Q}$ and each subsequence of template segment $\mathbf{T^{(i)}}$. If the minimum distance is less than a preset threshold γ, then $\mathbf{Q}$ can be clustered with $\mathbf{T}$.

However, considering that the ultimate goal is to build a database, where segments are grouped by composition, one problem with this decision scheme is that, unlike a traditional song retrieval system which has a large reference database in advance, this system has no prior information about the rehearsal audio stream, but only a stream of potentially unordered and unlabeled audio that needs to be clustered, i.e. there is no predefined template $\mathbf{T}$. To solve this problem, the database is constructed dynamically following Algorithm 1. The inputs are all the music segments obtained from Section 4.

Here a critical assumption is made: the longer segment is most likely to be a whole piece or at least the longest segment for this distinct piece, so it is reasonable to let it represent a cluster. At every step of the iteration, a new segment $\mathbf{S}$ is taken out. It can either be part of an existing piece in the database (in which case it will be clustered with

Algorithm 1: buildDB

Take out the next segment **S**.

- If database $\mathcal{D}$ is empty, put **S** into $\mathcal{D}$ as the cluster centroid for cluster $\mathcal{C}_1$.
- Otherwise match **S** with the cluster centroid for every cluster $\mathcal{C}_i$, where $i \in \{1, 2, \cdots, N\}$, in $\mathcal{D}$ by calculating distance function Δ. Let $\mathcal{C}_{min}$ be the cluster where **S** and cluster centroid have the best match.
  - If the distance function Δ between cluster centroid of $\mathcal{C}_{min}$ and **S** has a minimum less than γ, cluster **S** with $\mathcal{C}_{min}$.
  - Otherwise make **S** the cluster centroid of a new cluster $\mathcal{C}_{N+1}$ in $\mathcal{D}$.
- Repeat the algorithm until all segments are clustered.

a matching segment) or it is a segment for a new piece which does not yet exist in the database (in which case it will be made a new cluster).

For each cluster, the longest segment is kept as the *cluster centroid*. The remaining segments are only compared with all the cluster centroids. If a segment **S** matches with cluster $\mathcal{C}_i$, and **S** is longer than the cluster centroid for cluster $\mathcal{C}_i$, then **S** could simply replace the original centroid for cluster $\mathcal{C}_i$. During the process of clustering, since the CENS for centroids are frequently used, the CENS for all centroids are saved in a local file, and will be loaded every time new segments are added to the database.

The system also needs to take into account that the possibility that tempo differences cause misalignment between sequences. Different versions of CENS features can be obtained (for example, from 10% slower to 10% faster) for the same segment to represent the possible tempos. This is achieved by adjusting the length of the long-term window and the hop size as mentioned in Section 5.1. During matching, the version of the segment with the lowest distance function minimum will be chosen.

5.2.1 Segment Length vs. Threshold Value

While time scaling compensates for global tempo differences, it does not account for local variation within segments. It is worth noticing that the query segment may not be used in full length, instead, only a subsequence can be used to calculate the distance. It

will be interesting to consider the length of the subsequence of the query segment that is used to correlate with the template segment in the database. Intuitively, longer segments will be more selective, reducing spurious matches. However, if the length is too large, e.g. two segments both longer than 5 minutes, sequence misalignments due to tempo variation will decrease the correlation and increase the distance. If longer subsequences lead to greater distance, one might compensate with larger threshold values (γ). However, larger γ values may not prove strict enough to filter out noise, leading to clustering errors. On the other hand, if only a small subsequence length is used, the threshold values γ should decrease accordingly. Two pairs of configurations will be compared in Section 7.2.1: longer subsequence length with larger γ and shorter subsequence length with smaller γ.

5.3 Segment Alignment

Hu et al. (2003) proposed a audio-to-MIDI alignment algorithm based on dynamic programming. The chroma vectors for audio and MIDI are computed first, then a similarity matrix is formed as shown in Figure 9, where the (i, j) element of the matrix is the Euclidean distance between the i-th chroma vector of the MIDI and the j-th chroma vector of the audio. Then a path starting from the bottom left, all the way to the upper right can be searched via dynamic programming.

The basic idea can be adopted to perform segment-to-segment alignment as well. The feature used here is chroma vector instead of CENS, mainly because CENS is too "rough" for accurate alignment. Recall from Section 5.2, as part of the by-product of the clustering process, a matching position between two segments can also be found. Therefore, starting from this position, the similar alignment algorithm can be performed.

The only exception is that, unlike in audio-to-MIDI alignemnt in Hu et al. (2003) where the start/end of the audio and MIDI are the same, here this matching position is the only available information. There is no information on where the alignment begins or ends in either the query or the template segment. Therefore, a two-directional search is performed until the start/end of the query segment is reached. This process is illustrated in

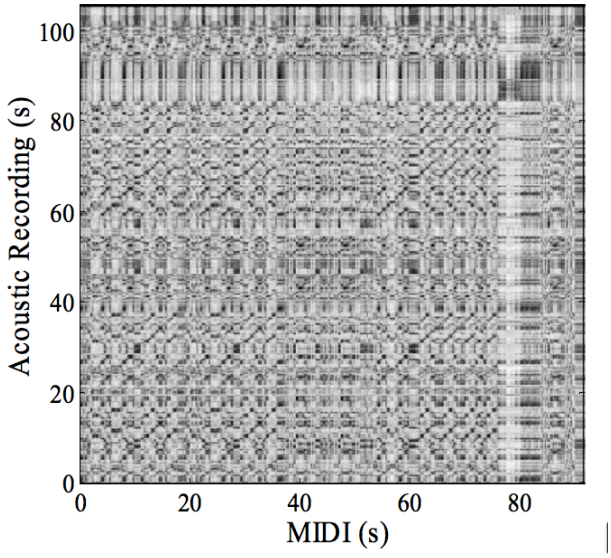

Figure 9: Similarity matrix for Beatles' "I Will" from Hu et al. (2003).

Figure 10. Note that this figure is only for the purpose of illustration. The template segment has 300 frames, while the query template has 100 frames. As mentioned in Section 5.2.1, not all of the segment is used (the length of the subsequence being used is determined by experiments in Section 7.2.1), thus the matching position is represented by the red dot, at about one fourth of the length of the query segment.

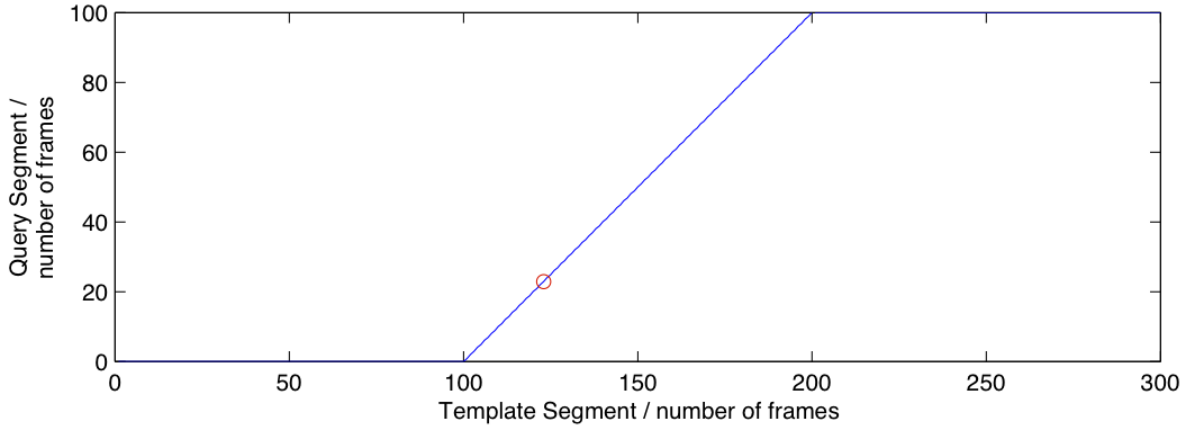

Figure 10: An illustration of template segment to query segment alignment. The matching position is shown as the red dot.

5.4 Segments Clustering and Alignment Implementation

The implementation issues for the clustering component are presented here. The implementation for segment alignment has not been finished yet, thus there is no description for alignment sub-component.

5.4.1 CENS Implementation

The implementation of CENS is straightforward, simply following the steps described in Müller et al. (2005). However, when computing the chroma vector, instead of filtering the audio with a filter bank consisting of elliptic filters to get the energy distribution for 12 pitch classes bins, an efficient implementation used in *Audacity*[3] is adopted to directly obtain the energy distribution by mapping and counting on the frequency domain.

To obtain the CENS from chroma vectors, first a Hann window is convolved with the chroma vector, then the results are downsampled to summarized the feature in a longer period. One implementation issue here is that the convolution of two vectors can be computed in $O(n \log n)$, by making use of the property of the convolution to first compute the FFT for both, then do the inverse FFT on their product, instead of $O(n^2)$ as the naïve implementation. However, the implementation used in the proposed system is in $O(n^2)$, as the length of Hann window is fairly small (41 for the default setting), so in fact the algorithm runs in $O(Cn)$, where C is the length of the Hann window. In practice, the computation is quite fast.

5.4.2 Segment Matching and Clustering Implementation

The class diagram for segment matching and clustering sub-component is shown in Figure 11. This is one of the most important parts of the system as it relates to all the other component classes.

Class **Segment_Cluster** is the main class to accomplish the matching and clustering task. From the class diagram, it can be observed that it is dependent on both class **Feature_Extractor** and **Music_Noise_Classifier**. The dependency with class **Feature_Extractor**

[3] http://audacity.sourceforge.net/

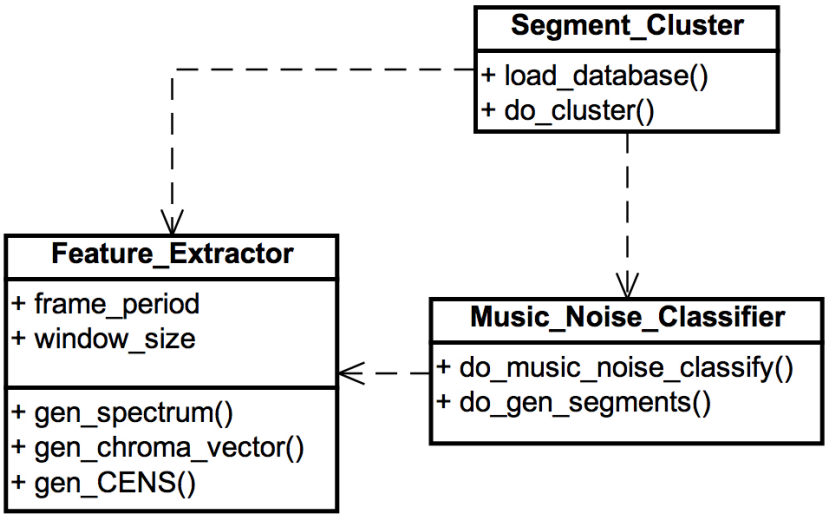

Figure 11: The class diagram for segment matching and clustering sub-component.

is intuitive as the clustering step first requires the computation of CENS features. It depends on class **Music_Noise_Classifier** because segment clustering needs the segments generated from the method *do_gen_segments()* in class **Music_Noise_Classifier**.

As mentioned in Section 3, the segments are not actually generated as audio files, instead only the start and end time indices are saved. Other information saved in a segment includes the original recording file name, the cluster ID (computed later) and the composition name. The composition name is optional, and needs to be provided by the user. If the composition name is provided for a cluster, in which all segments are considered to belong to the same composition, then search by composition name can be supported.

Following Algorithm 1, if the database has not been built, the first segment will be the centroid of the cluster, with cluster ID 0. The segment information will also be written to the text database file, with the format:

Recording filename	Start time index	End time index	Cluster ID

The start/end time indices represent the time in terms of the long frame (1.25 seconds). Every time a new segment is added to the databases, a new line of this format will be written to this text database file. On the other hand, if an existing segment is added again, as long as the system can detect it is a duplicated segment by comparing all the 5 fields in the table above, it will be ignored. Also, as mentioned in Section 5.2, the CENS for

cluster centroid is saved in a local file in a separate folder, with the file name containing the cluster ID. There are 2 reasons that the CENS for each cluster centroid has a separate file:

- Once the database is built, when the system starts again to add new segments into the database, the system can obtain the number of existing clusters by counting the number of files in this centroid CENS folder. At the same time, the system can load all of these centroid CENS files for comparison.
- As mentioned in Section 5.2, the cluster centroid may be changed to a longer segment. If all the centroid CENS are stored in one file, it is difficult to only replace part of the content programmatically, while it is only a few lines of code to replace a separate file.

Each remaining segment will be added by comparing it to the cluster centroid for each existing cluster. The distance determines if the segment should be added to one of the existing clusters or a new cluster needs to be created.

While the system starts adding new segments, the database will be loaded into the memory in the form of a *table* mapping from cluster ID to a list of segments within that cluster, instead of a huge *linear list* with all the segments as they are organized in the database file. This structure is much faster than the linear list in terms of search by cluster ID. This table will be empty if the database has not been built. Once a new segment has been processed to add to the database, the table is updated accordingly. Thus the table reflects the incremental updates. And because of this, it is basically the same algorithm to add a new segment to a database whether it has no recordings or 1000 hours of recordings and 1000 clusters. Both cases simply add a new entry to the table which is already in the memory, although the latter takes some more time to compare the segment with 1000 cluster centroids.

As the clustering may make mistakes, a manual correction mechanism is proposed (not implemented yet). For a certain segment which should belong to cluster i but clustered with cluster j, the user will first notice this when he or she tries to listen to the

recordings for the composition which is represented by cluster j. If this segment is not the cluster centroid for cluster j, there are 3 choices:

- A new line will be appended to the database files with the correct cluster ID i provided by the user. In this case, when the system tries to load the database into the memory, it will first add the old line in the database where the segment belong to cluster j, but later it will find this segment belongs to cluster i. Thus the previous entry will be removed.

- In the database file, the old cluster ID j for the misclustered segment can be overwritten with the new one (i). This requires a search to find the file position of the segment first.

- Since the database is loaded into memory as a mapping between cluster ID and a list of segments within that cluster, the misclustered segment can simply be updated in memory and then written to the database file.

The first choice can be efficient, as searching or rewriting the entire database file every time a mistake is found can be time-consuming, especially when the database gets larger. One could also use a more sophisticated data structure on disk to facilitate efficient updates.

If the segment is the cluster centroid for cluster j, the situation becomes trickier as it cannot be simply updated directly. Some of the information is related to the misclustered cluster centroid, e.g. the local CENS centroid file. The first step is the same with the situation above, assume the first if chosen, writing a new line to the database line. Since the data structure storing the database in memory is a table which maps the cluster ID to a list of segments, to find the second longest segment can be accomplished by doing a linear search within the list corresponding to the cluster j. The length of a segment can be computed by end time index minus start time index, both of which are stored as the segment information in table above.

6 Interactive Music Display

Ultimately, the purpose of the system is to integrate the rehearsal audio into a digital music display and practice support system. The display system acts as bi-directional interface: display as *input* and display as *output*. A implementation of this music display is shown in Figure 12.

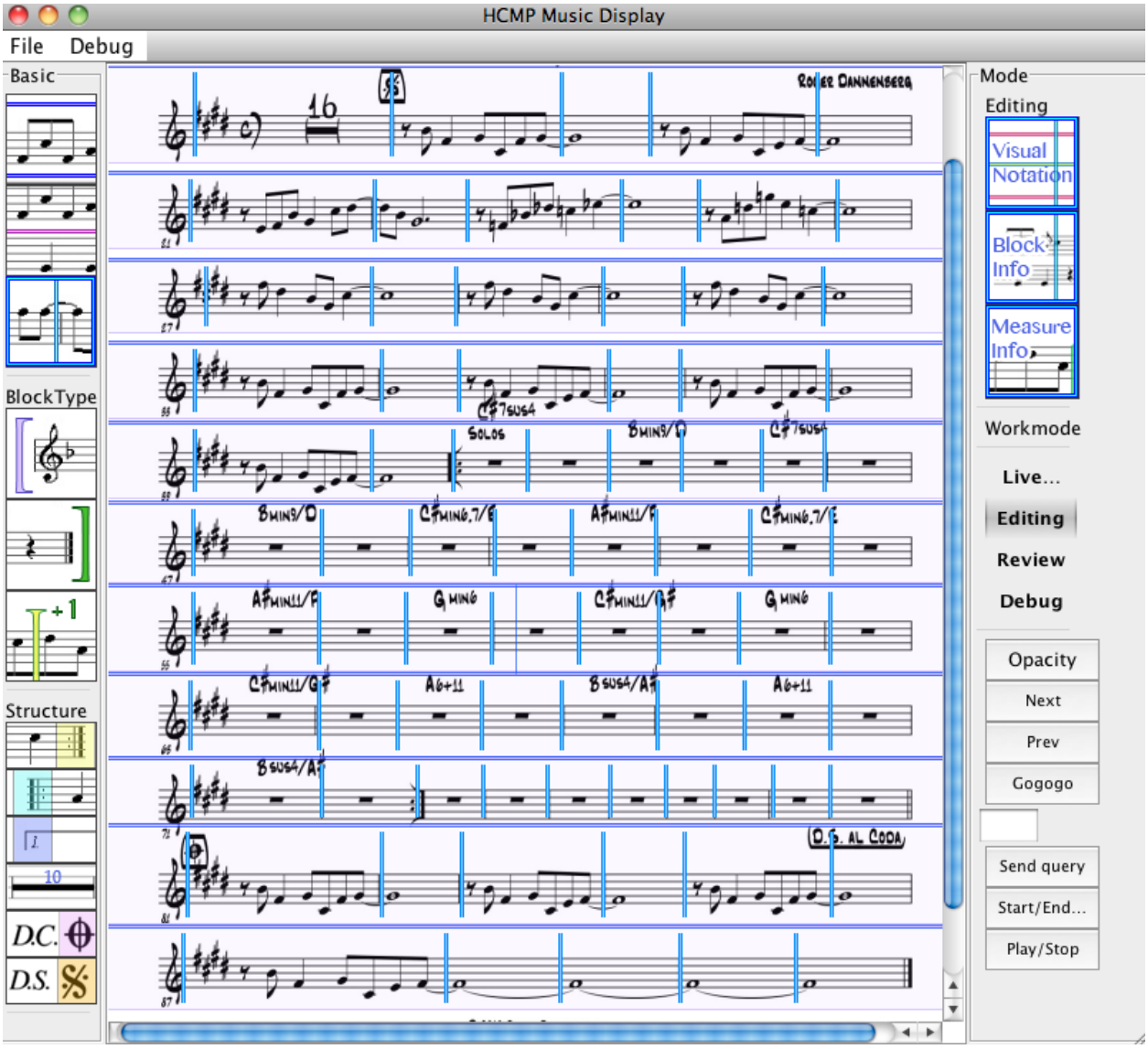

Figure 12: A labeled score.

6.1 Display as *Input*

In order to use the display interface as input, a link is needed between display location and audio location for at least one recording. As proposed in Liang et al. (2011), while listening to a performance, the user can tap on music locations to establish a correspondence between music audio and music notation. To integrate with the system in Section 4 and 5, it is enough to only tap the beat locations for the cluster centroid for each cluster, as once the music has been annotated in this manner, audio-to-audio alignment within a cluster can be accomplished automatically, as described in Section 5.3. The user can then point to a music passage in order to call up a menu of matching audio sorted by date, length, tempo, or other attributes.

6.1.1 Static vs. Dynamic Score

Even if the correspondence between music audio and music notation is established, there is an important difference between scores and audio, as scores are more like a program that must be "executed" before it can be played properly. Repeats and the "Dal Segno (D.S.) al coda" are forms of looping behavior. First and second endings and the coda are forms of conditional behavior based on the loop count. Taking the score in Figure 12 as an example, the repeat signs in the first and third system indicate that after reaching the repeat sign in the third system for the first time, the performer should go back to the repeat sign in the first system. There is also a 2rd repeat sign to indicate which measure should be played after the second repeat. Thus, the score is a "static" representation of music in the sense of static code, and an audio file is a "dynamic" representation of music in the sense of dynamic or run-time program behavior. Because of this static property, the mapping between audio location and score location is actually one-to many, which increases the difficulty for an interactive music display interface. The proposed solution in Liang et al. (2011) is that, given a static score (an image), the system first asks user to manually label all the notation positions, e.g. vertical lines to separate systems, bar lines, repeat signs, D.S. and so on (as shown in the left column of the UI in Figure 12). With this position information, the system can convert the static score to the corresponding

dynamic score by "unfolding" the repeats and computing a list of dynamic measures. Thus, all the display functionality can be operated on the dynamic score where a one-to-one mapping exists with music audio.

Manually labeling the score event positions is tedious. Some automatic mechanisms based on Optical Character Recognition (OCR) have already been used in commercial software. However, it is difficult to avoid mistakes during the recognition process. Furthermore, for some non-traditional scores with the non-standard notations or hand-written sketches (shown in Figure 13), the system presented in this thesis can still handle it, while OCR-based mechanism cannot.

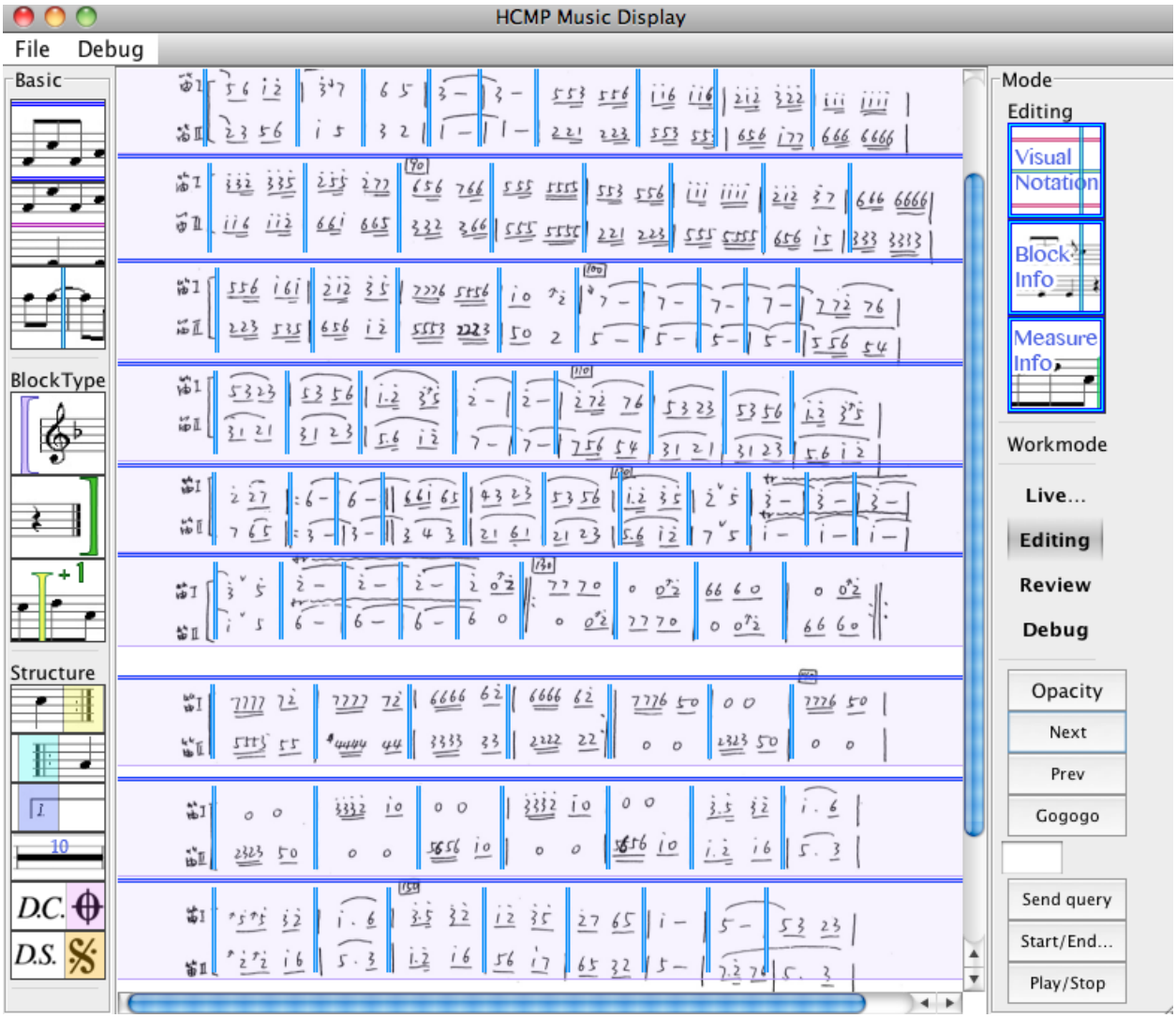

Figure 13: A labeled non-traditional score.

6.2 Display as *Output*

After inputing the necessary information to the display system, the user can then practice with the recording in order to work on tempo, phrasing, or intonation, or the user might simply review a recent rehearsal, checking on known trouble spots. When playing the segments, the interface can even indicate the currently playing position in the score, as the beat information (score position) and the time information in the audio have been aligned. One of the exciting elements of this interface is that useful audio can be made available quickly through a natural, intuitive interface (music notation). It is easy to import scanned images of notation into the system and create these interfaces.

As with the problem presented in Section 6.1.1, a static score display as output will also exhibit the problem of a many-to-one mapping. By "unwrapping" the score into a dynamic score in which the same static measure may occur multiple times, the mapping from score to audio position becomes one-to-one and unambiguous.

Another problem is automatic page turning. Various schemes have been implemented for "page turning" on a display screen of limited size. It is well known that musicians read ahead, so it is essential to display the current music as well as several measures in the future. The most common approach is to split the screen into top and bottom halves. While the musician reads one half, the computer updates the other half to the next system(s) of music. Other solutions include: scrolling up at a constant speed, scrolling up by one system when it is finished, scrolling at a variable speed which is proportional to the tempo, and scrolling an infinitely wide score horizontally. As implemented in Liang et al. (2011), the screen is divided into thirds and always displays previous, current, and next systems. For example, the initial display shows the first 3 systems as 1-2-3. When the player advances to the third system, the display is updated to 4-2-3. Then when the player continue to system 4, the display is updated to 4-5-3, etc.

6.3 Display Implementation

The display interface is implemented in Java. The Java Swing library provides a flexible and portable graphics layer. The display interface was developed for another project

named "Human Computer Music Performance (HCMP)" and described in Liang et al. (2011). Currently the Java implementation is still in its early stage, where the dynamic and static score conversion functionality has not been fully implemented. ZeroMQ[4] is used as a toolkit for communication between C/C++ and Java. Java Native Inteface (JNI) is often used as a framework to enables Java to call, and to be called by, libraries written in C or C++. However, in the case of this system, ZeroMQ is preferable to JNI because it allows the music display system to present a language-independent interface, as messages are passing via socket, which is supported by almost every language.

7 Experiments

In this section, the experimental results are presented, including: music/non-music classification results with AdaBoost, the result after HMM smoothing, and the clustering results based on Algorithm 1. Finally performance measurements are presented.

7.1 Music/Non-music classification

7.1.1 AdaBoost Evaluation

Data Collection and Representation The training data for AdaBoost is a collection of about 5 hours rehearsal and performance recordings of *Western music*; while the testing data is a collection of 2.5 hours of *Chinese music*. The distinct characteristics of Western and Chinese music will increase the difficulty of the music/non-music classification task, and show the robustness of the system presented here. Data collection can be described as follows:

- For the music parts, each data collection contains different combinations of wind instruments, string instruments, and singing voices.
- For the non-music parts, each data collection contains speech, silence, applause, noise, etc.

[4] http://www.zeromq.org/

Both data collections are labeled as music or non-music at the long frame level (1.25 seconds) as introduced in Section 4.1. From Section 4.1, it is also known that each long frame is a point in the 10-dimensional Eigenmusic space. Therefore, the number of total long frames for training data is $5(\text{hours}) \times 3600(\text{seconds/hour})/1.25(\text{seconds/frame}) = 14400$ and similarly, testing data has 7200 long frames.

Implementation and Evaluation The number of weak classifiers for AdaBoost is a parameter which can be adjusted for optimal performance. In the case of music/non-music classification, 100 weak classifiers are trained to construct the final strong classifier. The testing error rates for music and non-music are shown in Figure 14. The x-axis represents the number of weak classifiers. It can be observed that after 40 weak classifiers are trained, the error rates for both music and non-music converge. Therefore, for the computation in the system, only the weighted sum for the first 40 weak classifiers are necessary, from which significant computation can be saved.

The results were obtained in terms of the percentage of error at the frame level. Two different statistics have been calculated: the percentage of true music identified as non-music (false negative), shown as the solid line, and the percentage of true non-music identified as music (false positive), shown as the dotted line. It can be seen that the proposedsegment classifier in the Eigenmusic space is capable of achieving a relatively low error rate (about 5.5%) on both music and non-music data, even when the testing data comes from a completely different sound source from the training data.

As this task is new to the fields of MIR and Speech, there is no result from previous work to compare with. However, taking the results from the task of music/non-music classification for broadcast streaming audio using a Bayesian network (Pikrakis et al., 2008) with approximately 96% overall accuracy as state-of-the-art, the result from AdaBoost is comparable, and HMM smoothing will further improve the performance.

7.1.2 HMM Smoothing Evaluation

At the frame level, HMM smoothing reduced the error rate from about 5.5% to 1.8% on music and to 2.2% on non-music, which is a significant improvement over the already

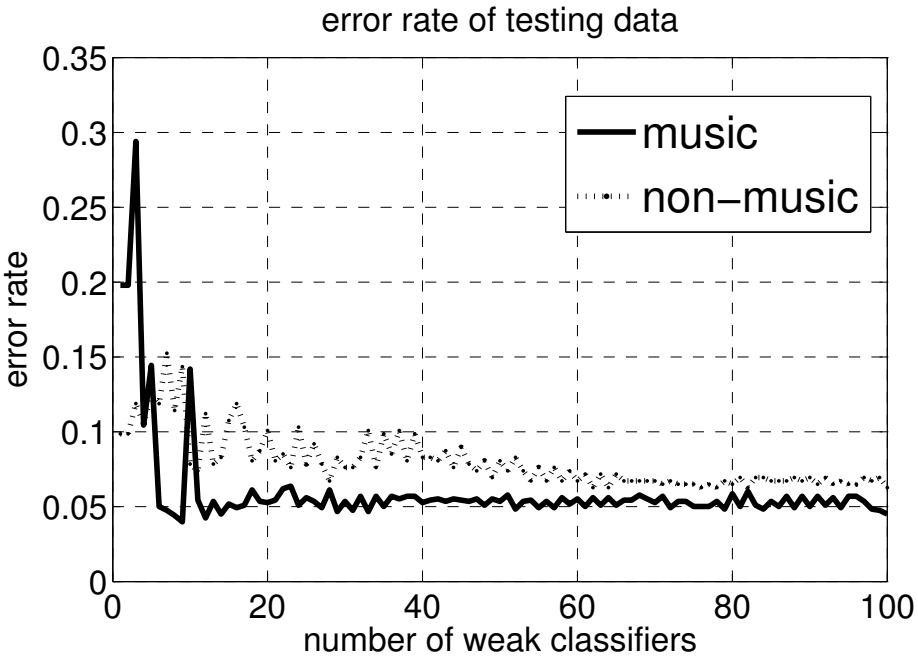

Figure 14: The testing error of music and non-music.

high accuracy from AdaBoost.

However, the results on the frame level are not so important in the sense that the only purpose of this step is to extract the music pieces/segments, not music frames. Since the pieces/segments level evaluation has been largely ignored in previous work on music/non-music classification, an evaluation method from speech segmentation community called Fuzzy Recall and Precision (Ziółko et al., 2007) is adopted. This method pays more attention to insertion and deletion than boundary precision. A Fuzzy Precision of 89.5% and Fuzzy Recall of 97% are obtained. The high Fuzzy Recall reflects that all true boundaries are well detected with only some imprecision around the boundaries. The lower Fuzzy Precision reflects that about 10% of the detected boundaries are not true ones. As mentioned later in Section 7.2.1 as well, this is more desirable than making sure all the detected boundaries are correct, but missing true boundaries which distinguish 2 different compositions. In that case, an incorrectly long cluster centroid may be used in the clustering process, where the ground truth actually belongs to 2 clusters.

7.2 Music Segments Clustering

To find the optimal cluster performance, two parameters can be tuned as discussed in Section 5.2.1: γ, threshold value, which determines if the two segments are close enough to be clustered together, and t, the length of the subsequence of the segments.

7.2.1 Experimental Evaluation

Hours of rehearsal recordings as test data are used, with styles that include classical, rock, and jazz. Live performance recordings, which are typically even longer, are also used. To evaluate the clustering results, F-measure is used as discussed in Manning et al. (2008):

$$\begin{aligned} \mathrm{P} &= \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}}, \quad \mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \\ \mathrm{F}_\beta &= \frac{(\beta^2+1)\mathrm{PR}}{\beta^2\mathrm{P}+\mathrm{R}} \end{aligned} \tag{9}$$

Here, P (precision) and R (recall) are determined by 4 different variables: TP (true positive) which corresponds to assigning two similar segments to the same cluster, TN (true negative) corresponding to assigning two dissimilar segments to the different clusters, FP (false positive) corresponding to assigning two dissimilar segments to the same cluster, and FN (false negative) which corresponds to assigning two similar segments to different clusters. β is the tuning parameter used to adjust the emphasis on precision or recall. In the case of this thesis, it is more important to avoid clustering segments from different pieces into one cluster than it is to avoid oversegmenting by creating too many clusters. The latter case is more easily rectified manually. Thus, it would be more preferable to penalize more on false positives, which leads to a choice of $\beta < 1$. Here, $\beta = 0.9$ is used.

Considering the possible noise near the beginning and the end of the recordings, the middle t seconds are chosen if the length of the full segment is greater than t seconds. As seen in Figure 15, for subsequence longer than 3 minutes, the relatively larger $\gamma = 0.25$ outperforms others, while for shorter subsequence length around 20 seconds to 60 seconds, the smaller $\gamma = 0.15$ has the best performance with 0.95 F-measure value. It is also shown that if γ is set too large (0.35), the performance drops drastically. Overall,

shorter subsequence length and smaller γ gives better results than longer subsequence length and larger γ. Finally, since calculating correlation has $O(n^2)$ complexity, shorter subsequence lengths can also save significant computation. Thus, the system described in this thesis uses a segment length t = 40 seconds and γ = 0.15.

It is hard to compare the cluster results presented here with some other algorithms. First of all, these results are on the training data set, and there is no independent evaluation on a test set, which might reveal some overfitting of parameters t and γ. Second, there is no previous work on this task to serve as a baseline. K-means clustering, as one of the most classic clustering algorithm, was tested but did not work as well as the algorithm here because of the non-uniform segment length and unknown number of clusters.

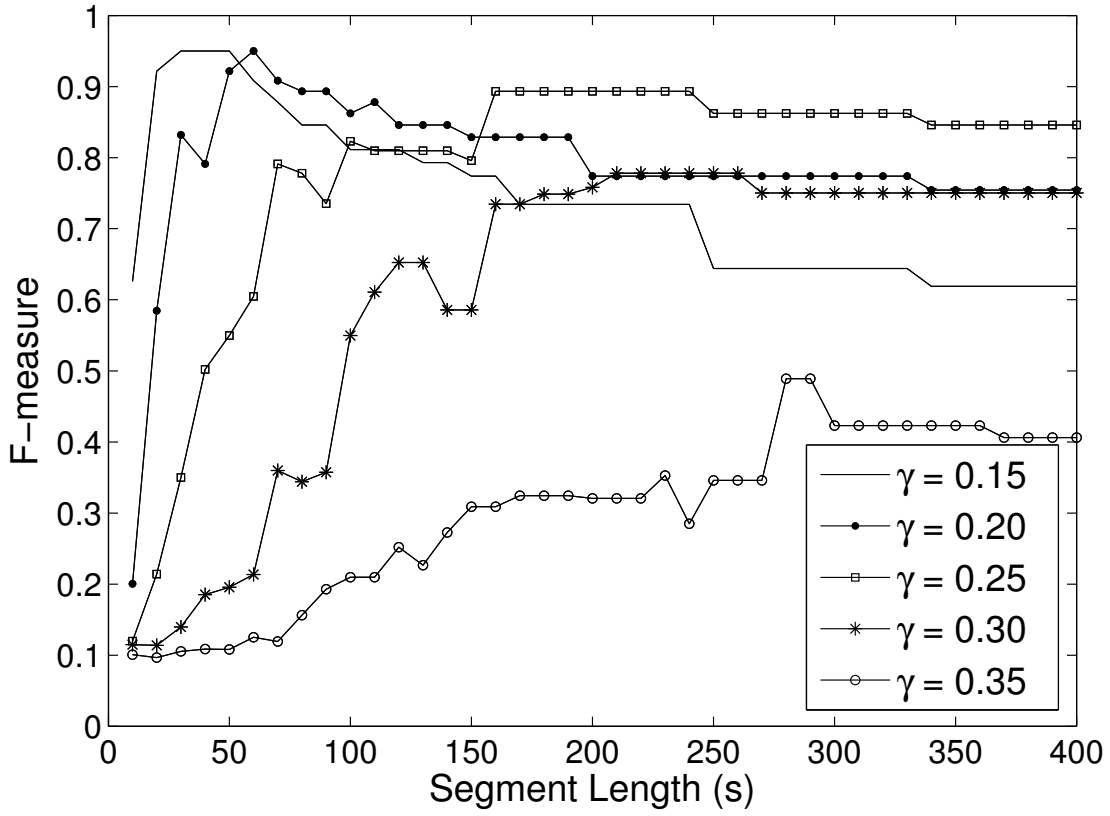

Figure 15: Experimental results with different segments of length t and matching threshold γ.

7.3 Performance Measurement

One of the most important factors for a working system is how fast it could accomplish the task. In this section, the performance measurement in terms of running time is presented for the segmentation and clustering components, which are expected to be the most computationally intensive. Here, all the running time measurements provided are from the experiment with a 5-minute-long rehearsal recording and 9 existing segments of a total length of about 1260 seconds (21 minutes) in the database with 5 clusters. The total length of all the cluster centroids is about 820 seconds (less than 14 minutes). The time complexity in big-O notation is provided as well so that the running time for input of arbitrary length can be estimated.

Music and Non-music Segmentation The running time for different sub-components of the segmentation is shown in the table below (all the time are in seconds):

Total	Resample	FFTs	AdaBoost + HMM
10.961	8.789	1.818	0.354

From the table, it can be observed that most of the running time is spent on resampling the audio to 16000 Hz. Here the resample is done using the *libsamplerate* toolkit[5]. Therefore, part of the future work is to implement a more efficient resampling function.

To represent the running time in big-O notation, the length of *one* rehearsal recording is denoted as $O(n)$ to represent different quantities in every step, i.e. the number of samples during resampling, the number of frames while computing the FFTs, and the number of longer frames (averaging among 5 FFT frames) during AdaBoost prediction and HMM smoothing. Note that all the constants are in upper case, and all the variables are in lower case.

The implementation for *libsamplerate* is unknown. But in general, resampling runs in $O(Cn)$, where during downsampling (which is the common case) C is the length of the low-pass filter to prevent aliasing. For FFTs, using the classic algorithm, the time is $O(nL \log L)$ where L is the length of the FFT frames. The AdaBoost prediction can

[5] http://www.mega-nerd.com/SRC/

be done in $O(Tn)$ as it only computes the weighted sum of the prediction for T weak classifiers ($T = 40$, as observed in Section 7.1.1). The same applies to *Viterbi* algorithm, the running time is in $O(Sn)$, where S is the number of the states (in the case of the system $S = 2$). Therefore, the total time complexity of the segmentation component is $O((C + L \log L + T + S)n)$, which grows linearly with the length of the input audio.

Segment Matching and Clustering The clustering takes 1.401 seconds to finish all 3 of the segments in the rehearsal recordings.The length of the 3 segments are 66 seconds, 43 seconds and 81 seconds, respectively. The running time for each of the 3 segments are shown below, which is decomposed into 2 separate running time: time to compute the CENS and the time to finish matching and clustering.

	Segment 1	Segment 2	Segment 3
Compute CENS	0.298	0.301	0.304
Match & Cluster	0.022	0.019	0.023

In terms of the running time in big-O notation, the time used to compute the CENS of query segment should be $O(n)$, as it is based on the FFTs. However, based on the experiment in Section 7.2.1, the length of CENS is fixed (the default setting is $D = 40$), thus the time in most cases is $O(1)$, and $O(n)$ only if a new cluster is created with this segment, in that case the CENS for the full length is computed and saved as a local file. The amortized analysis can be used to analyze the general time complexity $\psi(n)$. However, even without mathematical derivation, it is guaranteed that $\psi(n)$ must satisfy $\psi(n) \leq O(n)$ which is enough for the large-scale rehearsal recordings. Assume the length for each cluster centroid is m_i, the time to perform matching and clustering is in $O(D \sum_{i=1}^{k} m_i)$, where k is the number of existing clusters. D appears here to represent the time spend on each distance computation. Note that $\sum_{i=1}^{k} m_i$ is the total length of all the cluster centroids. The total time spent on the clustering component is $O(\psi(n) + D \sum_{i=1}^{k} m_i)$, which grows linearly with the number of the clusters and the length of each cluster centroid.

The streaming nature of the segmentation component and the incremental updates for clustering enables the system work with large-scale rehearsal recordings. As presented in

Section 5.4, it is almost equivalent to add a new segment to a database with no recordings and with 1000 hours recordings and 1000 clusters. To form the running time complexity for the whole system given a new rehearsal recording with length $O(n)$, the time on segmentation component is $O((C + L \log L + T + S)n)$ which can be simplified as $O(\mathcal{C}n)$, then assume N segments are generated. Since only one rehearsal recording is processed, it is reasonable to assume this is from one rehearsal which should be at most a few hours, thus N should be in the order of 10 or 100. In the process of the clustering component, for each segment, the time complexity is $O(\psi(n)+D\sum_{j=1}^{k} m_j)$. Thus, the total time for N segments is $\sum_{i=1}^{N} O(\psi(n) + D\sum_{j=1}^{k} m_j)$. Adding the running time for 2 components together, the total time for processing the rehearsal recording of length $O(n)$ into the database, where N segments are generated and k clusters already exist, is:

$$
\begin{aligned}
& O(\mathcal{C}n + \sum_{i=1}^{N}(\psi(n) + D\sum_{j=1}^{k} m_j)) \\
= & O(\mathcal{C}n + N\psi(n) + DN\sum_{j=1}^{k} m_j)
\end{aligned}
$$

From the total time complexity, given a rehearsal recording, $O(\mathcal{C}n + N\psi(n))$ is a linear form of the length $O(n)$ of the recording. As the number of existing clusters k increases, $O(DN\sum_{j=1}^{k} m_j)$ grows linearly. Therefore, system runs *linearly* with the length of the recording and the number of existing clusters, which guarantees the system could still work with large-scale rehearsal recordings.

For example, consider a database with 1000 hours of recordings (about 0.6 TB of CD quality audio). Assuming music is rehearsed 10 times on average, there are about 100 hours of segments as cluster centroids. From the clustering measurements, it can be estimated that each new segment will take about 0.3 second to compute CENS features and about 9.5 seconds for the match and cluster step. Thus, it takes much less time to update the database than it takes to record the rehearsal in the first place.

8 Applications

Aside from the basic usage of the system which organizes the rehearsal recordings and provides an interface for the user to play along for practice, there are many possible applications based on the system presented in this thesis. Some of them are introduced here:

Download Labeling One of the most tedious parts for the system is that the users have to manually label the score event position. However, if there is a score with correct manual labeling, all the users can download both the score and the labeling for convenience. This score and labeling could be provided by the music publisher as part of a purchased music download.

Combination with Music Minus One Music Minus One[6] is a company that provides music recordings with one of the instruments omitted. Music Minus One recordings are designed as a practice aid, but it is sometimes difficult to find a starting location when the musician wants to play a particular passage. The score display interface presented in Section 6 would allow the musician to simply point to a starting location and the system would automatically start the Music Minus One recording at the right place. In fact, users would not need any special support from Music Minus One to enhance their practice experience. They could simply load Music Minus One recordings into their system, scan their personal scores, and manually annotate the score with timing information.

Search by Attribute In cases where there are many recordings of a given piece of music, it might be interesting to search by different musical properties. What was the slowest performance of this piece? What was the fastest? How has the tempo changed over time? The automated audio alignment computation makes it easy to compare tempos in different recordings. Using fairly simple audio feature extraction, one could also search for the brightest or darkest performance. With more advanced techniques such as emotion classification, one could search for the happiest or saddest version of a song.

[6] http://musicminusone.com/

Musicological Study While this system is inspired by the need for practice tools, it could also be used with professional recordings. One could load a library of classical recordings and have them clustered by piece. Then one could have instant access to, say, the oboe cadenza in Beethoven's Fifth Symphony. Of all the recordings, who played the cadenza the fastest or the slowest? How to the cadenzas compare? The audio alignment and music notation interface make it easy to locate the audio.

More Practice Tools When practicing a difficult passage, repetition is essential. It would be simple to automatically cycle through a difficult passage many times while the musician plays along. The music notation interface would provide a natural place to indicate exactly what to repeat. (The MakeMusic, Inc. (2011) system supports repetition, but the user must manually enter measure numbers to create a repeat.) In addition to repetition, practicing at different tempos (especially slow ones in the beginning) is an established way to develop virtuosity. Audio time stretching could be used to play audio from professional CDs, Music Minus One recordings, or rehearsal recordings at any desired tempo. In addition to audio playback, the rehearsal system could listen to the musician and estimate pitch to assist with intonation, record performances to allow the musician to listen critically to a practice performance, or even compare details of articulation, rhythm, and vibrato to recordings of a teacher, for example.

9 Previous work and Contributions

The work in this thesis is based on the joint work between Guangyu Xia, Roger B. Dannenberg, and Mark J. Harvilla (Liang et al., 2011; Xia et al., 2011), thus it is not starting from scratch. The original work was done as the course project of Machine Learning for Signal Processing[7] (11-755), resulting in prototypes of two of the three components (segmentation and clustering). Guangyu Xia worked on the AdaBoost and HMM approach for segmentation and provided an implementation in MATLAB, and I worked on the segment matching and clustering and proposed the unsupervised database building

[7] http://mlsp.cs.cmu.edu/courses/fall2010/

algorithm, also in MATLAB. We tested these 2 components separately without any integration. Finally a music display system was implemented jointly by Roger B. Dannenberg and Guangyu Xia. The score display interface was reimplemented in Java by Zeyu Jin and forms the basis for music display in this thesis.

Besides my original contribution on the music segment clustering component, the work in this thesis provides an integrated implementation for the whole system, which is overviewed in Section 3. The original prototypes (in MATLAB) did not address many practical issues. For example, in the prototypes, we saved all data as a .mat file in the MATLAB workspace. However, designing an appropriate relational database-like table (Section 5.4) is one of the most important foundations for the integrated software. Also, the MATLAB prototypes were not interactive or connected to a graphical user interface and score display. However, the connection between display interface and recordings processing backend can be essential for a usable, working system (Section 6.3).

10 Conclusion and Future Work

In this thesis, a system for automated management of a personal audio database for practicing musicians is presented. The system takes the recordings as input, segments them into musically meaningful parts, and then organizes them through unsupervised clustering and alignment. A score display interface based on common music notation allows the user to quickly retrieve music audio for practice or review. This work integrates a music/non-music classifier, HMM-based music segmentation, an unsupervised clustering algorithm for music audio organization, and a notation-based interface that takes advantage of audio-to-audio alignment. Experiments show that both the segmentation and clustering components can achieve comparable results with the state-of-the-art, though it is not easy to find a counterpart in the previous work. A working implementation has be written in C++/Java.

Some of the sub-components are not fully implemented (e.g. segment alignment and the static and dynamic score conversion in display interface). Most of the testing has been done in Mac OS X. Although C++ and Java are cross-platform, there will still be minor

issues which need to be addressed in order to make the software work on Windows and Linux. This software is new and there is not much experience with it. In the future, it could be tested by practicing musicians. This experience will lead to recommendations for new functions and improvements in the interface.

Acknowledgements

First of all, thanks Bhiksha Raj for opening the course Machine Learning for Signal Processing and the suggestions and comments on the work, and the Chinese Music Institute of Peking University for providing recordings of rehearsal for analysis.

I will probably write a Ph.D. thesis in another few years. But as a master's thesis, I would still like to thank those people who give me help through the work in this thesis, and other work during my study in CMU. First and foremost, I would like to thank my advisor Roger B. Dannenberg for all the guidance and assistance he has provided since I first came to CMU. The inspiration I got from Roger (even when he does not have expertise for the topic!) and the rigorous attribute for research will always be invaluable treasures for me. Also many thanks to my colleague, Guangyu (Gus) Xia, for giving me a ride every week to the supermarket. Wish you succeed in the internship hunting. And other members in the Computer Music Group: Dalong Cheng, Zeyu Jin, Ashiqur KhudaBukhsh, and Jiuqiang Tang. Also, thanks to all of my course project partners: Haijie Gu, Mark J. Harvilla, Brendan O'Conner and Jing Xiang. I have learned so much from you.

I must thank my parents, Jianguo Liang and Jianmin Li, for raising me and I can imagine how difficult it is for them to see their only child flying to another continent to study for a few years, who can only go back home for less than a month per year. Also, I would like to thank Qimin Xu, I do not know if I could make it without your support. Your love will always be a courage for me.

Finally, I would like to to dedicate this work to my maternal grandfather Ruixi Li (1931-2012). May he rest in peace.

References

J. Ajmera, I. McCowan, and H. Bourlard. Speech/music segmentation using entropy and dynamism features in a hmm classification framework. *Speech Communication*, 40(3): 351–363, 2003.

M.A. Bartsch and G.H. Wakefield. To catch a chorus: Using chroma-based representations for audio thumbnailing. In *Applications of Signal Processing to Audio and Acoustics, 2001 IEEE Workshop on the*, pages 15–18. IEEE, 2001.

D. Beyerbach and H. Nawab. Principal components analysis of the short-time fourier transform. In *Acoustics, Speech, and Signal Processing, 1991. ICASSP-91., 1991 International Conference on*, pages 1725–1728. IEEE, 1991.

A. Bugatti, A. Flammini, and P. Migliorati. Audio classification in speech and music: a comparison between a statistical and a neural approach. *EURASIP J. Appl. Signal Process.*, 2002(1):372–378, January 2002. ISSN 1110-8657.

S. Chen and P.S. Gopalakrishnan. Speaker, environment and channel change detection and clustering via the bayesian information criterion. In *Proc. DARPA Broadcast News Transcription and Understanding Workshop*, pages 127–132, 1998.

C. Dimitrakakis and S. Bengio. Boosting hmms with an application to speech recognition. In *Acoustics, Speech, and Signal Processing, 2004. Proceedings.(ICASSP'04). IEEE International Conference on*, volume 5, pages V–621. IEEE, 2004.

Y. Freund and E. R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1995.

J. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The annals of statistics*, 28(2): 337–407, 2000.

N. Hu, R.B. Dannenberg, and G. Tzanetakis. Polyphonic audio matching and alignment

for music retrieval. In *Applications of Signal Processing to Audio and Acoustics, 2003 IEEE Workshop on.*, pages 185–188. IEEE, 2003.

T. Izumitani, R. Mukai, and K. Kashino. A background music detection method based on robust feature extraction. In *Acoustics, Speech and Signal Processing, 2008. ICASSP 2008. IEEE International Conference on*, pages 13–16. IEEE, 2008.

D. Liang, G. Xia, and R.B. Dannenberg. A framework for coordination and synchronization of media. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 167–172, 2011.

G. Lu and T. Hankinson. A technique towards automatic audio classification and retrieval. In *Signal Processing Proceedings, 1998. ICSP'98. 1998 Fourth International Conference on*, volume 2, pages 1142–1145. IEEE, 1998.

C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

M. Müller, F. Kurth, and M. Clausen. Audio matching via chroma-based statistical features. *Proc. ISMIR, London, GB*, pages 288–295, 2005.

A. Pikrakis, T. Giannakopoulos, and S. Theodoridis. A speech/music discriminator of radio recordings based on dynamic programming and bayesian networks. *Multimedia, IEEE Transactions on*, 10(5):846–857, 2008.

C. Rhodes, M. Casey, S. Abdallah, and M. Sandler. A markov-chain monte-carlo approach to musical audio segmentation. In *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, volume 5, pages V–V. IEEE, 2006.

J. Saunders. Real-time discrimination of broadcast speech/music. In *Acoustics, Speech, and Signal Processing, 1996. ICASSP-96. Conference Proceedings., 1996 IEEE International Conference on*, volume 2, pages 993–996. IEEE, 1996.

H. Schwenk. Using boosting to improve a hybrid hmm/neural network speech recognizer. In *Acoustics, Speech, and Signal Processing, 1999. ICASSP'99. Proceedings., 1999 IEEE International Conference on*, volume 2, pages 1009–1012. IEEE, 1999.

MakeMusic, Inc. Smartmusic interactive music software transforms the way students practice, 2011. http://www.smartmusic.com.

M.A. Turk and A.P. Pentland. Face recognition using eigenfaces. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR'91., IEEE Computer Society Conference on*, pages 586–591. IEEE, 1991.

A. Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *Information Theory, IEEE Transactions on*, 13(2):260–269, 1967.

H.D. Wactlar, T. Kanade, M.A. Smith, and S.M. Stevens. Intelligent access to digital video: Informedia project. *Computer*, 29(5):46–52, 1996.

G. Xia, D. Liang, and R.B. Dannenberg. Segmentation, clustering, and display in a personal audio database for musicians. In *Proceedings of the International Society for Music Information Retrieval*, pages 139–144, 2011.

B. Zhou and J.H.L. Hansen. Unsupervised audio stream segmentation and clustering via the bayesian information criterion. In *Sixth International Conference on Spoken Language Processing*, 2000.

B. Ziółko, S. Manandhar, and R. Wilson. Fuzzy recall and precision for speech segmentation evaluation. In *Proceedings of 3rd Language & Technology Conference, Poznan, Poland, October*, 2007.