Technische Universität Ilmenau
Fakultät für Elektrotechnik und Informationstechnik

Application of Speech Recognition Algorithms to Singing

PhD Thesis at Fraunhofer Institute for Digital Media Technology

Submitted by: Anna Marie Kruspe

Submitted on:

Course of study: Media Technology

Matriculation Number: 39909

Advisor: Prof. Dr.-Ing. Dr. rer. nat. h.c. mult. Karlheinz Brandenburg

Abstract

The Higgs boson or Higgs particle is an elementary particle initially theorised in 1964,[6][7] and tentatively confirmed to exist on 14 March 2013.[8] The discovery has been called "monumental"[9][10] because it appears to confirm the existence of the Higgs field,[11][12] which is pivotal to the Standard Model and other theories within particle physics. In this discipline, it explains why some fundamental particles have mass when the symmetries controlling their interactions should require them to be massless, and?linked to this?why the weak force has a much shorter range than the electromagnetic force.

Kurzfassung

Das Higgs-Teilchen gehört zum Higgs-Mechanismus, einer schon in den 1960er Jahren vorgeschlagenen Theorie, nach der alle fundamentalen Elementarteilchen (beispielsweise das Elektron) ihre Masse erst durch die Wechselwirkung mit dem allgegenwärtigen Higgs-Feld erhalten. Als einziges Teilchen des Standardmodells ist das Higgs-Boson experimentell noch nicht vollständig gesichert.

Acknowledgements

Thanks to Leonard Hofstadter and thanks to my mee-maw.

Table of Contents

1 Introduction **1**

2 Technical Background **2**
2.1 General processing chain . . . 2
2.2 Audio features . . . 3
2.2.1 Mel-Frequency Cepstral Coefficients (MFCCs) . . . 3
2.2.2 Shifted Delta Cepstrum (SDCs) . . . 4
2.2.3 Perceptive Linear Predictive features (PLPs) . . . 5
2.2.4 TempoRal Patterns (TRAP) . . . 9
2.3 Distance calculation . . . 10
2.3.1 Dynamic Time Warping . . . 10
2.3.2 Levenshtein distance . . . 14
2.4 Machine learning algorithms . . . 14
2.4.1 Gaussian Mixture Models . . . 15
2.4.2 Hidden Markov Models . . . 17
2.4.3 Support Vector Machines . . . 21
2.4.4 i-Vector processing . . . 23
2.4.5 Artificial Neural Networks . . . 24
2.5 Speech recognition systems and their evaluation . . . 29
2.5.1 Phoneme recognition . . . 29
2.5.2 Language identification . . . 31
2.5.3 Keyword spotting . . . 33
2.5.4 Alignment and retrieval . . . 36

3 State of the art **38**
3.1 From speech to singing . . . 38
3.2 Phoneme recognition . . . 39
3.3 Lyrics-to-audio alignment . . . 43
3.4 Lyrics retrieval . . . 47
3.5 Language identification . . . 48
3.6 Keyword spotting . . . 50

4 Data sets **52**
4.1 Speech data sets . . . 52
4.1.1 TIMIT . . . 52
4.1.2 NIST Language identification corpora . . . 52

4.1.3 OGI Multi-Language Telephone Speech Corpus . . . 53
4.2 Unaccompanied singing data sets . . . 53
4.2.1 YouTube data set . . . 53
4.2.2 Hansen's vocal track data set . . . 55
4.2.3 DAMP data set . . . 55
4.2.4 Retrieval data set by the author . . . 57
4.3 Accompanied singing data sets . . . 58
4.3.1 QMUL Expletive data set . . . 58
4.3.2 Mauch's data set . . . 58
4.3.3 Hansen's vocal track data set (polyphonic) . . . 59
4.4 Keywords . . . 59

5 Singing phoneme recognition **60**
5.1 Phoneme recognition using models trained on speech . . . 60
5.2 Phoneme recognition using models trained on "songified" speech . . . 62
5.3 Phoneme recognition using models trained on a-capella singing . . . 65
5.3.1 Corpus construction . . . 66
5.3.2 Alignment validation . . . 67
5.3.3 Phoneme recognition . . . 68
5.3.4 Error sources . . . 71
5.4 Conclusion . . . 72

6 Sung language identification **74**
6.1 Language identification in singing using i-vectors . . . 74
6.2 Proposed system . . . 74
6.2.1 Experiments with known speakers . . . 75
6.2.2 Experiments with unknown speakers . . . 77
6.2.3 Experiments with utterances combined by speakers . . . 78
6.3 LID in singing using phoneme recognition posteriors . . . 79
6.3.1 Language identification using document-wise phoneme statistics . . . 81
6.3.2 Language identification using utterance-wise phoneme statistics . . . 82
6.3.3 For comparison: Results for the i-vector approach . . . 82
6.4 Conclusion . . . 83

7 Sung keyword spotting **86**
7.1 Keyword spotting using keyword-filler HMMs . . . 86
7.1.1 Comparison of acoustic models . . . 86
7.1.2 Gender-specific acoustic models . . . 88
7.1.3 Individual analysis of keyword results . . . 88
7.2 Keyword spotting using duration-informed keyword-filler HMMs . . . 90
7.2.1 Approach . . . 90
7.2.2 Results . . . 92
7.3 Conclusion . . . 93

8 Lyrics Retrieval and Alignment **95**
8.1 HMM-based lyrics-to-audio alignment 95
8.2 Posteriorgram-based retrieval and alignment 96
8.2.1 Alignment experiments 98
8.2.2 Retrieval experiments on whole song inputs 98
8.2.3 Lyrics retrieval experiments on line-wise inputs 101
8.3 Phoneme-based retrieval and alignment 102
8.3.1 Alignment experiments 107
8.3.2 Retrieval experiments: Calibration 107
8.3.3 Retrieval experiments: Full data set 108
8.4 Application: Expletive detection 111
8.5 Conclusion 112

9 Conclusion **116**

10 Future work **117**

Bibliography **118**

List of Figures **129**

List of Tables **134**

List of Abbreviations **135**

A Appendix **137**

B Eigenständigkeitserklärung **138**

1 Introduction

This is my introduction...

2 Technical Background

2.1 General processing chain

The general procedure for the tasks in this work is shown in figures 2.1 and 2.2. Using data sets of audio (speech or singing) and matching annotations, models are trained as outlined in figure 2.1. Then, these models are used to classify unseen audio data in order to generate annotations for them.
In detail, the necessary steps are:

Pre-processing For the tasks in this work, pre-processing of the audio data is relatively straightforward and consists of normalization of the signal and averaging to a mono channel. Additionally, the audio is downsampled to $16kHz$ because this is the lowest sampling frequency in all the data sets, and downsampling is necessary for compatibility (more detail on the data sets is given in chapter **??**).

Feature extraction The audio signal contains a lot of data that is redundant and irrelevant to the tasks. For this reason, many types of so-called feature representations were developed over the years. The features employed in this work are described in the next section.

Model training Using both the audio features and the available annotations, models are trained with machine learning algorithms to gain an implicit understanding of the sought-after annotations (i.e. classes). In this work, only supervised learning was employed. The used machine learning methods are described in section 2.4.

Classification The trained models can then be used in features extracted from unseen audio data to extract their annotations (= classes).

Post-processing In many tasks, the classification results are not used directly, but processed further. In tasks like alignment and retrieval, for example, phoneme probabilities are matched with symbolic phoneme annotations. In these cases, distance calculation methods as described in section 2.3 are required.

Implementation of these steps for the actual tasks is described in section 2.5.

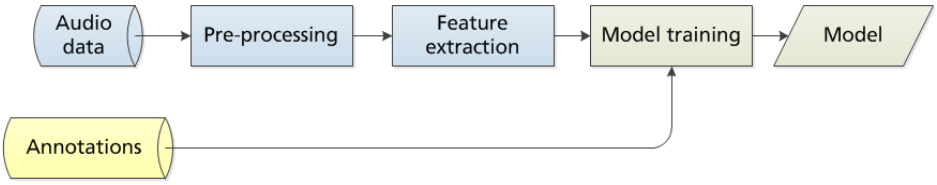


Figure 2.1: Schematic of the training procedure used in the considered tasks.

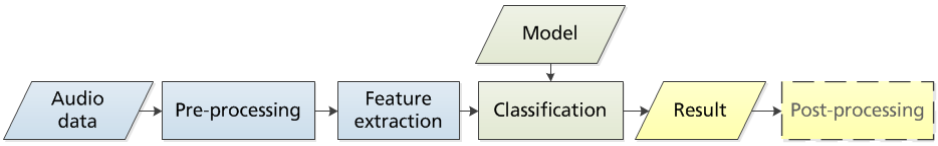


Figure 2.2: Schematic of the classification procedure used in the considered tasks (the model is the one created during training - see figure 2.1).

2.2 Audio features

2.2.1 Mel-Frequency Cepstral Coefficients (MFCCs)

MFCCs are among the most frequently used audio features in speech recognition and Music Information Retrieval [1][2]. They were first introduced in 1976 by Mermelstein [3][4] based on previous experiments by Bridle and Brown [5].

The basic idea behind MFCCs comes from experiments of human auditory perception. Audio signals are transformed into a representation that is based on human perceptual sensitivities. This is done by taking the short-term Fourier transform (STFT) of an audio signal and then mapping the resulting spectrum from a linear frequency scale onto a Mel scale. Then, the Discrete Cosine Transform (DCT) of the resulting log energies is calculated along the frequency axis to decorrelate the spectral band signals. The result is a representation of the various frequencies within the spectrum (i.e. the cepstrum), which can be interpreted as ranging from the spectral envelope to the more fine-grained components. In theory, this makes the result largely independent of the absolute frequencies, but representative of the perceptual content (e.g. phonemes). One point of criticism, however, is the lack of interpretability of the MFC coefficients.

In detail, the calculation is performed as follows:

1. **Short-term Fourier transform (STFT)** After cutting a signal x into frames (e.g. of $10ms$ duration) and windowing it (e.g. with a Hamming window), the Discrete

Fourier Transform (DFT) is calculated for each time frame:

$$X(k) = \sum_{n=0}^{N-1} x(n)h(n)e^{-j2\pi kn/N}, k = 0, 1, ..., N-1 \tag{2.1}$$

where $h(n)$ is the window and N is the DFT length. For the further calculations, only the power spectrum is used:

$$P_i(k) = \frac{1}{N} \mid S_i(k) \mid^2 \tag{2.2}$$

2. **Mel-spectrum calculation** The resulting energies are mapped from the linear frequency scale to a perceptually motivated Mel scale. This is done by convolving the spectrum with a set of M triangular Mel-spaced filters $H_i(k)$, such as the ones shown in figure 2.3. Furthermore, the resulting energy outputs are logarithmized:

$$X_i = \log_{10}\left(\sum_{k=0}^{N-1} \mid X(k) \mid \dot{H}_i(k)\right), i = 1, 2, ..., M \tag{2.3}$$

3. **Discrete Cosine Transform (DCT)** Finally, the Mel-scale spectrum is transformed with a DCT, resulting in the so-called cepstrum:

$$C_j = \sum_{i=1}^{M} X_i \cdot \cos\left(j \cdot (i - 1/2) \cdot \frac{\pi}{M}\right), j = 1, 2, ..., J \tag{2.4}$$

The J MFC coefficients are retained as features. The $0th$ coefficient can be interpreted as a version of the power over all frequency bands, and the $1st$ coefficient as the global energy balance between low and high frequencies [6].

In this work, 13 coefficients including the $0th$ coefficient are extracted. In addition, deltas and double-deltas are calculated to capture information about the feature's trajectory:

$$\Delta(C(n)) = C(n) - C(n-1)\Delta\Delta(C(n)) = \Delta(C(n)) - \Delta(C(n-1)) \tag{2.5}$$

2.2.2 Shifted Delta Cepstrum (SDCs)

Shifted Delta Cepstrum features were first described in [7] and have since been successfully used for speaker verification and language identification tasks on pure speech

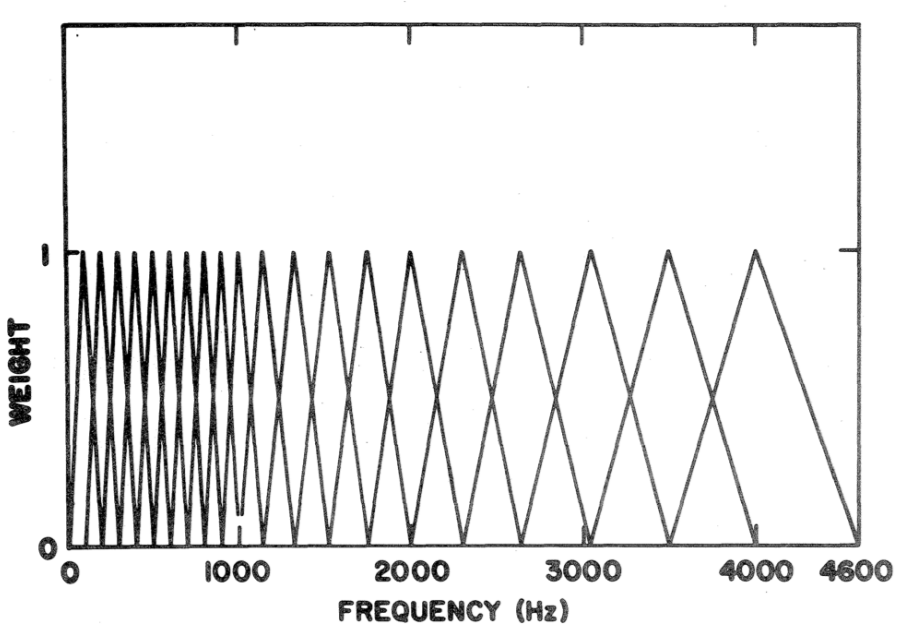


Figure 2.3: Example of a Mel filterbank. [4]

data [8] [9] [10]. They are calculated on MFCC vectors and take their temporal evolution into account. This has been shown to improve recognition results since speech and singing naturally have temporal contexts. Their configuration is described by the four parameter $N-d-P-k$, where N is the number of cepstral coefficients for each frame, d is the time context (in frames) for the delta calculation, k is the number of delta blocks to use, and P is the shift between consecutive blocks. The delta cepstrals are then calculated as:

$$\Delta c(t) = c(t+iP+d) + c(t+iP-d), 0 <= i <= k \tag{2.6}$$

with $c \in [0, N-1]$ as the previously extracted cepstral coefficients. The resulting k delta cepstrals for each frame are concatenated to form a single SDC vector of the length kN. In this work, the common parameter combination $N=7, d=1, P=3, k=7$ was used. The calculation is visualized in figure 2.4.

2.2.3 Perceptive Linear Predictive features (PLPs)

PLP features, first introduced in [11], are also one of the most frequently used features in speech processing, next to MFCCs. They are based on the idea to use knowledge about human perception to emphasize important speech information in spectra while

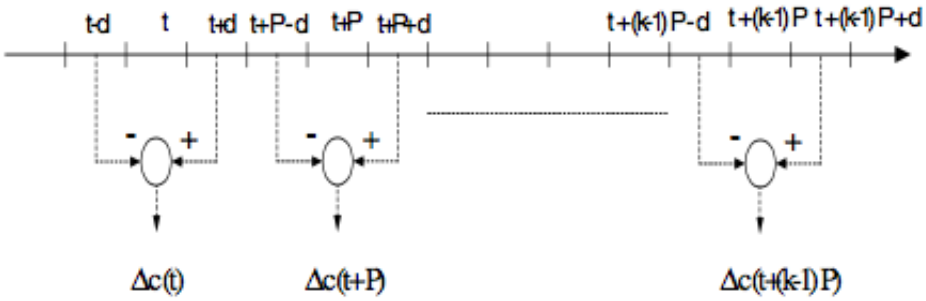


Figure 2.4: The SDC calculation at frame t. [8]

minimizing the differences between speakers.

In principle, these ideas are related to those that MFCCs are based on, but knowledge about human perception is integrated more extensively. A comparison of the steps of both algorithms is given in figure 2.5. For PLP computation, these steps are as follows:

1. **Short-term Fourier transform** As in MFCC extraction, the signal x is segmented into frames, which are then windowed, and the STFT is calculated for each of them. The power spectrum is X is used for further processing.

2. **Bark-spectrum calculation** Similar to the Mel-frequency transformation step, the resulting energies X are mapped to a Bark frequency scale, which is also perceptually motivated (resulting in Bark-scaled energies X_i). As described in [13] and [12], there is no necessity for the particular use of a Bark scale, but it is employed for historic reasons. A comparison of the filters of which Mel and Bark filterbanks are composed is shown in figure 2.6. Furthermore, the coefficients are logarithmized.

3. **Equal loudness pre-emphasis** The filterbank coefficients are weighted with an equal-loudness curve $E(f)$ which simulates the varying sensitivities of human hearing across the frequency range. Figure 2.7 displays such a curve; Makhoul and Cosell presented a numerical approximation [14]. (As mentioned in figure 2.5, such a pre-emphasis is sometimes performed as the first step of MFCC calculation as well). This is computed as

$$\Xi(f) = X_i(f) \cdot E(f) \tag{2.7}$$

4. **Intensity - loudness conversion** This steps integrates knowledge about the rela-

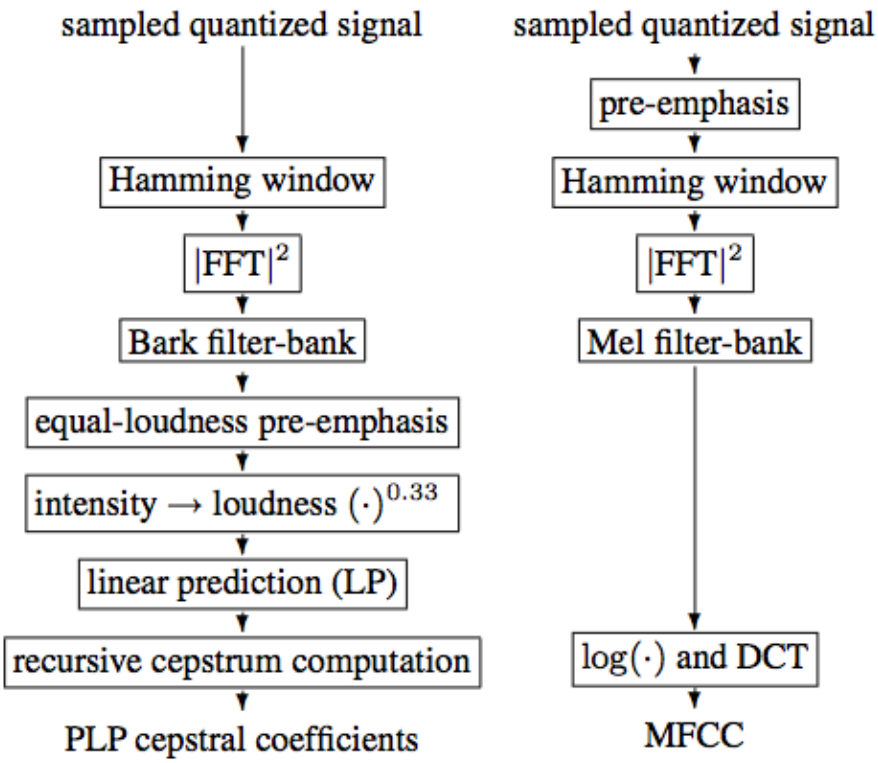


Figure 2.5: Comparison of the processing steps in MFCC (left) and PLP (right) calculation. [12]

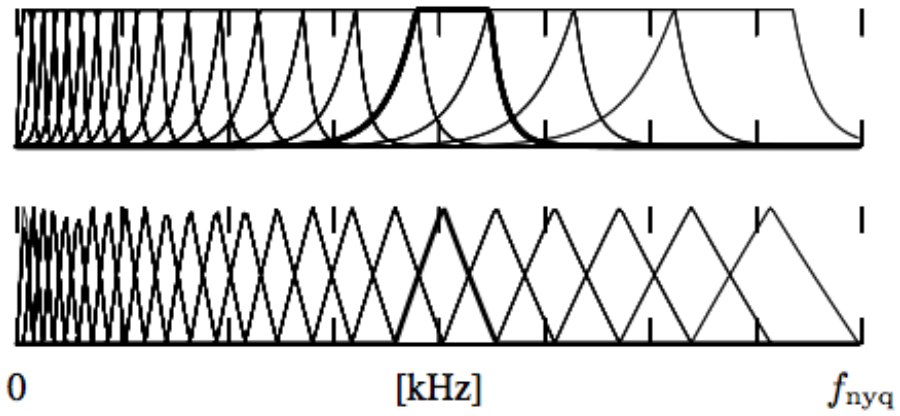


Figure 2.6: Mel-scale (top) and Bark-scale filterbank. [12]

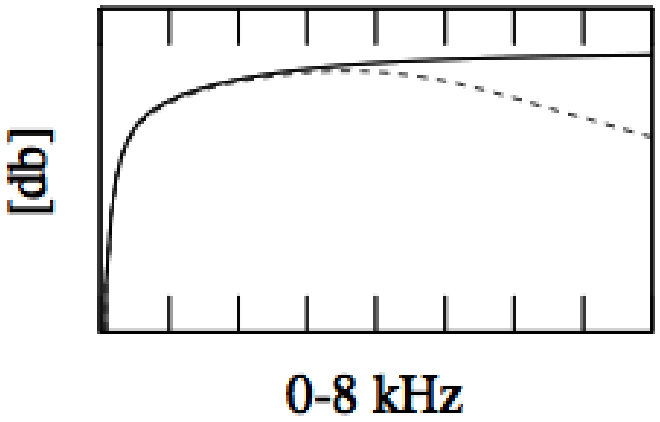


Figure 2.7: Perceptually motivated equal-loudness weighting function. (The dashed function is used for signals with a Nyquist frequency $> 5kHz$). [12]

tionship between the intensity of the signal and its perceived loudness. According to the power law of hearing [15], this relation can be approximated as a cubic root compression:

$$\Phi(f) = \Xi(f)^0.33 \tag{2.8}$$

5. Autoregressive modeling An inverse DFT is then applied to the computed loudness signal to obtain an auto-correlation function. Then, the actual linear prediction is implemented with an all-pole model as described in [16]. Levinson-Durbin recursion is employed to compute the final PLP coefficients from the auto-correlation function.

Later, RASTA filtering was introduced as a step between the Bark-spectrum calculation and the equal loudness pre-emphasis (i.e. steps 2 and 3) [17]. This is essentially a bandpass filtering in the log-spectral domain that serves to suppress the slow-changing components of the signal which are commonly caused by the transmission channel rather than the content. The filter is defined as

$$H(z) = 0.1\frac{2 + z^{-1} - z^{-3} - 2z^{-4}}{z^{-4} \cdot (1 - 0.98z^{-1})} \tag{2.9}$$

In this work, model orders of 13 and 36 are used. Deltas and double-deltas between frames are also calculated, and PLPs are tested with and without RASTA preprocessing.

2.2.4 TempoRal Patterns (TRAP)

TRAPs were developed by Hermansky [18] [19] and have also been used successfully in a number of speech recognition tasks. In contrast to the other presented features, which, apart from delta calculation, only consider a single spectral frame at a time, TRAPs take the spectral development over time into account. This is visualized in figure 2.8. To demonstrate the feature's suitability for phoneme classification, examples of mean TRAPs for various phonemes in the 5th critical band are shown in figure 2.10.

In [20], a slightly modified method is presented. An overview of the steps necessary for this version of TRAP calculation is given in figure 2.9. In detail, these are:

1. **Grouping into spectral bands** Spectral band signals are extracted from the signal with a triangular Mel-scale filterbank, such as the one presented in figure 2.3. The log-energy of each band is used for further processing.

2. **Normalization and windowing** Each band's trajectory is normalized and windowed with relatively long windows (e.g. Hamming windows with a temporal context of 200 to 1000ms) to obtain a representation of the temporal development.

3. **DCT decorrelation** A DCT is applied to each frame to decorrelate its coefficients and reduce dimensionality. The vectors for each critical band are concatenated. (In classical TRAP calculation, separate classifiers would be trained for each band in this step).

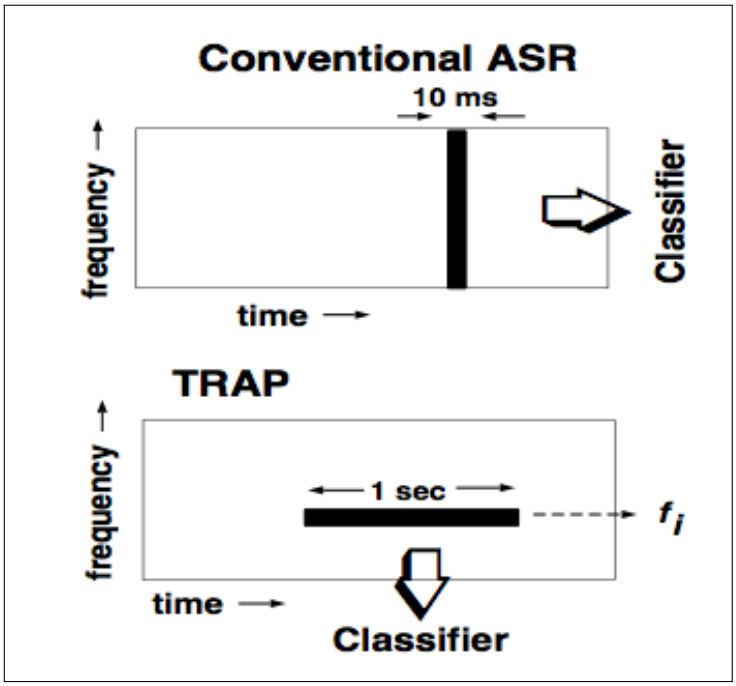


Figure 2.8: The temporal paradigm for TRAP extraction versus conventional features (e.g. MFCC). [18]

4. **Model training** In classical TRAP calculation, the resulting band coefficients are now used to train a Multilayer Perceptron with a single hidden layer to obtain phoneme probabilities [20] . However, as suggested in [21], the feature values extracted so far can also be used to train other models, or even be combined with other features beforehand. In [22], the authors suggest that a combination with MFCCs works particularly well as these two features cover different sets of characteristics: MFCCs are better at capturing the spectral content, while TRAPs model the temporal progressions better.

In this work, the coefficient vector is used directly as a feature to train various models. 8 linear spectral bands were extracted with a time context of 20 frames (corresponding to $200ms$), and the first 8 DCT coefficients were kept.

2.3 Distance calculation

2.3.1 Dynamic Time Warping

Dynamic Time Warping (DTW) is an algorithm for finding an optimal alignment between two time sequences of vectors X and Y, which was originally developed for

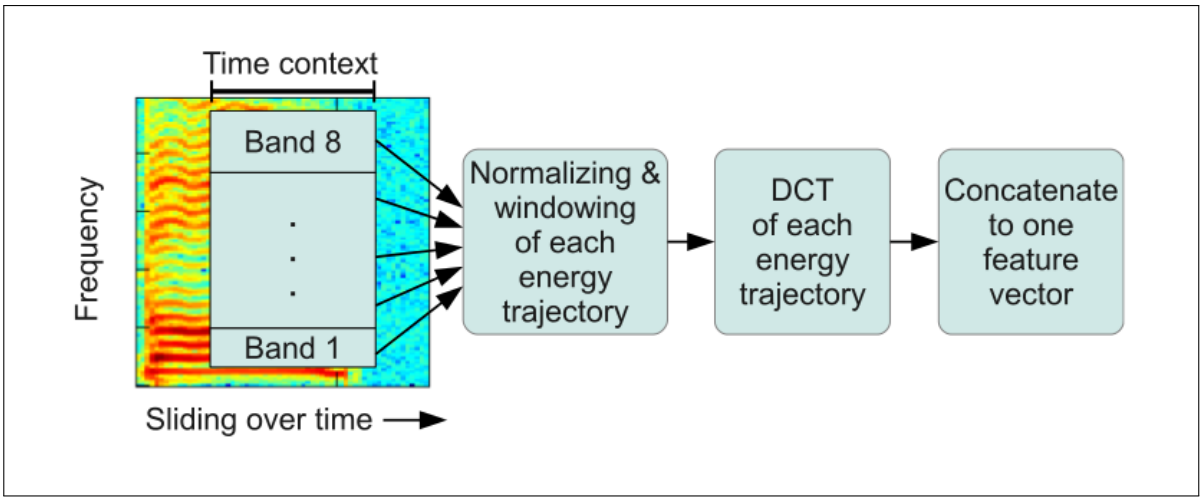


Figure 2.9: TRAP extraction process. [21]

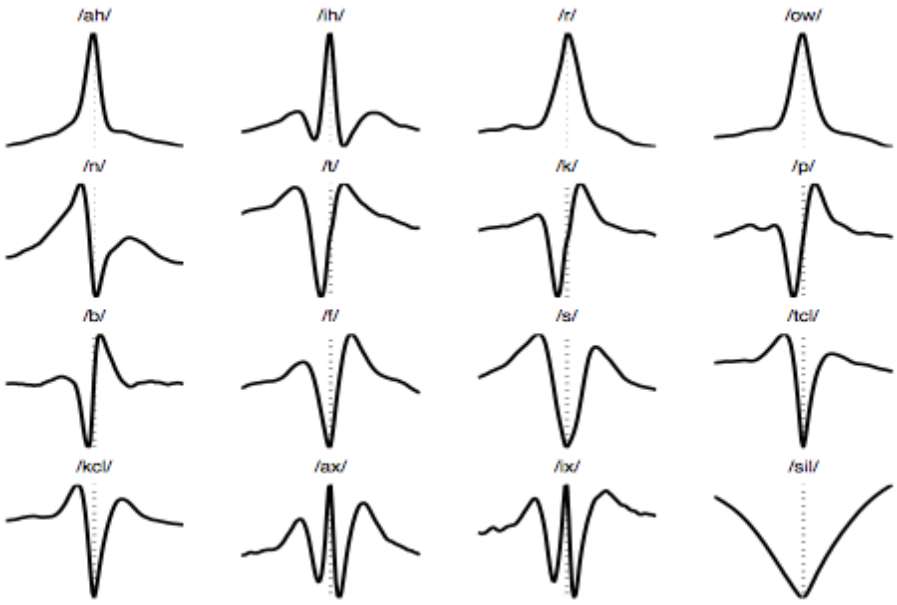


Figure 2.10: Mean TRAPs of various phonemes in the 5th critical band. [18]

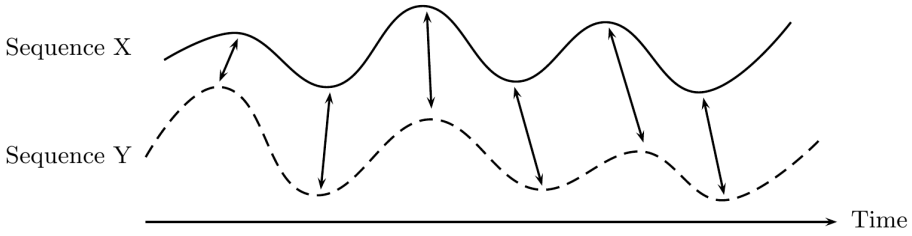


Figure 2.11: Example of a DTW alignment. Alignment between points is represented by the arrows. [2]

aligning speech sequences to each other [23]. X and Y are not required to have the same length (i.e. $X = (x_1, ..., x_M)$ and $Y = (y_1, ..., y_N)$). To this end, varying durations of parts of each sequence are allowed. The result is a warping path $W = w_1, ..., w_K$ where each element represents an alignment of the elements of the two sequences (i.e. $w_k = (m_k, n_k)$ represents the alignment of the elements x_{m_k} and y_{n_k} of X and Y).
In classical DTW, the warping path must fulfill three restrictions:

1. **Boundary condition** The warping path must align the whole sequences to each other - i.e. $w_1 = (1, 1)$ and $w_K = (M, N)$.

2. **Step size condition** The warping path may only step sequentially forward in either direction - i.e. $w_{k+1} - w_k \in \{(0, 1), (1, 0), (1, 1)\}$.

3. **Monotonicity condition** The warping path cannot skip backwards - i.e. $m_1 \leq m_2 \leq ... \leq m_K$ and $n_1 \leq n_2 \leq ... \leq n_K$. (This is, in fact, already implied by condition 2).

A graphic example of such an alignment is given in figure 2.11.

A DTW consists of two steps: Cost calculation and path detection. In the cost calculation steps, a local cost c is calculated for all pairs (x_m, y_n), resulting in a cost matrix C. An example is shown in figure 2.12. Common cost functions include the Manhattan distance and the cosine distance (i.e. the complement of the normalized inner product), which was used in this work:

$$c(x_m, y_n) = 1 - cos(\theta) \tag{2.10}$$

where θ is the angle between x_m and y_n.

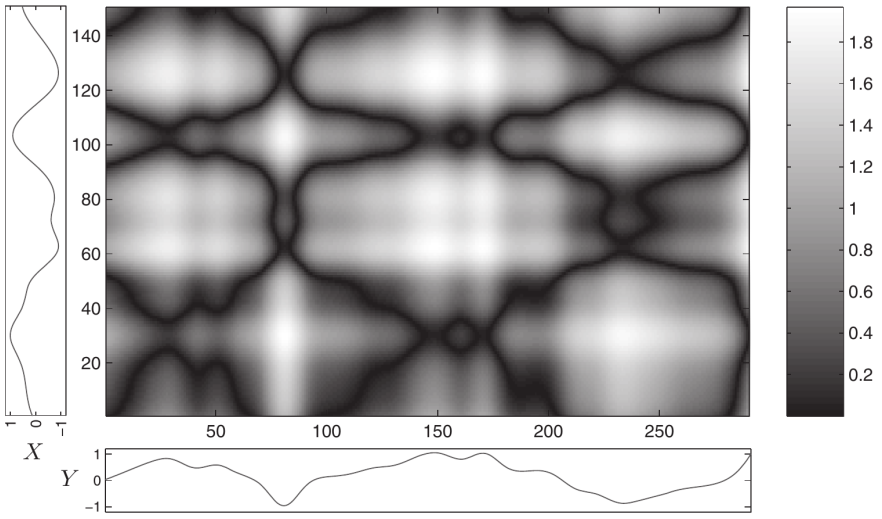


Figure 2.12: Example of a matrix of costs between two sequences X and Y using the Manhattan distance as the local cost measure. [2]

In the second step, an optimal warping path is calculated on the cost matrix. The cost of a warping path is

$$c_W(X,Y) = \sum_{k=1}^{K} c(x_{m_k}, y_{n_k}) \tag{2.11}$$

and the optimal warping path is the one with minimal cost - i.e. the DTW cost:

$$DTW(X,Y) = \min\{c_W(X,Y)|W \text{ is a warping path}\} \tag{2.12}$$

Consequently, the DTW cost can also be used to compare the quality of alignments of one query sequence to multiple other sequences when taking the varying lengths into account. The path calculation is commonly solved using a Dynamic Programming algorithm [2]. In this work, the implementation from [24] is used.

Subsequence DTW is a variant of this algorithm in which the boundary condition is loosened. This means that the optimal warping path can run along a subsequence of the longer compared sequence instead of the full series. This is more computationally expensive since more paths need to be calculated for comparison.

2.3.2 Levenshtein distance

The Levenshtein distance, also called edit distance, is a measure of similarity between two strings (or character sequences). The algorithm was first described by Levenshtein in 1965 [25] and can also be used to retrieve an optimal alignment (approximate string matching [26]). In that sense, it serves a similar purpose as DTW for strings instead of time sequences. This measure is commonly used in the fields of Computational Biology, Natural Language Processing, and signal processing.

The distance is the sum of character operations necessary to transform one string into the other. These operations can be substitutions, insertions, or deletions, which may be weighted differently. An example is shown in figure 2.13. If each operation has a cost of 1, the Levenshtein distance in this example is 5; if substitutions are weighted with a cost of 2, the distance is 8.

Just like DTW, this problem is usually solved efficiently with a Dynamic Programming approach. For two strings $X = (x_1, x_2, ..., x_M)$ and $Y = (y_1, y_2, ..., y_N)$, the initial step is then defined as

$$L(0,0) = 0, L(i,0) = \sum_{k=1}^{i} I(x_k), L(0,j) = \sum_{k=1}^{j} D(y_k) \tag{2.13}$$

and the recursive step is defined as

$$L(i,j) = \min \begin{cases} L(i-1,j) + D(y_j) \\ L(i,j-1) + I(x_i) \\ L(i-1,j-1) + S(x_i, y_j) \end{cases} \tag{2.14}$$

where $L(i,j)$ is the Levenshtein distance at step (i,j) and D, I, and S are the costs for deletions, insertions, and substitutions respectively [27]. The over-all Levenshtein distance is $L(M,N)$.

2.4 Machine learning algorithms

This section describes the various Machine Learning algorithms employed throughout this thesis. Gaussian Mixture Models (GMMs), Hidden Markov Models (HMMs), and Support Vector Machines (SVMs) are three traditional approaches that are used as the

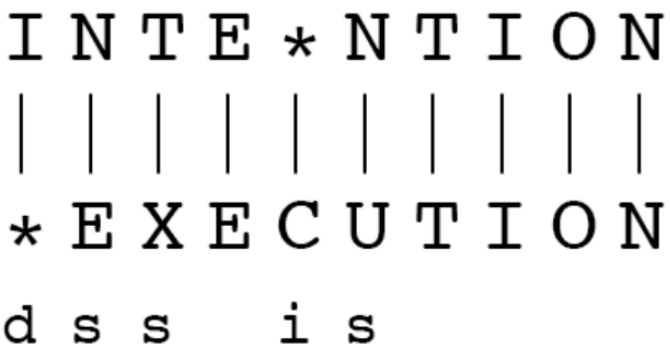


Figure 2.13: Example of a Levenshtein distance calculation between the two strings "INTENTION" and "EXECUTION". Found operations (**d**eletions, **i**nsertions, **s**ubstitutions) are shown at the bottom. [28]

basis of many new approaches, and were used for several starting experiments. i-Vector processing is a relatively new, more sophisticated approach that bundles several other machine learning techniques.
In recent years, Deep Learning has become the standard for machine learning applications []. This chapter also describes a new approach that was used extensively in this work: Deep Neural Networks (DNNs).

2.4.1 Gaussian Mixture Models

In their basic form, Gaussian Mixture Models (GMMs) are a form of unsupervised learning. Given a set of observations $X = (x_1, x_2, ..., x_N)$, their probability distribution is modeled with a superposition of Gaussian distributions:

$$p(x_n|\lambda) = \sum_{i=1}^{M} p_i b_i(x) \tag{2.15}$$

where b_i are the constituting distributions, p_i are the mixture weights, and λ are the model parameters. A visualization is shown in figure 2.14. If the observations are multidimensional (which is the case for audio features), multivariate Gaussians (with D dimensions) are used for this:

$$b_i(x) = \frac{1}{(2\pi)^{D/2} \det(\Sigma_i)^{1/2}} \exp(\left\{ -\frac{1}{2}(x - \mu_i)\Sigma_i^{-1}(x - \mu_i) \right\}) \tag{2.16}$$

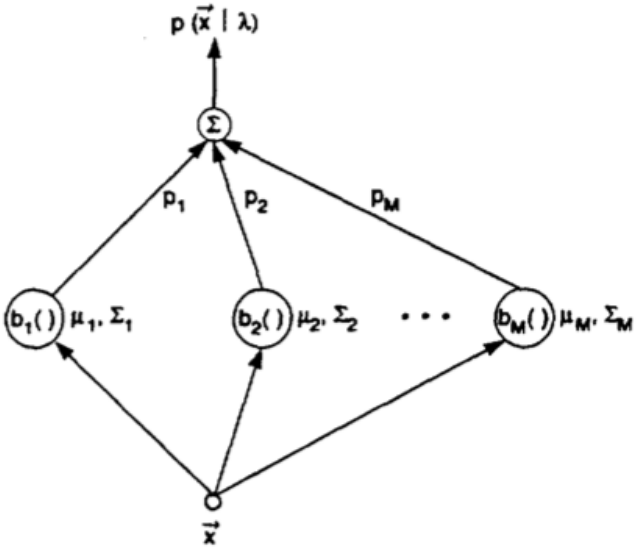


Figure 2.14: Visualization of a GMM. The Gaussian mixture is the weighted sum of several Gaussian distributions, where p_i are the mixture weights and b_i are the Gaussians. [30]

where μ_i is the mean vector and Σ_i is the covariance matrix.
These parameters, together with the mixture weights, define the model:

$$\lambda = p_i, \mu_i, \Sigma_i, i = 1, ..., M \tag{2.17}$$

For each observation o_j, the contribution of each Gaussian b_i can be calculated as:

$$p_n i = P(i|n) = \frac{b_i P(i)}{P(x_n)} \tag{2.18}$$

The overall likelihood of the model is:

$$L = \prod_{n=1}^{N} P(x_n) \tag{2.19}$$

In order to find the optimal parameters λ, the iterative Expectation Maximization (EM) algorithm is commonly used [29]. In the expectation step, L is calculated; in the maximization step, the parameters are adapted. This is repeated until convergence. An example of such a training procedure is visualized in figure 2.15.

Gaussian Mixture Models have been used in many areas of machine learning, for example in ASR [30], in image retrieval [32], in financial modeling [33], and in visual

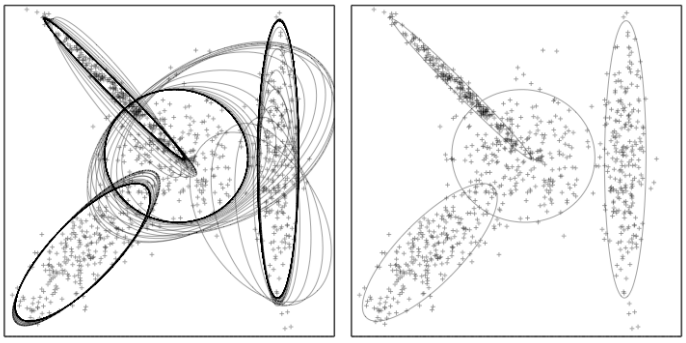


Figure 2.15: Example of a GMM in two dimensions. Crosses signify the data points, ellipses the three multivariate Gaussians. On the left-hand side, the evolution of the estimated means and covariances of these Gaussians during the EM process is shown; the right-hand side shows the converged mixture. [31]

tracking [34]. When used for classification, one GMM is trained for each class separately $S = (s_1, s_2, ..., s_J)$, resulting in J sets of parameters λ. The likelihood L_j of each class is then determined, and the most likely class is chosen:

$$C = \underset{1 \leq j \leq J}{\operatorname{argmax}} P(\lambda_j|X) = \underset{1 \leq j \leq J}{\operatorname{argmax}} \frac{p(X|\lambda_j)P(\lambda_j)}{p(X)} \tag{2.20}$$

(according to Bayes' rule). If all classes and all observations are equally likely, this simplifies to

$$C = \underset{1 \leq j \leq J}{\operatorname{argmax}} p(X|\lambda_j) \tag{2.21}$$

In practice, log-probabilities are commonly used for numerical reasons, resulting in the calculation:

$$C = \underset{1 \leq j \leq J}{\operatorname{argmax}} \sum_{n=1}^{N} p(x_n|\lambda_j) \tag{2.22}$$

In addition to this direct classification concept, GMMs are often used to model the emission probabilities in Hidden Markov Models.

2.4.2 Hidden Markov Models

Markov models are statistical models of Markov processes - i.e. sequences of states in which the probability of each state only depends on the previous one. In a Hidden Markov Model (HMM), these states are not directly observable, but may be inferred from the models emissions. HMMs were first suggested by Baum et al. around 1970

[35][36][37][38][39]. Due to their ability to model temporal processes, they have been employed extensively in ASR [40][41][42][43]. Apart from this field, they are also frequently used in Natural Language Processing [44], Optical Character Recognition [45], and Computational Biology [46][47].

A HMM consists of four basic components:

- The observation (= emission) sequence $O = (o_1, o_2, ..., o_T)$
- The hidden state nodes $Q = (q_1, q_2, ..., q_N)$; the sequence of hidden states corresponding to the observation sequence will be denoted as $I = (i_1, i_2, ..., i_T)$ where $i_t \in Z$.
- The transition probabilities between the hidden states, defined by a transition matrix $A \in \mathbb{R}^{NxN}$; additionally, the initial state distribution (i.e. the probability of starting in each state) $\pi \in \mathbb{R}^N$
- The emission probabilities $B = (b_1(k), b_2(k), ..., b_N(k))$, mapping from the hidden states to the observations. These can, for example, be Gaussians for continuous outputs o_t, or conditional probabilities for discrete o_t.

The transition probabilities, initial state distribution, and emission probabilities define the model λ.

In the case of speech recognition, the observations are the feature vectors, and the hidden states are the phonemes generating these features. Such a model is visualized in figure 2.16. Different variants of HMMs can be created by restricting the transition matrix in certain ways; e.g., left-to-right HMMs, which only allow transitions to subsequent states and are often used in speech recognition and handwriting recognition [48]. A particularly interesting property of HMMs for speech recognition is their relative invariance to warping along the time axis since states can usually be repeated for arbitrary amounts of time.

Three problems commonly need to be solved for problems modeled with HMMs:

Evaluation - i.e., how probable is an observation sequence given this model? In mathematical terms, the probability $P(O, I|\lambda)$ is sought. The most straightforward way to do this would be to calculate this probability for each possible Y of the

length T of the observation sequence, but this is very computationally expensive. For this reason, an algorithm called *forward procedure* is used. A forward variable α representing the probability at time t is introduced:

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, i_t = q_i|\lambda) \tag{2.23}$$

This can be solved inductively with the initialization

$$\alpha_1(i) = \pi_i b_i(O_1), 1 \leq i \leq N \tag{2.24}$$

and the induction step

$$\alpha_{t+1}(j) = \left[\sum_{i=1}^{N} \alpha_t(i) a_i j\right] b_j(O_{t+1}) \tag{2.25}$$

This is a form of Dynamic Programming.

Training - i.e., how can the parameters λ be set optimally to maximize the probability of observed sequences? There is no analytical way to compute this, but the so-called *Baum-Welch* algorithm (which is a special case of the Expectation Maximization algorithm) allows for an iterative estimation of the parameters. In order to do this, a backward variable β is calculated analogous to α to represent the probability of the sequence from time $t + 1$ to the end:

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T, i_t = q_i|\lambda) \tag{2.26}$$

$$\beta_T(i) = 1, 1 \leq i \leq N \tag{2.27}$$

$$\beta_t(i) = \sum_{j=1}^{N} a_i j b_j(O_{t+1}) \beta_{t+1}(j) \tag{2.28}$$

The probability of a path being in state q_i at t and making a transition to q_j at $t + 1$ is then:

$$\xi_t(i, j) = \frac{\alpha_t(i) a_i j b_j(O_{t+1} \beta_{t+1}(j)}{P(O|\lambda)} \tag{2.29}$$

This can be used to calculate the expected numbers of transitions and emissions, which can be re-adapted with statistics from the observation sequences and used to adjust α and β. This process is repeated until convergence.

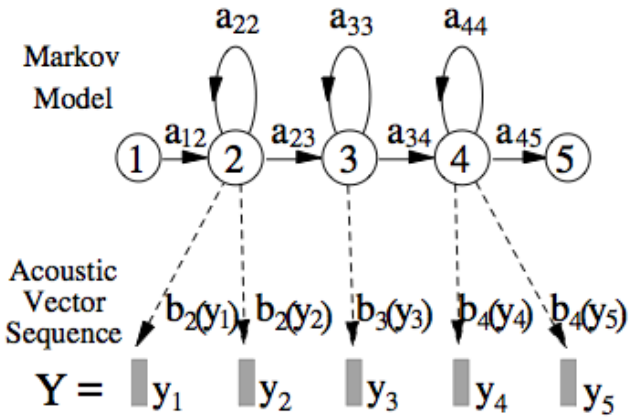


Figure 2.16: A HMM as it is commonly used in ASR, with phonemes as hidden states and acoustic feature vectors as the observations. $a_12, a_22, a_23, ...$ are elements of the transition matrix A; $b_2(y1_), b_2(y_2), b_3(y_3)$ are elements of the output probability matrix B. [43]

Decoding - i.e., given an observation sequence, what is the most probable underlying sequence of hidden states? This is particularly interesting for speech recognition since the interpretation here is the detection of the generating phonemes from a series of feature vectors.

Again, this problem is broken down by first defining a variable for the probability of being in state q_i at time t:

$$\gamma_t(i) = P(i_t = q_i|O, \lambda) \tag{2.30}$$

and therefore, the most likely state at t is:

$$i_t = \underset{1 \leq i \leq N}{\operatorname{argmax}}[\gamma_t(i)], 1 \leq t \leq T \tag{2.31}$$

Using the forward and backward variables, this can be expressed as

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\ \lambda)} \tag{2.32}$$

This problem can be solved efficiently with the *Viterbi algorithm*, which again employs Dynamic Programming.

2.4.3 Support Vector Machines

In all supervised classification tasks, a training data set and a classification data set are provided. Each of these sets consists of previously extracted features of the training entities (or musical pieces in the case of genre classification) and, for the training data set, annotation labels (or genres in the case of genre classification). The training data set is denoted here as pairs of $(\mathbf{x}_i, y_i), i = 1..l$, with $\mathbf{x}_i \in \mathbb{R}^n$ being the feature vectors and y_i being their assigned classes. The classifier tries to find similarities behind the training data for each label and separation conditions between the labels. With this information, the classifier is then able to assign a label to the classification data entities.

SVMs attempt to solve this problem by grouping the training data vectors $\mathbf{x}_i$ and finding separating (hyper-)planes (with the normal vector $\mathbf{w}$ and the offset b) between the points of the different classes (or annotation labels) y_i. In doing so, they try to maximize the margin between the plane and the data points. Additionally, the feature vectors may be transformed into a higher-dimensional space by the function $\phi(\mathbf{x}_i)$ to make them more easily separable (as demonstrated in figure 2.17).

In [49], this training process is expressed (for a two-class problem) as:

$$\begin{aligned} \min_{\mathbf{w},b,\xi} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y_i(\mathbf{w}^T\phi(\mathbf{x}_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0 \end{aligned}$$

(with ξ_i being a slack variable and $C > 0$ being a penalty parameter for the error term, higher Cs allowing for fewer outliers ([50]).

$K(\mathbf{x}_i, \mathbf{x}_j) \equiv \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$ is called the *kernel function*. Several variants are possible, e.g. a linear one:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{x}_j \tag{2.33}$$

It is often useful to use a non-linear kernel because the data points may not be linearly separable (even after the transformation into a higher-dimensional space). An example is shown in figure 2.18. The RBF kernel (Radial Basis Function) is a popular one:

$$K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\gamma\|\mathbf{x}_i - \mathbf{x}_j\|^2}, \gamma > 0 \tag{2.34}$$

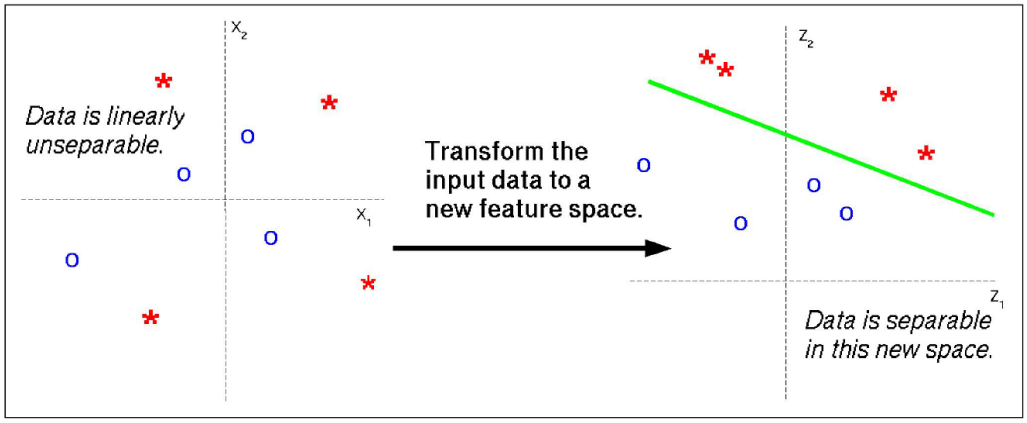


Figure 2.17: A set of data points which cannot be separated linearly in their original form (left), but can be separated after transformation into another space (right) [50]

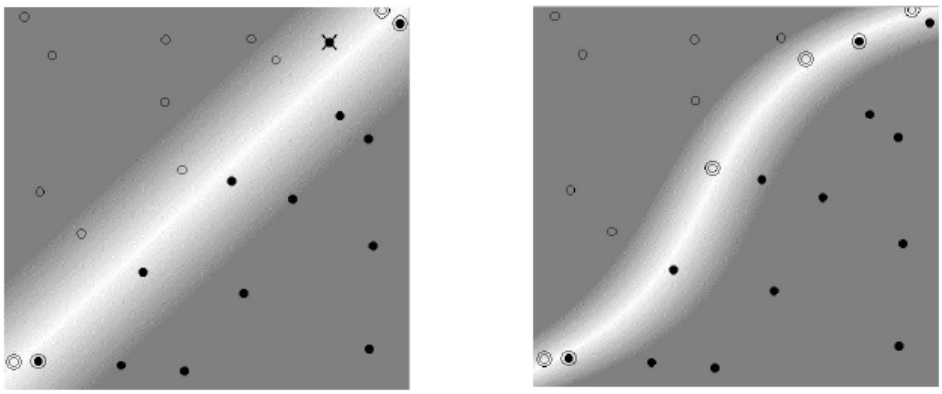


Figure 2.18: A set of data points which cannot be separated using a linear kernel (left), but can be separated with a polynomial kernel (right) [51]

As can be seen from the above equations, C and γ are free parameters. Their optimum values depend on the actual training data vectors. In [49], a grid search during each training is suggested to find them.

The presented training process is appropriate for a two-class problem. For multi-class problems, one-vs-one trainings for all combinations of classes are performed. Then, all of the developed classifiers are used for the classification of the evaluation data and a voting strategy is applied to determine the resulting class. This procedure is not quite in agreement with the one used in the feature selection algorithm (IRMFSP, see

section **??**), as this algorithm selects features to separate all classes at once, rather than on a one-vs-one basis.

2.4.4 i-Vector processing

I-Vector (identity vector) extraction was first introduced in [52] and has since become a state-of-the-art technique for various speech processing tasks, such as speaker verification, speaker recognition, and language identification [53]. To our knowledge, it has not been used for any Music Information Retrieval tasks before.
The main idea behind i-vectors is that all training utterances contain some common trends, which effectively add irrelevance to the data in respect to training. Using i-vector extraction, this irrelevance can be filtered out, while only the unique parts of the data relevant to the task at hand remain. The dimensionality of the training data is massively reduced, which also makes the training less computationally expensive. As a side effect, all feature matrices are transformed to i-vectors of equal length, eliminating problems that are caused by varying utterance lengths.
Mathematically, this assumption can be expressed as:

$$M(u) = m + Tw \tag{2.35}$$

In this equation, $M(u)$ is the GMM supervector for utterance u. The supervector approach was first presented in [54] and has since been successfully applied to a number of speech recognition problems. A music example can be found in [55]. m represents the language- and channel-independent component of u and is estimated using a Universal Background Model (UBM). T is a low-rank matrix modeling the relevant language- and channel-related variability, the so-called Total Variability Matrix. Finally, w is a normally distributed latent variable vector: The i-vector for utterance u.

Step 1: UBM training A Universal Background Model (UBM) is trained using Gaussian Mixture Models (GMMs) from all utterances. This UBM models the characteristics that are common to all of them.

Step 2: Statistics extraction 0th and 1st order Baum-Welch statistics are calculated for each of the utterances from the UBM according to:

$$N_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega) \tag{2.36}$$

$$\widetilde{F}_c(u) = \sum_{t=1}^{L} P(c|y_t, \Omega)(y_t - m_c) \tag{2.37}$$

where $u = y_1, y_2, ..., y_L$ denotes an utterance with L frames, $c = 1, ..., C$ denotes the index of the Gaussian component, Ω denotes the UBM, m_c is the mean of the UBM mixture component c, and $P(c|y_t, \Omega)$ denotes the posterior probability that the frame y_t was generated by mixture component c. As the equation shows, the 1st order statistics are centered around the mean of each mixture component.

Step 3: T matrix training Using the Baum-Welch statistics for all utterances, the Total Variability Matrix T is now trained iteratively according to:

$$w = (I + T^t \Sigma^{-1} N(u) T)^{-1} T^t \Sigma^{-1} \widetilde{F}(u) \tag{2.38}$$

using Expectation Maximization.

Step 4: Actual i-vector extraction Finally, an i-vector w can be extracted for each utterance using equation 2.38 again. This can also be done for unseen utterances, using a previously trained T.

2.4.5 Artificial Neural Networks

Artificial Neural Networks have a long research history. Based on an idea by McCulloch and Pitts from 1943 [56], they were slowly developed into functional algorithms for classification and pattern recognition. Rosenblatt proposed a hardware design for a single-layer perceptron in 1958 [57], while the first multi-layer networks were introduced by Ivakhnenko and Lapa in 1965 [58]. In 1969, Minsky and Papert posited many practical limitations for Neural Networks [59], which led to a decrease in interest.
The introduction of the backpropagation algorithm solved some of these issues and increased the training speed of multilayer networks [60], leading to a wider usage in speech recognition [61] and other fields, such as computer vision [62] and Natural Language Processing [63]. However, other algorithms such as SVMs began to produce better results over time and thus overtook Neural Networks in popularity.
Over time, processing speed of computers increased, and better strategies and hardware for parallel computing became available. This enabled the training of networks with many more layers, allowing for a much better adaptation to many high-dimensional

problems [64]. Over the past 10 years, this so-called "deep learning" became the state of the art for many machine learning problems [65][66][67].

Artifical Neural Networks (ANNs) are inspired by "real" Neural Networks - i.e. the human brain and nervous system. They generally consist of neurons, which are nodes that can process inputs and send outputs, and the connections between them. These neurons are grouped in layers: An input layer, an output layer, and a number of hidden layers in between them. Historically, ANNs first had no hidden layers at all; this type of ANN was called "perceptron". Later, the addition of a hidden layer was introduced and the resulting networks were called "Multilayer Perceptrons" (MLPs). Networks with no hidden layers are only able to solve linear problems; the introduction of hidden layers added non-linear projections to the calculation. Recent "Deep" Neural Networks (DNNs) possess three or more hidden layers, leading to an exponential increase of the degrees of freedom and thus the possibility to model much more complex problems.

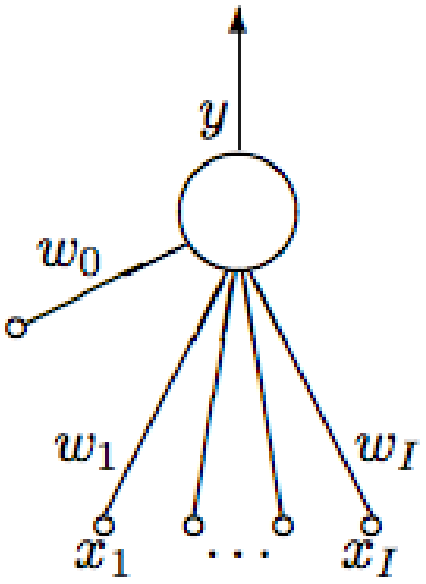
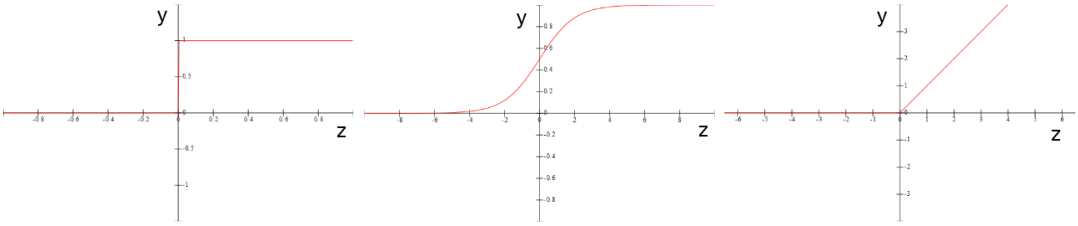


Figure 2.19: Functionality of a single neuron in an ANN. The neuron computes a weighted sum of its inputs, and then applies an activation function, resulting in output y. [68]

The function of a single neuron is visualized in figure 2.19. Classical neurons compute



(a) Heaviside binary step activation function (b) Sigmoid activation function (c) ReLU activation function

Figure 2.20: Activation functions used for ANN neurons. [69]

a weighted sum z of their inputs $\mathbf{x} = (x_1, x_2, ..., x_I)$:

$$z = w_0 + \sum_{i=1}^{I} x_i w_i = w_0 + \mathbf{w}^T \mathbf{x} \tag{2.39}$$

where $\mathbf{w} = (w_1, w_2, ..., w_I)$ is a vector of weights, and w_0 is a constant bias. This result is often called the "logit". Then, a nonlinear activation function can be applied. In perceptrons, this was the Heaviside step function, shown in figure 2.20a, resulting in a binary output:

$$y = \begin{cases} 1 \text{ if } z \leq 0 \\ 0 \text{ otherwise} \end{cases} \tag{2.40}$$

An activation function commonly used is the sigmoid function, shown in figure 2.20b:

$$y = \frac{1}{1 + e^{-z}} \tag{2.41}$$

Neurons of this type are called "logistic" neurons. This function is often applied because it generates a smooth, real-valued output and has easy-to-use derivatives, simplifying the training process.
Rectified linear units (ReLUs) are also used frequently since they train faster than logistic units and retain more information that is relevant in the middle layers (in particular, sparsity for small inputs and lower risk of vanishing or exploding gradients). The function is shown in figure 2.20c:

$$y = \max(0, z) \tag{2.42}$$

In the last layer, a so-called softmax activation function is often applied. This function takes the outputs of all neurons into account, and computes a probability distribution (i.e. all the outputs will sum up to one and represent the likelihood of the corresponding

class):

$$y_j = \frac{e^{z_j}}{\sum_{k=1}^{M} e^{z_k}}, 1 \leq j \leq M \tag{2.43}$$

where M is the number of output neurons.

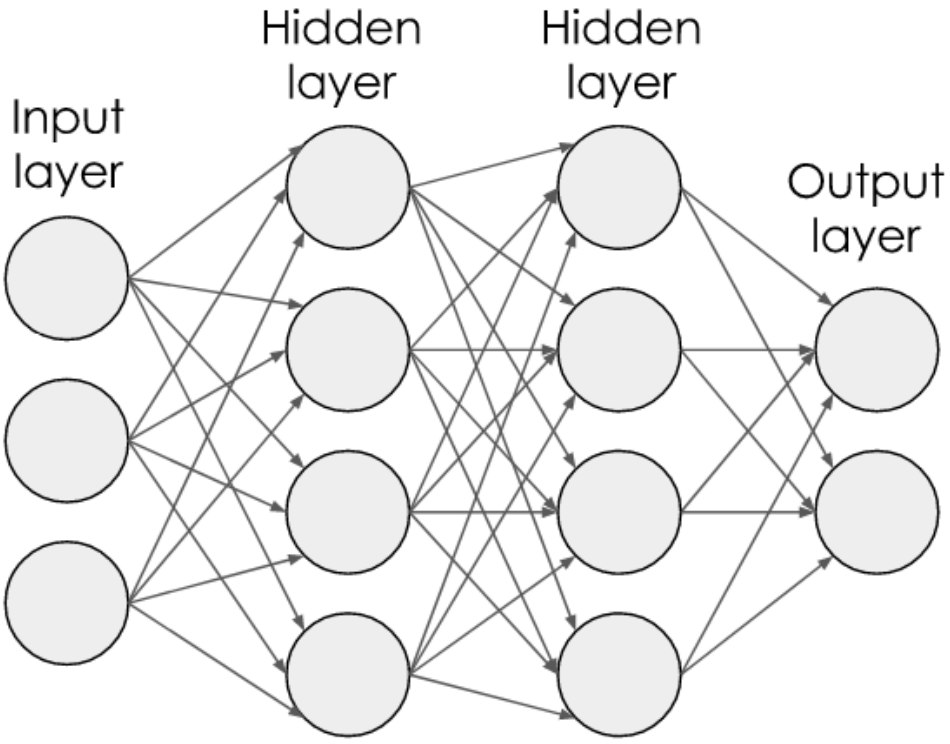


Figure 2.21: Schematic of a feed-forward neural network with an input layer with four neurons, two hidden layers with four neurons each, and an output layer with two neurons. [69]

In addition to the various types of neurons, ANNs themselves can be grouped into a number of types. In their basic configuration, ANNs will have multiple layers of the described neurons where connections are only allowed in one direction. A schematic is shown in figure 2.21. This type of network is called "feed forward" or, in the context of Deep Learning, a Deep Neural Network (DNN).
If connections that skip back are allowed in the network, the network is called a Recursive Neural Network (RNN). These networks are particularly useful for modeling time-dependent series because they have the ability to "store" temporal states in their units. However, they were deemed impractical for a long time because they exhibit the exploding/vanishing gradient problem during training [70]. This problem was solved with the introduction of memory cells in place of neurons, in particular Long Short-

Term Memory (LSTM) units [71] and Gated Recurrent Units (GRU) [72]. Nowadays, RNNs are being used successfully in a number of research tasks [**?**].
A third type of Neural Network are Convolutional Neural Networks (CNNs). These networks add layers of filters before or in between classical fully-connected layers. The parameters of these filters are trained jointly with the other layers. For this reason, CNNs are able to "learn" a feature representation of the input. They were first used in image recognition [**?**], but are now also being used for audio-related tasks such as environmental sound classification [**?**] and ASR [**?**]. A disadvantage of both RNNs and CNNs is the computational complexity of training them, and the requirement for even more training data because they possess even more degrees of freedom than DNNs of comparable sizes.
In this work, DNNs with logistic units in the hidden layers and a softmax output layer were used in phoneme classification tasks.

Neural Network training is performed via the backpropagation algorithm. This algorithm is based on the calculation of a cost (or error) function E, which computes the difference between the network output and the expected output (e.g. the annotated classes in the training data). Then, the partial derivatives of the cost are calculated with regards to the weights, using the outputs and logits for the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial z_j} \frac{\partial z_j}{\partial w_{ij}} \tag{2.44}$$

Then, the weights are adjusted accordingly:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}} \tag{2.45}$$

where η is the so-called learning rate which must be chosen carefully to ensure convergence of the training process.
In the same way, the error is propagated further backwards through the model for the weights from the previous layers. This is done until all the weights have been adjusted. Then, the next batch of training data is passed through the model and the process is repeated. The training process is performed until the weights converge (or until a fixed number of iterations has been reached).

A commonly used cost function is the squared error measure:

$$E_S = \frac{1}{2}\sum_{j=1}^{M}(t_j - y_j)^2 \tag{2.46}$$

where t_j is the target value for output unit j.

Alternatively, the cross-entropy is often chosen as the cost function when using a softmax output layer:

$$E_C = -\sum_{j=1}^{M} t_j \log y_j \tag{2.47}$$

One of the major issues in Neural Networks is that of overfitting: The model adapts too well to the training data and is not able to sufficiently generalize to new data anymore. (In extreme cases, the model simply learns all the training data by heart). This is especially relevant for deep networks because of the many degrees of freedom. There are several strategies to overcome this problem. One of the most frequently used regularization strategies is dropout: During training, nodes in the network are deactivated randomly, effectively resulting in many different network structures and making the whole network robust to variations in the data. The main sticking point, however, is the amount of data used for training the network. The more variety is available, the lower the risk of overtraining.
The so-called "curse of dimensionality" is a concept that applies to many machine learning algorithms: However, Goodfellow makes an argument that Deep models operate on a different level altogether [73]: Where traditional machine learning methods make an assumption of smoothness of the hidden functions to be represented, Deep models actually attempt to model the underlying structures in a distributed way. This means that information about outputs can be learned in a shared way - i.e., if two classes have something in common, the model is also able to represent this fact. Practically, the many degrees of freedom do not lead to "learning by heart", but instead allow for an internal representation of highly complex relationships.

2.5 Speech recognition systems and their evaluation

2.5.1 Phoneme recognition

In common ASR systems, phoneme recognition is performed by training an acoustic model that is able to determine the probability of phoneme occurrences in unseen au-

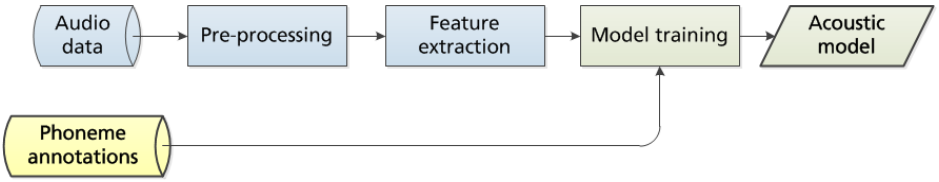


Figure 2.22: Schematic of the training procedure for phoneme recognition.

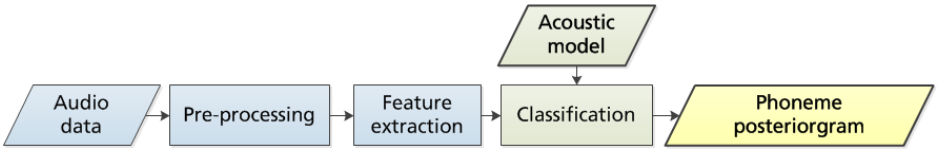


Figure 2.23: Schematic of the classification procedure for phoneme recognition.

dio frames. The result of this is a so-called phoneme posteriorgram - i.e. a matrix of the probabilities of each possible phoneme over time. In a second step, another model is applied to this posteriorgram, taking into account the probabilities of phoneme sequences and words formed by these phonemes, and thus generating a phoneme string from the posterior probabilities. This step is called language modeling.

Language modeling is a field of research unto itself, and is a complex tasks for lyrics. For this reason, this work focuses on improving the acoustic modeling step. The training process is performed according to the general schema described in section 2.1. The specific adaptations are shown in figures 2.22 and 2.23: In training, phoneme annotations are used to defined the possible classes. A set of around 39 phonemes is common; the phoneme set used in this work is the one used in CMU Sphinx[1] in the *TIMIT* annotations. The set is listed with examples in **??**. The used data sets contained annotations of monophones. In ASR, splitting phonemes into three temporal phases is common, resulting in so-called triphones. This was also tested as described in section 5.3.

The whole training process then results in an acoustic model, which can be used for classification as shown in figure 2.23. As described, classification results in a phoneme posteriorgram. An example is shown in figure 2.24. The time resolution of the posteriograms in this work is $10ms$. Instead of language modeling, many of the following tasks build directly on the posteriogram. For evaluation of the phoneme recognition

[1] `http://cmusphinx.sourceforge.net/`

itself, Viterbi decoding with evenly weighted transitions is performed on the posteriorgrams. Then, the Phoneme Error Rate (PER) and the Weighted Phoneme Error Rate (WPER) are used for assessing the quality. The Phoneme Error Rate is simply the Levenshtein distance between the generated and the expected phoneme sequence where insertions, deletions, and replacements are weighted equally, normalized by the length of the expected sequence:

$$PER = \frac{D + I + S}{N} \tag{2.48}$$

where D are deletions, I are insertions, and S are substitutions of phonemes and N is the length of the sequence. (The accuracy measure used by in some of the state-of-the-art works is the same as $1 - PER$).
Some work also use a measure called *correct*, which ignores insertions. This makes sense if it is assumed that the phoneme results are used afterwards by an algorithm that is tolerant to insertions. In many cases, they will then also be tolerant to deletions. For cases like this, Hunt suggested a weighted error rate that punishes insertions and deletions less heavily than substitutions [74]:

$$Weighted\ PER = \frac{0.5D + 0.5I + S}{N} \tag{2.49}$$

2.5.2 Language identification

Language identification has been extensively researched in the field of Automatic Speech Recognition since the 1980's. A number of successful algorithms have been developed over the years. An overview over the fundamental techniques is given by Zissman in [75].
Fundamentally, four properties of languages can be used to discriminate between them:

Phonetics The unique sounds that are used in a given language.

Phonotactics The probabilities of certain phonemes and phoneme sequences.

Prosody The "melody" of the spoken language.

Vocabulary The possible words made up by the phonemes and the probabilities of certain combinations of words.

Even modern system mostly focus on phonetics and phonotactics as the distinguishing factors between languages. Vocabulary is sometimes exploited in the shape of language

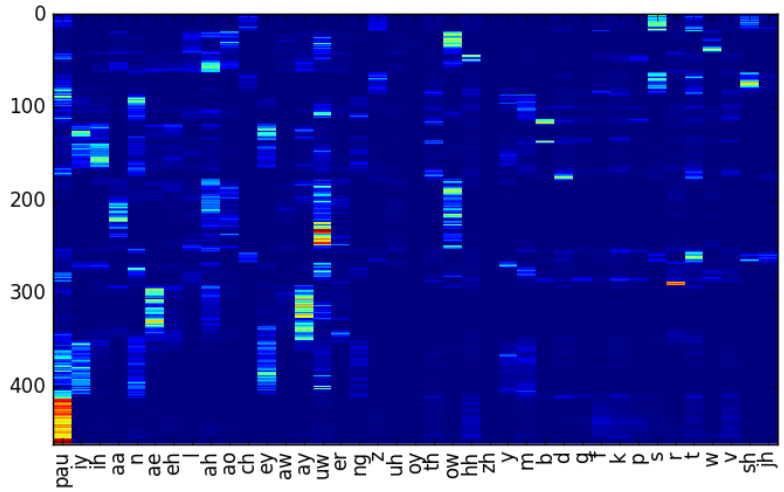


Figure 2.24: Example of a phoneme posteriorgram, the result of classification with an acoustic model. The posteriorgram represents the posterior probabilities of each phoneme over time. The time resolution in this example is $10ms$.

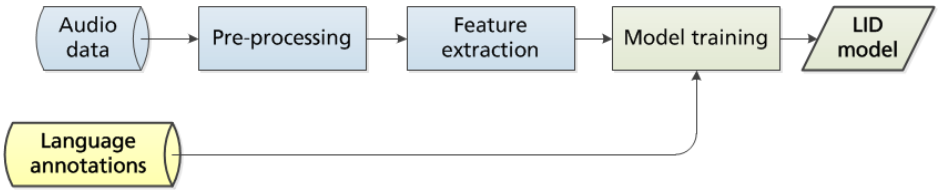


Figure 2.25: Schematic of the training procedure for language identification.

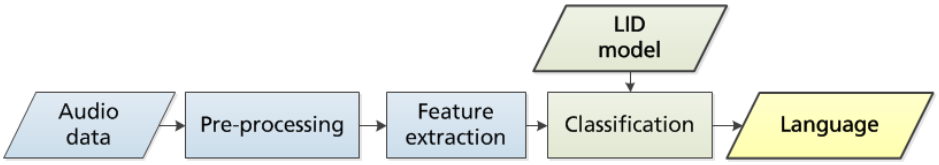


Figure 2.26: Schematic of the classification procedure for language identification.

models.
In ASR, the standard technique for language identification is Parallel Phone Recognition followed by Language Modeling (PPRLM). In this approach, acoustic and language models are trained for each language (or, in some cases, only the language models are different and just one acoustic model is used) . Unseen examples are then run through each model or combinations thereof, and the result with the highest likelihood determines the language.
Other approaches directly train models for each language (e.g. GMMs). This technique can be considered a "bag of frames" approach, i.e. the single data frames are considered to be statistically independent of each other. The generated models then describe probability densities for certain acoustic characteristics of each language. GMM approaches used to perform worse than their PPRLM counterparts, but the development of new features has made the difference negligible [76]. They are in general easier to implement since only audio examples and their language annotations are required. Allen et al. [10] report results of up to 76.4% accuracy for ten languages. Different backend classifiers, such as Multi-Layer Perceptrons (MLPs) and Support Vector Machines (SVMs) [9] have also been used successfully instead of GMMs.

In this work, an approach that trains directly on the acoustic characteristics using i-vector extraction and SVMs is presented. The modifications to the general processing chain are presented in figures 2.25 and 2.26.
Additionally, a second approach based on phoneme posteriorgrams is tested. Statistics from the posteriorgrams are calculated, and then a second model is trained on these.

For evaluation, all test examples are classified into exactly one language class. Then, the accuracy (i.e. the average retrieval) is calculated:

$$Accuracy = \frac{TP}{N} \tag{2.50}$$

where TP are the True Positives, and N is the number of all documents. In ASR, the average cost measure as recommended in [77] is also used widely now; however, for comparison with other singing language identification approaches such as those described in chapter 3, the accuracy was still used in this work.

2.5.3 Keyword spotting

In [78], three basic principles for keyword spotting in speech are mentioned:

LVCSR-based keyword spotting In this approach, a full transcription of the audio recording is performed using Large-Vocabulary Continuous Speech Recognition (LVCSR), which can then be searched for the keyword. This is expensive to implement and offers no tolerance for transcription errors - if the keyword is not transcribed correctly, it will never be found later.

Acoustic keyword spotting Acoustic KWS algorithms only search for the keyword on the basis of its acoustic properties. This approach is easy to implement and provides some tolerance for pronunciation variations. However, it does not take any a-priori knowledge about the language into account (e.g. about plausible word or phoneme sequences).

Phonetic search keyword spotting Again, a full transcription of the audio recording is performed, but the full lattices are retained instead of just the final transcription. A phonetic search for the keyword can then be run on these lattices. This approach combines the a-priori knowledge of the LVCSR-based approach with the robustness of the acoustic approach.

As described in the next chapter (section 3.1), there are significant differences between speech and singing signals, which means that ASR approaches for keyword spotting cannot simply be transferred to singing. In particular, both LVCSR-based keyword spotting and Phonetic search keyword spotting depend heavily on predictable phoneme durations (within certain limits). When a certain word is pronounced, its phonemes will usually have approximately the same duration across speakers. The language model employed in both approaches will take this information into account. However, phoneme durations in singing are not as predictable in speech, as figure 3.1 demonstrates. For this reason, a simpler acoustic approach using keyword-filler HMMs is employed in this work.
Keyword-filler HMMs have been described in [79] and [80]. In general, two separate HMMs are created: One for the requested keyword, and one for all non-keyword regions (=filler). The keyword HMM is generated using a simple left-to-right topology with one state per keyword phoneme, while the filler HMM is a fully connected loop of states for all phonemes. These two HMMs are then joined. Using this composite HMM, Viterbi decoding is performed on the phoneme posteriorgrams. Whenever the Viterbi path passes through the keyword HMM, the keyword is detected. The likelihood of this path can then be compared to an alternative path through the filler HMM, resulting in a detection score. A threshold can be employed to only return highly scored occurrences. Additionally, the parameter β can be tuned to adjust the model. It determines

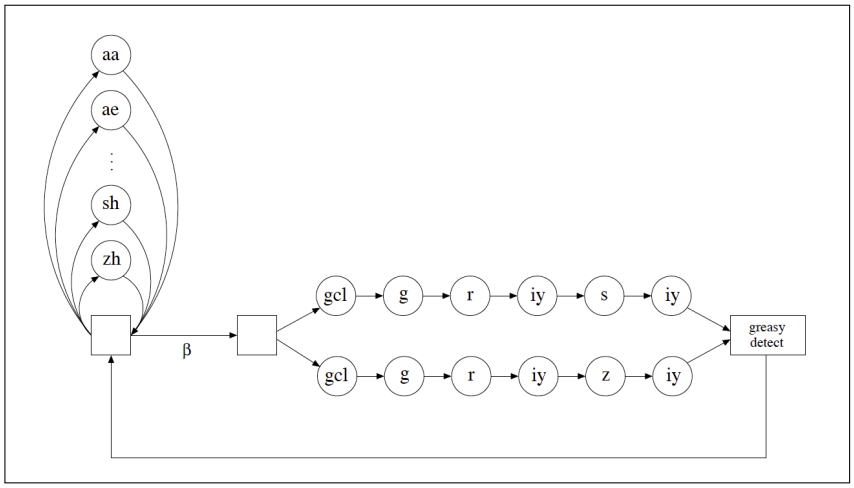


Figure 2.27: Keyword-filler HMM for the keyword "greasy" with filler path on the left hand side and two possible keyword pronunciation paths on the right hand side. The parameter β determines the transition probability between the filler HMM and the keyword HMM. [80]

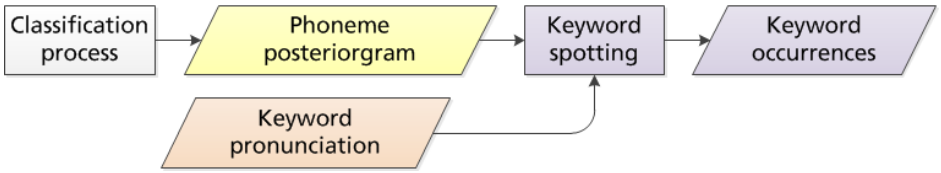


Figure 2.28: Schematic of the procedure for keyword spotting.

the likelihood of transitioning from the filler HMM to the keyword HMM. The whole process is illustrated in figure 2.27.

Integration with the phoneme recognition system is shown in figure 2.28. Effectively, the keyword-filler HMM is added as a post-processing step after classification, and thus performed on the posteriorgrams.

Keyword spotting results are considered correct when they are detected within the expected songs, and are evaluated according to the F_1 measure. This measure is the harmonic mean of precision P and recall R:

$$F_1 = 2\frac{P \cdot R}{P + R} \tag{2.51}$$

This measure is especially suited for cases where the classes are not balanced; in keyword spotting, occurrence of a keyword is much rarer than non-occurrence. In continuous speech recognition, the Figure Of Merit measure is often used [81]; however,

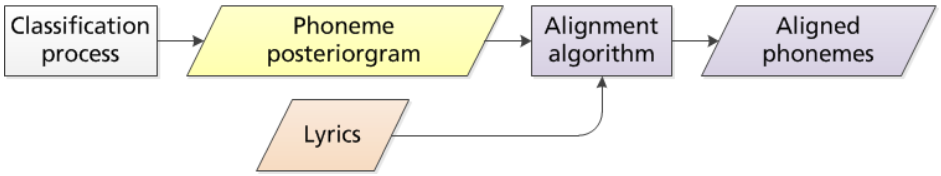


Figure 2.29: Schematic of the procedure for lyrics-to-audio alignment.

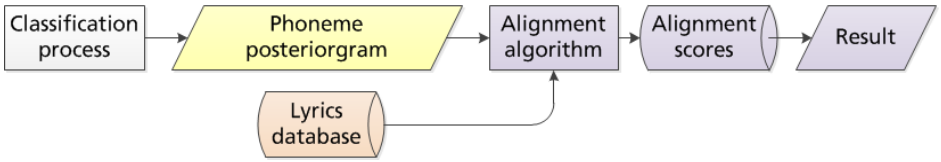


Figure 2.30: Schematic of the procedure for lyrics retrieval.

since timing is not an issue in many applications for sung keyword spotting, this measure is not used here.

2.5.4 Alignment and retrieval

Alignment is also performed on the phoneme posteriorgram. The lyrics with their phonetic pronunciation must be known; then, an algorithm finds the best positions of each phoneme in the sequence in the posteriorgram. Traditionally, Viterbi decoding is used for this, with the transition matrix shaped such that only transitions through the expected phonemes are possible. In this work, two other methods (based on DTW and the Levenshtein distance) are also tested. The general process is shown in figure 2.29.

For evaluation, the alignment approaches were submitted to the *MIREX* 2017 challenge for lyrics-to-audio alignment[2]. In this challenge, the mean absolute error in seconds was used as the evaluation measure. For its calculation, all absolute deviations between the expected phoneme start and end timestamps and the automatically detected ones are calculated, averaged over the whole song, and once again over all songs. The measure was suggested in [82].

Lyrics retrieval can be interpreted as an alignment task as well. As shown in figure 2.30, alignment is performed on a whole database of lyrics instead of just those for a single song. Then, the scores for each alignment are compared, and the best one

[2] `http://www.music-ir.org/mirex/wiki/2017:Automatic_Lyrics-to-Audio_Alignment`

determines the best match.

For evaluation, the accuracy when taking the first N number of results into account (also called recognition rate in the state of the art) is calculated. For clarification: The first N results are checked for occurrence of the expected result; then, the percentage of queries for which the correct result was detected is calculated. An alternative measure for tasks like this is the Mean Reciprocal Rank (i.e. the average inverse of the position where the expected result occurs); once again, the previously described measure was used for comparability with the state of the art.

3 State of the art

3.1 From speech to singing

Singing presents a number of challenges for speech recognition when compared to pure speech [83] [84] [85]. The following factors make speech recognition on singing more difficult than on speech, and make it necessary to adapt existing algorithms.

Larger pitch fluctuations A singing voice varies its pitch to a much higher degree than a speaking voice. It often also has very different spectral properties.

Larger changes in loudness In addition to pitch, loudness also fluctuates much more in singing than in speech.

Higher pronunciation variation Singers are often forced by the music to pronounce certain sounds and words differently than if they were speaking them.

Larger time variations In singing, sounds are often prolonged for a certain amount of time to fit them to the music. Conversely, they can also be shortened or left out completely.

In order to research this effect more closely, a small experiment was performed on a speech corpus and a singing corpus (*TIMIT* and *ACAP*, see chapter 4): The standard deviations for all the phonemes in each data set were calculated. The result is shown in figure 3.1, confirming that the variation in singing is much wider. This is particularly true for vowels.

Different vocabulary In musical lyrics, words and phrases often differ from normal conversation texts. Certain words and phrases have different probabilities (e.g. higher focus on emotional topics in singing).

Background music This is the biggest interfering factor when considering polyphonic recordings. Harmonic and percussive instruments add a huge amount of spectral

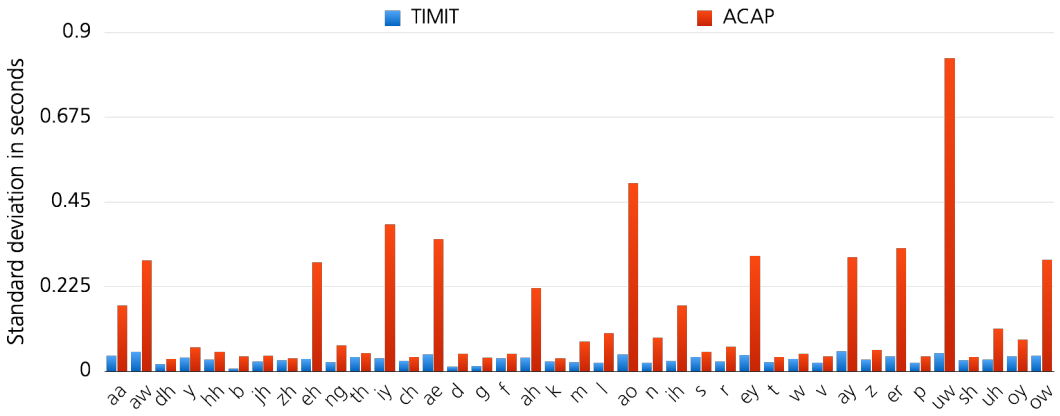


Figure 3.1: Standard deviations of phoneme durations in the *TIMIT* and *ACAP* data sets.

components to the signal, which lead to confusion in speech recognition algorithms. Ideally, these components should be removed or suppressed in a precursory step. This could be achieved, for example, by employing source separation algorithms. However, such algorithms add additional artifacts to the signal, and may not even be sufficient for this purpose at the current state of research.

Vocal activity detection (VAD) could be used as a non-invasive first step to at the very least discard segments of songs that do not contain voice. However, such algorithms often make mistakes in the same cases that are problematic for speech recognition algorithms (e.g. instrumental solos)[86].

For these reasons, most of the experiments in this work were performed on unaccompanied singing. The integration of the mentioned pre-processing algorithms would be a very interesting next step of research.

The exception are the lyrics-to-singing alignment algorithms presented in Chapter 8. Those were also tested on polyphonic music, and the algorithms appear to be largely robust to these influences.

3.2 Phoneme recognition

Due to the factors mentioned above in section **??**, phoneme recognition on singing is more difficult than on clean speech. It has only been a topic of research for a few years, and there are few publications.

One of the earliest systems was presented by Wang et al. in 2003 [87]. Acoustic modeling is performed with triphone HMMs trained on read speech in Taiwanese and

Mandarin. The language model is completely restricted to lines of lyrics in the test dataset. Testing is performed on 925 unaccompanied sung phrases in these language. Due to the highly specific language model, the word error rate is just .07.

Hosoya et al. employ a similarly classical approach from ASR that employs monophone HMMs also trained on read speech for acoustic modeling (2005) [88]. These models are adapted to singing voices using the Maximum Likelihood Linear Regression (MLLR) technique [89]. Language modeling is performed with a Finite State Automaton (FSA) specific to the Japanese language, making it more flexible than the previous system.
The system is tested on five-word unaccompanied phrases, while the adaptation is performed on 127 choruses performed by different singers. The Word Error Rate is .36 without the adaptation, and .27 after adaptation.

In 2007, Gruhne et al. presented a classical approach that employs feature extraction and various machine learning algorithms to classify singing into 15 phoneme classes [90] [91]. The specialty of this approach lies in the pre-processing: At first, fundamental frequency estimation is performed on the audio input, using a Multi-Resolution Fast Fourier Transform [92]. Based on the estimated fundamental frequency, the harmonic partials are retrieved from the spectrogram. Then, a sinusoidal re-synthesis is performed, using only the detected fundamental frequency and partials. Feature extraction is then performed on this re-synthesis instead of the original audio. Extracted features include MFCCs, PLPs [93], LPCs, and WLPCs [94]. MLP, GMM, and SVM models are trained on the resulting feature vectors. The re-synthesis idea comes from a singer identification approach by Fujihara [95].
The approach is tested on more than 2000 separate, manually annotated phoneme instances from polyphonic recordings. Only one feature vector per phoneme instance is calculated. The phoneme set is reduced down to 15 classes. Using SVM models, 56% of the tested instances were classified correctly. This is significantly better than the best result without the re-synthesis step (34%).
In [96] (2010), the approach is expanded by testing a larger set of perceptually motivated features, and more classifiers. No significant improvements are found when using more intricate features, and the best-performing classifier remains SVM.

Fujihara et al. described an approach based on spectral analysis in 2009 [97]. The underlying idea is that spectra of polyphonic music can be viewed as the weighted sum

of two types of spectra: One for the singing voice, and one for the background music. This approach then models these two spectra as probabilistic spectral templates. The singing voice is modeled by multiplying a vocal envelope template, which represents the spectral structure of the singing voice, with a harmonic filter, which represents the harmonic structure of the produced sound itself. This is analogous to the source-filter model of speech production [98]. For recognizing vowels, five such harmonic filters are prepared (a - e - i - o - u). Vocal envelope templates are trained on voice-only recordings, separated by gender. Templates for background music are trained on instrumental tracks. In order to recognize vowels, the probabilities for each of the five harmonic templates are estimated. As a side product, the algorithm also estimates the fundamental frequency of the singing voice.
As described, the phoneme models are gender-specific and only model five vowels, but also work for singing with instrumental accompaniment. The approach is tested on 10 Japanese-language songs. The best result is 65% correctly classified frames, compared to the 56% with the previous approach by this team, based on GMMs.

Also in 2009, Mesaros et al. picked Hosoya's approach back up by using MFCC features and GMM-HMMs for acoustic modeling [99] and adapting the models for singing voices. These models are trained on the CMU ARCTIC speech corpus [1]. Then, different Maximum Likelihood Linear Regression (MLLR) techniques for adapting the models to singing voices are tested [89].
The adaptation and test corpus consists of 49 voice-only fragments from 12 pop songs with durations between 20 and 30 seconds. The best results are achieved when both the means and variances of the Gaussians are transformed with MLLR. The results improved slightly when not just a single transform was used for all phonemes, but when they were grouped into base classes beforehand, each receiving individual transformation parameters. The best result is around .79 Phoneme Error Rate on the test set.
In [100] and [101], language modeling is added to the presented approach. Phoneme-level language models are trained on the CMU ARCTIC corpus as unigrams, bigrams, and trigrams, while word-level bigram and trigram models are trained on actual song lyrics in order to match the application case. The output from the acoustic models is then refined using these language models. The approach is tested on the clean singing corpus mentioned above, and on 100 manually selected fragments of 17 polyphonic pop songs. To facilitate recognition on polyphonic music, a vocal separation algorithm is

[1] `http://festvox.org/cmu_arctic/`

introduced [102].
Using phoneme-level language modeling, the phoneme error rate on clean singing is reduced to .7. On polyphonic music, it is .81. For the word recognition approach, the word error rate is .88 on clean singing, and .94 on the polyphonic tracks.
A more detailed voice adaptation strategy is tested in [103]. Instead of adapting the acoustic models with mixed singing data, they are adapted gender-wise, or to specific singers. With the gender-specific adaptations, the average phoneme error rate on clean singing is lowered to .81 without language modeling, and .67 with language modeling. Singer-specific adaptation does not improve the results, probably because of the very small amount of adaptation data in this case.
In [104] (2014), McVicar et al. build on a very similar baseline system, but also exploit repetitions of choruses to improve transcription accuracy. This has been done for other MIR tasks, such as chord recognition, beat tracking, and source separation. They propose three different strategies for combining individual results: Feature averaging, selection of the chorus instance with the highest likelihood, and combination using the Recogniser Output Voting Error Reduction (ROVER) algorithm [105]. They also employ three different language models, two of which were matched to the test songs (and therefore not representative for general application). 20 unaccompanied, English-language songs from the RWC database [106] were used for testing; chorus sections were selected manually. The best-instance selection and the ROVER strategies improve results significantly; with the ROVER approach and a general-purpose language model, the Phoneme Error Rate is at .74 (versus .76 in the baseline experiment), while the Word Error Rate is improved from .97 to .9. Interestingly, cases with a low baseline result benefit the most from exploiting repetition information.

The final system was proposed by Hansen in 2012 [21]. It also employs a classical approach consisting of a feature extraction step and a model training step. Extracted features are MFCCs and TRAP (TempoRAl Pattern) features. Then, Multilayer Perceptrons (MLPs) are trained separately on both feature sets. The assumption is that each feature models different properties of the considered phonemes: Short-term MFCCs are good at modeling the pitch-independent properties of stationary sounds, such as sonorants and fricatives. On the flip-side, TRAP features are able to model temporal developments in the spectrum, forming better representations for sounds like plosives or affricates.
The results of both MLP classifiers are combined via a fusion classifier, also an MLP. Then, Viterbi decoding is performed on its output.

The approach is trained and tested on a data set of 12 vocal tracks of pop songs, which were manually annotated with a set of 27 phonemes. The combined system achieves a recall of .48, compared to .45 and .42 for the individual MFCC and TRAP classifiers respectively. This confirms the assumption that the two features complement each other. The phoneme-wise results further corroborate this.

3.3 Lyrics-to-audio alignment

In contrast to the other tasks discussed in this chapter, the task of lyrics-to-audio alignment has been the focus of many more publications. A comprehensive overview until 2012 is given in [84].

A first approach was presented in 1999 by Loscos et al. [107]. The standard forced alignment approach from speech recognition is adapted for singing. MFCCs are extracted first, and then a left-to-right HMM is employed to perform alignment via Viterbi decoding. Some modifications are made to the Viterbi algorithm to allow for low-delay alignment. The approach is trained and tested on a very small (22 minutes) database of unaccompanied singing, but no quantitative results are given.

The first attempt to synchronize lyrics to polyphonic recordings was made by Wang et al. in 2004 [108]. They propose a system, named "LyricAlly", to provide line-level alignments for karaoke applications. Their approach is heavily based on musical structure analysis. First, the hierarchical rhythm structure of the song is estimated. The result is combined with an analysis of the chords and then used to split the song into sections, based on a chorus detection algorithm. Second, Vocal Activity Detection (VAD) using HMMs is performed on each section. Then, sections of the text lyrics are assigned to the detected sections (e.g. verses, choruses). In the next step, the algorithm determines whether the individual lines of the lyrics match up with the vocal sections detected by the VAD step. If they do not, grouping or partitioning is performed. This is based on the assumption that lyrics match up to rhythmic bars as determined by the hierarchical rhythm analysis. The expected duration of each section and line is estimated using a Gaussian distributions of phoneme durations from a singing data set. In this manner, lines of text are aligned to the detected vocal segments. The approach is tested on 20 manually annotated pop songs. On the line level, the average error is .58 seconds for the starting points and $-.48$ seconds for the durations. The system components are analyzed in more detail in [109].

In 2006, the same team also presented an approach that also performs rhythm and bar analysis to facilitate syllable-level alignment [110]. For the phoneme recognition

step, an acoustic model is trained on speech data and adapted to singing using the previously mentioned 20 songs. The possible syllable positions in the alignment step are constrained to the note segments detected in the rhythm analysis step. Due to annotator disagreement on the syllable level, the evaluation is performed on the word level. On three example songs, the average synchronization error rate is .19 when allowing for a tolerance of 1/4 bar.

Sasou et al. presented a signal parameter estimation method for singing employing an auto-regressive HMM (AR-HMM) in 2005 [111]. This method is particularly suited for modeling high-pitched signals, which is important for singing voices and usually not taken into account in speech processing. Models trained on speech are adapted to singing using MLLR. For evaluation, the method is applied to the task of lyrics-to-audio alignment and tested on 12 Japanese-language songs. For each song, a specific language model is prepared. The correct word rate is 0.79.

Chen et al. presented an based on MFCC features and Viterbi alignment, which is commonly used in ASR, in 2006 [112]. Vocal Activity Detection is performed in a pre-processing step, and then GMM-HMMs are used for Viterbi alignment between the audio and the lyrics. Once again, MLLR is used to adapt the acoustic models to singing. In addition, the grammar is specifically tailored to the lyrics. On a data set of Chinese songs by three singers, a boundary accuracy of 0.76 is obtained on the syllable level.

A similar approach which does not require in-depth music analysis was presented by Fujihara et al. in 2006 [113]. Once again, a straightforward Viterbi alignment method from speech recognition is refined by introducing three singing-specific pre-processing steps: Accompaniment sound reduction, Vocal Activity Detection, and phoneme model adaptation.
For accompaniment reduction, the previously mentioned harmonic re-synthesis algorithm from [95] is used again. For Vocal Activity Detection, a HMM is trained on a small set of unaccompanied singing using LPC-derived MFCCs and F_0 differences as features. The HMM can be parameterized to control the rejection rate. For the phoneme model adaptation, three consecutive steps are tested: Adaptation to a clean singing voice, adaptation to a singing voice segregated with the accompaniment reduction method, and on-the-fly adaptation to a specific singer. MFCC features are used for the Viterbi alignment, which is performed on the vowels and syllabic nasals

([m],[n],[l]) only.
Ten Japanese pop songs were used for testing. Evaluation was done one the phrase level by calculating the proportion of the duration of correctly aligned sections to the total duration of the song. For eight of the ten songs, this proportion was .9 or higher when using the complete system, which the authors judge as satisfactory. Generally, the results are lower for performances by female singers, possibly because of the higher F_0s. These performances also benefit the most from the Vocal Activity Detection step, even though this also seems to perform a bit worse for female singing. All three levels of phoneme model adaptation contribute to the success of the system.
In 2008, the authors improved upon this system with three modifications: Fricative detection, filler models, and new features for the Vocal Activity Detection step [114]. Fricative detection is introduced because the previous system was only based on vowels and nasals, due to the fact that the harmonic re-synthesis discards other consonants. In the new system, fricatives are detected before this step and then retained for the alignment (stops are not used because they are too short).
The filler model is employed because singers sometimes add extraneous lyrics (like "la la la" or "yeah") to their performances.
As mentioned above, Vocal Activity Detection does not work as well for female performances because of inaccuracies of the spectral envelope estimation in high pitch regions. For this reason, the features are replaced in the new version by comparing the power of the harmonic components directly with those of similar F_0 regions.
The approach is again evaluated on ten Japanese pop songs. The original system produces an average accuracy of .81, which is raised to .85 with the new improvements.
In [115], the whole system is presented succinctly and evaluated in more detail. Additionally, integration into a music playback interface is described.

Mauch et al. augmented the same approach in 2010 by using chord labels, which are often available together with the lyrics on the internet [116] [117]. Chords usually have longer durations than individual phonemes, and are therefore easier to detect. In this way, they provide a coarse alignment, which can be used to simplify the shorter-scale phoneme-level alignment. A chroma-based approach is used to estimate the chords. Information about the chord alignments is directly integrated into the HMM used for alignment.
In [117], a large range of parameterizations is tested on 20 English-language pop songs. The highest accuracy for the baseline approach (without chord information) is .46. Using chord position information, this rises to .88. Interestingly, Vocal Activity Detection

improves the result when not using chords, but decreases it for the version with chord alignments, possibly because the coarse segmentation it provides is already covered by the chord detection in the second case. The method is also able to cope with incomplete chord transcriptions while still producing satisfactory results.

In 2007, Wong et al. presented a specialized approach for Cantonese singing that does not require phoneme recognition [118]. Since Cantonese is a tonal language, prosodic information can be inferred from the lyrics. This is done by estimating relative pitch and timing of the syllables by using linguistic rules. On the other side, a vocal enhancement algorithm is applied to the input signal, and its pitches and onsets are calculated. Then, both sets of features are aligned using Dynamic Time Warping. 14 polyphonic songs were used for evaluation. The approach reaches an average (duration) accuracy of .75.

Lee et al. also follow an approach without phoneme recognition (2008) [119]. It is purely based on structural analysis of the song recording, which is done by calculating a self-similarity matrix and using it for segmentation. The algorithm takes structural a-priori knowledge into account, e.g. the fact that choruses usually occur most frequently and do not differ very much internally. Lyrics segments are annotated by hand, splitting them up into paragraphs and labeling them with structural parts ("ï¿½ntro", "verse", "chorus", and "bridge"). Then, Dynamic Programming is performed to match the lyrics paragraphs to the detected musical segments. A Vocal Activity Detection step is also introduced. Testing the approach on 15 English-language pop songs with 174 lyrics paragraphs in total, they obtain an average displacement error of 3.5 seconds.

Mesaros et al. also present an alignment approach that makes use of their phoneme recognition approach described above in **??** [82], but also using a harmonic re-synthesis step for vocal separation. Based on these models, they employ straightforward Viterbi alignment and obtain an average displacement error of 1.4 seconds for line-wise alignment (.12 seconds when not using absolute values). The test set consists of 17 English-language pop songs. They identify mistakes in the vocal separation step as the main source of error.

Two specialized approaches were presented in the past two years. The first one is part of a score-following algorithm by Gong et al. [120]. Here, vowels are used to

aid alignment of the musical score to the recording, assuming the score contains the lyrics. This is done by training vowel templates on a small set of sung vowels. Spectral envelopes are used as features. Two different strategies for fusing the vowel and melody information are tested ("early" and "late"), as well as singer-specific adaptation of the templates via MAP (Maximum A-Posteriori). The training set consists of 160 vowel instances per singer, the test set of 8 full unaccompanied French-language songs per singer. The average displacement error is around .68 for both singer-specific and -adapted models (best strategy).

Finally, Dzhambazov et al. presented a method that integrates knowledge of note onsets into the alignment algorithm [121]. Pitch extraction and note segmentation are performed in parallel with phoneme recognition via HMMs, and both results are refined with a transition model. A variable time HMM (VTHMM) is used to model the rules for phoneme transitions at note onsets.
On a test dataset of 12 unaccompanied Turkish-language Makam performances, the method achieves an alignment accuracy of .76. For polyphonic recordings (usually with accompaniment by one or more string instruments), a vocal resynthesis step is introduced. The average accuracy in this case is .65.

3.4 Lyrics retrieval

As described in 3.2, Hosoya et al. developed a system for phoneme recognition, which they also apply to lyrics retrieval [88]. On a dataset of 238 children's songs, they obtain a recognition rate of .86 for the Top 1 result, and of .91 for the Top 10 results. In [122] and [123], more experiments are conducted. As a starting point, the number of words in the queries is fixed at 5, resulting in a recognition rate of .9 (Top 1 result). Then, a melody recognition is used to verify the matches proposed by the speech reognition step, raising the recognition rate to .93. The influence of the number of words in the query is also evaluated, confirming that retrieval becomes easier the longer the query is. However, even at a length of just three words, the recognition rate is .87 (vs. .81 without melody verification).

Similarly, Wang et al. presented a query-by-singing system in 2010 [124]. The difference here is that melody and lyrics information are weighted equally in the distance calculation. Lyrics are recognized here with a bigram HMM model trained on speech. The results are interpreted as syllables. A syllable similarity matrix is employed for

calculating phoneme variety in the query, which is used for singing vs. humming discrimination. Assuming that only the beginning of each song is used as the starting point for queries, the first 30 syllables of each song are transformed into an FSM language model and used for scoring queries against each song in the database. The algorithm is tested on a database of 2154 Mandarin-language songs, of which 23 were annotated and the remainder are used as "noise" songs. For the Top 1 result, a recognition rate of .91 is achieved for the system combining melody and lyrics information, compared to .88 for the melody-only system.

As described in 3.2, Mesaros et al. developed a sophisticated system for phoneme and word recognition in singing. In [125], [126], and [101], they also describe how this system can be used for lyrics retrieval. This is the only purely lyrics-based system in literature. Retrieval is performed by recognizing words in queries with the full system, including language modeling, and then ranking each lyrics segment by the number of matching words (bag-of-words approach). The lyrics database is constructed from 149 song segments (between 9 and 40 seconds in the corresponding recordings). Out of these segments, recordings of 49 are used as queries to test the system. The Top 1 recognition rate is .57 (.71 for the Top 10).

3.5 Language identification

A first approach for language identification in singing was proposed by Tsai and Wang in 2004 [127]. At its core, the algorithm is similar to Parallel Phoneme Recognition (PPR). However, instead of full phoneme modeling, they employ an unsupervised clustering algorithm to the input feature data and tokenize the results to form language-specific codebooks (plus one for background music). Following this, the results from each codebook are run through a matching language models to determine the likelihood that the segment was performed in this language. Prior to the whole process, vocal/non-vocal segmentation is performed. This is done by training GMMs on segments of each language, and on non-vocal segments. MFCCs are used as features.
The approach is tested on 112 English- and Mandarin-language polyphonic songs each, with 32 songs performed in both languages. A classification accuracy of .8 is achieved on the non-overlapping songs. On the overlapping songs, it is only at .7, suggesting some influence of the musical material (as opposed to the actual language characteristics). Misclassifications occur more frequently on the English-language songs, possibly because of accents of Chinese singers performing in English, and because of louder

background music.

A second, simpler approach was presented by Schwenninger et al. in 2006 [128]. They also extract MFCC features, and then use these to directly train statistical models for each language. Three different pre-processing strategies are also tested: Automatic vocal/non-vocal segmentation, distortion reduction, and azimuth discrimination. Vocal/non-vocal segmentation is performed by thresholding the energy in high-frequency bands as an indicator for voice presence over 1 second windows. This leaves a relatively small amount of material per song. Distortion reduction is employed to discard strong drum and bass frames where the vocal spectrum is masked by using a mel-based approach. Finally, azimuth discrimination attempts to detect and isolate singing voice panned to the center of the stereo scene.
The approach is tested on three small data sets of speech, unaccompanied singing, and polyphonic music. Without pre-processing steps, the accuracy is .84, .68, and .64 respectively, highlighting the increased difficulty of language identification on singing versus speech, and on polyphonic music versus pure vocals. On the polyphonic corpus, the pre-processing steps do not improve the result.

In 2011, Mehrabani and Hansen presented a full Parallel Phoneme Recognition followed by Language Modeling (PPRLM) approach for singing language identification. MFCC features are run through phoneme recognizers for Hindi, German, and Mandarin; then, the results are scored by individual language models for each considered language. In addition, a second system is employed which uses prosodic instead of phonetic tokenization. This is done by modeling pitch contours with Legendre polynomials, and then quantizing these vectors with previously trained GMMs. The results are then again used as inputs to language models.
The approach is trained and tested on a corpus containing 12 hours of unaccompanied singing and speech in Mandarin, Hindi, and Farsi. The average accuracy for singing is .78 and .43 for the phoneme- and prosody-based systems respectively, and .83 for a combination of both.

Also in 2011, Chandrasekhar et al. presented a very interesting approach for language identification on music videos, taking into account both audio and video features [129]. On the audio side, the spectrogram, volume, MFCCs, and perceptually motivated Stabilized Auditory Images (SAI) are used as inputs. One-vs-all SVMs are trained for each language. The approach is trained and tested on 25,000 music videos,

taking 25 languages into consideration. Using audio features only, the accuracy is .45; combined with video features, it rises to .48. It is interesting to note that European languages seem to achieve much lower accuracies than Asian and Arabic ones. English, French, German, Spanish and Italian rank below .4, while languages like Nepali, Arabic, and Pashto achieve accuracies above .6. It is possible that the language characteristics of European languages make them harder to discriminate (especially against each other) than others.

3.6 Keyword spotting

Keyword spotting in singing was first attempted in 2008 by Fujihara et al. [130]. Their approach employs a phoneme recognition step first, which is again based on the vocal re-synthesis method first described in [95]. MFCCs and power features are extracted from the re-synthesized singing and used as inputs to a phoneme model, similar to Gruhne's phoneme recognition approach mentioned above in **??**. Three phoneme models are compared: One trained on pure speech and adapted with a small set of singing recordings, one adapted with all recordings, and one trained directly on singing. Viterbi decoding is then performed using keyword-filler HMMs (see **??**) to detect candidate segments where keywords may occur. These segments are then re-scored through the filler HMM to verify the occurrence.
The method is tested on 79 unaccompanied Japanese-language songs from the RWC database [106]. The Phoneme Error Rate is .73 for the acoustic models trained on speech, .67 for the adapted models, and .49 for the models trained on singing (it should be mentioned that the same songs were used for training and testing, although a cross-validation experiment appears to show that the effect is negligible). The employed evaluation measure is "link success rate", describing the percentage of detected phrases that were linked correctly to other occurrences of the phrase in the data set. In that sense, it is a sort of accuracy measure. The link success rate for detecting the keywords is .3. The authors show that the result depends highly on the number of phonemes in the considered keyword, with longer keywords being easier to detect.

In 2012, Mercado et al. presented an approach to keyword spotting in singing based on a different principle: Dynamic Time Warping (DTW) between a sung query and the requested phrase in the song recording. In particular, Statistical Sub-sequence DTW is the algorithm employed for this purpose. MFCCs are used as feature inputs, then the costs of the warping paths are calculated from all possible starting points to obtain

candidate segments, which are then further refined to find the most likely position.
The approach is tested on a set of vocal tracks of 19 pop songs (see Section **??**) as the references, and phrase recordings by amateur singers as the queries, but no quantitative results are given. The disadvantage of this approach lies in the necessity for audio recordings of the key phrases, which need to have at least similar timing and pitch as the reference phrases.

Finally, Dzhambazov et al. developed a score-aided approach to keyword spotting in 2015 [131]. A user needs to select a keyword phrase and a single recording in which this phrase occurs. The keyword is then modeled acoustically by concatenating recordings of the constituent phonemes (so-called acoustic keyword spotting). Similar to Mercado's approach, Sub-Sequence DTW is then performed between the acoustic template and all starting positions in the reference recording to obtain candidate segments. These segments are then refined by aligning the phonemes to the score in these positions to model their durations. This is done by using Dynamic Bayesian Network HMMs. Then, Viterbi decoding is performed to re-score the candidate segments and obtain the best match.
The approach is tested on a small set of unaccompanied Turkish-language recordings of traditional Makam music. The Mean Average Precision (MAP) for the best match is .08 for the DTW approach only, and .05 for the combined approach. For the top-6 results, the MAP is .26 and .38 respectively.

4 Data sets

This chapter contains descriptions of all the data sets (or corpora) used over the course of this thesis. They are grouped into speech-only data sets, data sets of unaccompanied (=a-capella) singing, and data sets of full musical pieces with singing ("real-world" data sets).

4.1 Speech data sets

4.1.1 TIMIT

TIMIT is, presumably, the most widely used corpus in speech recognition research [132]. It was developed in 1993 and consists of 6300 English-language audio recordings of 630 native speakers with annotations on the phoneme, word, and sentence levels. The corpus is split into a training and a test section, with the training section containing 4620 utterances, and the test section containing 1680. Each of those utterances has a duration of a few seconds. The recordings are sampled at $16,000Hz$ and are mono channel.
The phoneme annotations follow a model similar to ARPABET and contain 61 different phonemes [133].

4.1.2 NIST Language identification corpora

The National Institute for Standards and Technology (NIST) regularly runs various speech recognition challenges, one of them being the Language Recognition Evaluation (LRE) task, which is offered every two to four years[1]. To this end, they publish training and evaluation corpora of speech in several languages. They consist of short segments (up to 35 seconds) of free telephone speech by many different speakers. The recordings are mono channel with a sampling rate of $8000Hz$.
The corpus for the 2003 challenge was used in this work for comparison of language

[1] `https://www.nist.gov/itl/iad/mig/language-recognition`

identification algorithms against speech data [134]. Only the English-, German-, and Spanish-language subsets were chosen, since those languages are covered by the corresponding singing dataset. To balance out the languages, 240 recordings were used for each of them, making up around 1 hour of material.

4.1.3 OGI Multi-Language Telephone Speech Corpus

In 1992, the Oregon Institute for Science and Technology (OGI) also published a multilingual corpus of telephone recordings, called the OGI Multi-Language Telephone Speech Corpus, to facilitate multi-language ASR research. Just like the NIST corpora, it has become widely used for speech recognition tasks. Again, the English-, German-, and Spanish-language subsets were used in this work. They each consist of more than 1000 recordings per language of up to 50 seconds duration, making up a total of about three to five hours. In contrast to the NIST corpus, the recordings are somewhat less "clean" with regards to accents and background noise; the sampling rate is also $8000Hz$ on a mono channel.
In many experiments, languages were balanced out by randomly discarding utterances to obtain equal numbers. For experiments on longer recordings, results on individual utterances were aggregated for each speaker and also balanced down to the lowest number, producing 118 documents per language (354 in sum).

4.2 Unaccompanied singing data sets

4.2.1 YouTube data set

As opposed to the speech case, there are no standardized corpora for sung language identification. For the sung language identification experiments, files of unaccompanied singing were therefore extracted from *YouTube*[2] videos. This was done for three languages: English, German, and Spanish. The corpus consists of between 116 (258min) and 196 (480min) examples per language. These were mostly videos of amateur singers freely performing songs without accompaniment. Therefore, they are of highly varying quality and often contain background noise. The audio was downloaded in the natively provided quality, but then downsampled to $8000Hz$ and averaged to a mono channel for uniformity and for compatibility with the speech corpora for language identifica-

[2] `http://www.youtube.com`, Last checked: 05/16/13

Table 4.1: Amounts of data in the three used data sets: Sum duration on top, number of utterances in italics.

hh:mm:ss ***#Utterances*** ***#Speakers*** **∅ Duration/Speaker**	**NIST2003LRE**	**OGIMultilang**	**YTAcap**
English	00:59:08 *240* - -	05:13:17 *1912* 200 00:01:34	08:04:25 *1975* 196 00:02:28
German	00:59:35 *240* - -	02:52:27 *1059* 118 00:01:28	04:18:57 *1052* 116 00:02:14
Spanish	00:59:44 *240* - -	03:05:45 *1151* 129 00:01:26	07:21:55 *1810* 185 00:02:43

tion. Most of the performers contributed only a single song, with just a few providing up to three. This was done to avoid effects where the classifier recognizes the singer's voice instead of the language.

Special attention was paid to musical style. Rap, opera singing, and other specific singing styles were excluded. All the songs performed in these videos were pop songs. Different musical styles can have a high impact on language classification results. The author tried to limit this influence as much as possible by choosing recordings of pop music instead of language-specific genres (such as Latin American music).

For many experiments, they were split up into segments of 10-20 seconds at silent points (3,156 "utterances" in sum). In most cases, these segments were then balanced for the languages by randomly discarding superfluous segments. In other experiments, a balanced set of the whole songs was used (i.e. 116 songs per language).

An overview of the amounts of data in the three corpora for language identification is given in Table 4.1.

4.2.2 Hansen's vocal track data set

This is one of the data sets used for keyword spotting and phoneme recognition. It was first presented in [21]. It consists of the vocal tracks of 19 commercial English-language pop songs. They are studio quality with some post-processing applied (EQ, compression, reverb). Some of them contain choir singing. These 19 songs are split up into 920 clips that roughly represent lines in the song lyrics. The original audio quality is $44,100Hz$ on mono channels; for compatibility with models trained on *TIMIT*, they were downsampled to $16,000Hz$.
Twelve of the songs were annotated with time-aligned phonemes. The phoneme set is the one used in CMU Sphinx[3] and TIMIT [132] and contains 39 phonemes. All of the songs were annotated with word-level transcriptions. This is the only one of the singing data sets that has full manual phoneme annotations, which are assumed to be reliable and can be used as ground truth.
For comparison, recordings of spoken recitations of all song lyrics were also made. These were all performed by the same speaker (the author).
This data set will be referred to as *ACAP*.

4.2.3 DAMP data set

As described, Hansen's data set is very small and therefore not suited to training phoneme models for singing. As a much larger source of unaccompanied singing, the *DAMP* data set, which is freely available from Stanford University[4][135], was employed. This data set contains more than 34,000 recordings of amateur singing of full songs with no background music, which were obtained from the *Smule Sing!* karaoke app. Each performance is labeled with metadata such as the gender of the singer, the region of origin, the song title, etc. The singers performed 301 English-language pop songs. The recordings have good sound quality with little background noise, but come from a lot of different recording conditions. They were originally provided in OGG format at a sampling rate of $22,050Hz$ (mono channel), but were also converted to wave format and downsampled to $16,000Hz$ for compatibility.
No lyrics annotations are available for this data set, but the textual lyrics can be obtained from the *Smule Sing!* website[5]. These are, however, not aligned in any way. Such an alignment was performed automatically on the word and phoneme levels (see

[3] `http://cmusphinx.sourceforge.net/`
[4] `https://ccrma.stanford.edu/damp/`
[5] `http://www.smule.com/songs`

section 5.3).
The selection of songs is not balanced, with performances ranging between 20 and 2038 instances. Female performances are also much more frequent than male ones, and gender often plays a role when training and evaluating models. For these reasons, several subsets of the dataset were composed by hand.
In order to generate training data sets, the data was balanced by songs so that certain songs (and their phoneme distributions) would not be overrepresented. Since least represented song in the original data set has 20 recordings, 20 to 23 recordings per song were chosen at random to generate a training data set, which was named **DampB**. For each song, as many different singers as possible were selected. Additionally, recordings with more "Loves" (user approval) were more likely to be selected (however, a large percentage of the original data set did not have such ratings). This resulted in a data set of 6,902 recordings.
This process was then repeated, this time only taking recordings by singers of one gender into account. In this way, data sets of recordings by female and male singers only were, named **DampF** and **DampM** respectively. Due to the gender split, there were fewer recordings of some of the songs available. Male singers in particular are underrepresented in the original data set. Therefore, **DampF** contains 6,564 recordings, while **DampM** contains 4,534. These sizes are roughly in the same range as for the mixed-gender data set, which enables the comparison of models trained on all three.
These three data sets do not contain balanced amounts of phonemes; therefore. subsets of them were created where phoneme frames were discarded until they were balanced and a maximum of 250,000 frames per phoneme were left, where possible. These data set are named *DampBB*, *DampFB*, and *DampMB*, and they are about 4% the size of their respective source data sets.
Since all of these data sets are still much larger than the *TIMIT* speech corpus, another subset of similar size was created for comparison. On the basis of the balanced *DampBB* data set, phoneme instances were discarded until they were balanced and 60,000 frames per phoneme were left, resulting in the *DampBB_small* data set.
For testing new algorithms, two small test data sets were created from the original *DAMP* data set. These are, again, split by gender, and contain one recording per song, resulting in 301 recordings for the female data set and 300 for the male one (since there was one song with not enough available male recordings). They are called **DampTestF** and **DampTestM** respectively. For mixed-gender training, testing was simply performed on both data sets.
Additionally, 20 phrases from each of the test data sets were manually selected for

good singing and recording quality. This was necessary for fine-tuning and analysis of the retrieval algorithms.

To sum up:

DampB Contains 20 to 23 full recordings per song (more than 6000 in sum), both male and female.

DampBB Same as before, but phoneme frames were discarded until they were balanced and a maximum of 250,000 frames per phoneme were left, where possible. This data set is about 4% the size of *DampB*.

DampBB_small Same as before, but phoneme frames were discarded until they were balanced and 60,000 frames per phoneme were left (a bit fewer than the amount contained in *TIMIT*). This data set is about half the size of *DampBB*.

DampF and DampM Each of these data sets contains 20 to 23 full recordings per song performed by singers of one gender, female and male respectively.

DampFB and DampMB Starting from *DampF* and *DampM*, these data sets were then reduced in the same way as *DampBB*. *DampFB* is roughly the same size, *DampMB* is a bit smaller because there are fewer male recordings.

DampTestF and DampTestM Contains one full recording per song and gender (300 each). These data sets were used for testing. There is no overlap with any of the training data sets.

DampRetrievalF and DampRetrievalM 20 hand-picked sung phrases from *DampF* and *DampM* of a few seconds duration with good enunciation and good audio quality. These were selected for fine-tuning retrieval approaches. An overview can be found in appendix **??**.

An overview of the amounts of data is given in table 4.2.

4.2.4 Retrieval data set by the author

This is a data set of 90 short lyrics phrases from the *DAMP* data set sung with clear enunciation by the author. The phrases were recorded with a smartphone microphone for testing a demo app for the retrieval algorithm. They were then further utilized to finetune this retrieval algorithm. The recordings are a few seconds in duration with a sampling rate of 16,000 Hz on a mono channel.

	Both genders	Female	Male
Full	DampB (6,902) *11:08:34:30*	DampF (6,654) *10:19:05:27*	DampM (4,534) *07:13:01:06*
Balanced	DampBB *08:49:02*	DampFB *08:29:39*	DampMB *05:53:01*
Reduced	DampBB_small *04:10:13*		
Test		DampTestF (301) *18:15:43*	DampTestM (300) *17:58:02*
Retrieval		DampRetrievalF (20) *01:56*	DampRetrievalM (20) *01:52*

Table 4.2: Overview of the structure of the *DAMP*-based phonetically annotated data sets, number of recordings in brackets, duration in italics (dd:hh:mm:ss). Note that no number of recordings is given for some data sets because their content was selected phoneme-wise, not song-wise. For retrieval, the recordings are short phrases instead of full songs.

4.3 Accompanied singing data sets

4.3.1 QMUL Expletive data set

This data set consists of 80 popular full songs which were collected at Queen Mary University, most of them Hip Hop. 711 instances of 48 expletives were annotated on these songs. In addition, the matching textual, unaligned lyrics were retrieved from the internet. The audio is provided at $44,100kHz$ sampling rate in stereo, but was also resampled to $16,000Hz$ and a mono track for compatibility with some models.

4.3.2 Mauch's data set

This data set was first presented in [117] and is used for alignment evaluation. It consists of 20 English-language pop songs with singing and accompaniment. Word onsets were manually annotated. This data set will be referred to as *Mauch*.

#Phonemes	Keywords
2	eyes
3	love, away, time, life, night
4	never, baby, world, think, heart, only, every
5	always, little

Table 4.3: All 15 tested keywords, ordered by number of phonemes.

4.3.3 Hansen's vocal track data set (polyphonic)

This data sets consists of the songs in the *ACAP* data set described above in section 4.2.2, but with instrumental annotations. The annotations are carried over. This data set is also used for alignment evaluation and denoted as *ACAP_Poly*.

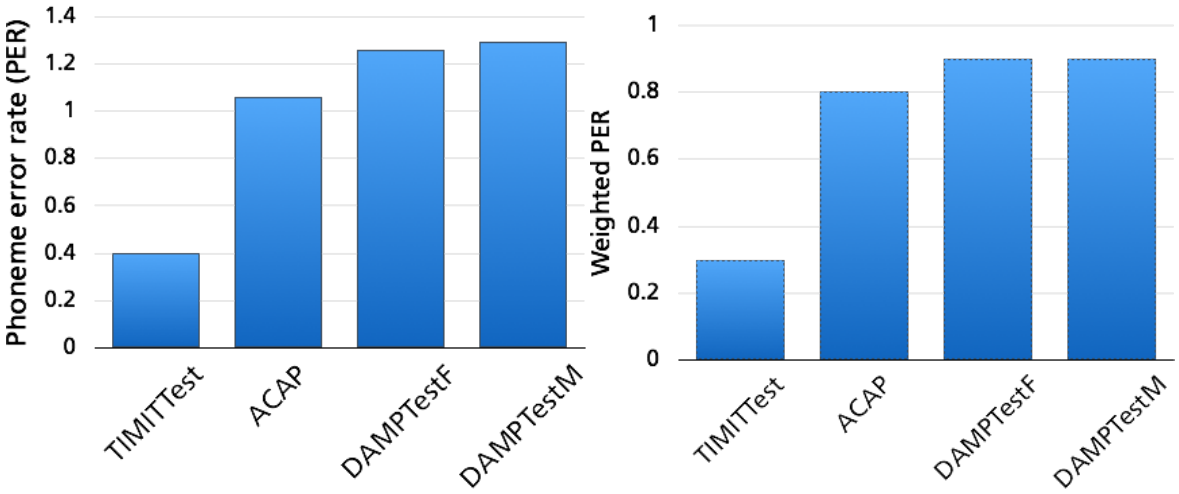
4.4 Keywords

From the 301 different song lyrics of the *DAMP* data sets, 15 keywords were chosen by semantic content and frequency to test the keyword spotting algorithms. Each keyword occurs in at least 50 of the 301 songs, and also appears in the *ACAP* data set. The keywords are shown in table 4.3. A list of the number of occurrences in each data set is given in appendix **??**.

5 Singing phoneme recognition

5.1 Phoneme recognition using models trained on speech

As a starting point, models for phoneme recognition were trained on speech data, specifically the *TIMIT* corpus. As in many classical ASR approaches, HMMs were selected as a basis and trained using the Hidden Markov Toolkit (HTK)[136].
Additionally, Deep Neural Networks (DNN) were trained for comparison with models trained on the other data sets. They had three hidden layers of 1024 nodes, 850 nodes, and 1024 nodes again.
In both cases, 13 MFCCs were extracted, and deltas and double-deltas were calculated, resulting in a feature vector of dimension 39. As the output, the set of 39 monophones described in section 2.5.1 was used.
The HMMs were used for aligning text lyrics to audio in some of the following approaches. To verify the quality of such an alignment, this was tested on the part of the *ACAP* singing corpus that has phoneme annotations. On average, the alignment error was 0.16 seconds. A small manual check suggests that this value may be in the range of the agreement of annotators. A closer inspection of the results also shows that the biggest contribution to this error occur because a few segments were heavily misaligned, whereas most of them are just slightly off.
The DNN models were trained to perform phoneme recognition. This system was evaluated by first generating phoneme posteriorgrams from the test audio using the DNN models, and then running Viterbi decoding on those to extract phoneme strings. Then, the Phoneme Error Rate and the Weighted Phoneme Error Rate were calculated as described in section **??**.
The results for DNN models trained on *TIMIT* are shown in figure 5.1. For validation, the model was tested on the test part of *TIMIT*, resulting in a Phoneme Error Rate of .4 and a Weighted Phoneme Error Rate of .3. Higher values on these corpora can be found in literature, but those systems are usually more sophisticated and include lan-



(a) Phoneme error rate (b) Weighted phoneme error rate

Figure 5.1: Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT*.

guage modeling steps and gender- or speaker-adapted models. In this scenario, those values serve as a validation of the recognition ability of the model on unseen data, and as an upper bound for the other results.

On the *ACAP* data set, the phoneme error rate is 1.06 and the weighted phoneme error rate is .8. Performing the same evaluation on the *DAMP* test data sets generates similar results: A phoneme error rate of 1.26/1.29, and a weighted phoneme error rate of .9. This demonstrates that the performance of models trained on speech leaves room for improvement when used for phoneme recognition in singing. The difference between both singing data sets can be explained by their content: *ACAP* is much smaller, and contains cleaner singing, both considering the recording quality and the singing performance. Songs in this data set were performed by professional singers, whereas the *DAMPTest* sets contain recordings by amateurs who may not always enunciate as clearly. Additionally, the phoneme annotations for the *DAMPTest* sets were generated from the song lyrics; these do not correspond to the actual performed phonemes in all cases.

It can be assumed that models trained on better-matching conditions (i.e. singing) would perform much better at this task. The problem with this approach lies in the lack of data sets that can be used for these purposes. In contrast with speech, no large corpora of phonetically annotated singing are available. In the following sections, workarounds for this problem are tested.

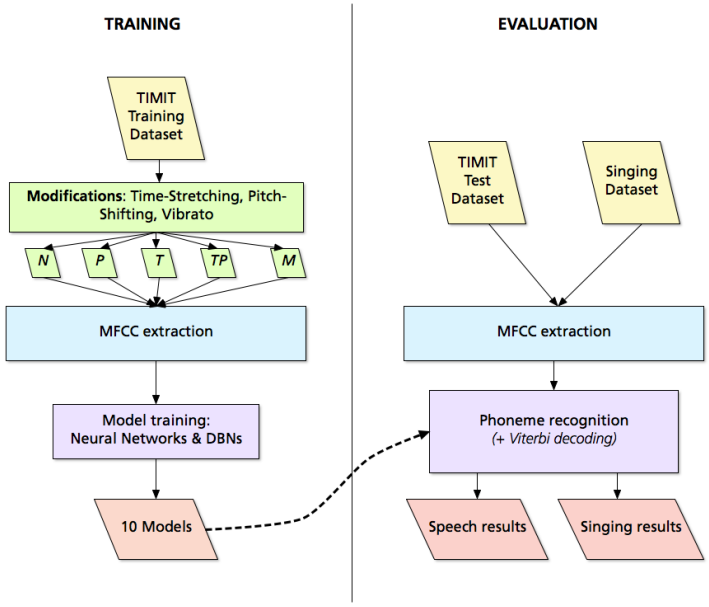


Figure 5.2: Overview of the "songified" phoneme recognition system

5.2 Phoneme recognition using models trained on "songified" speech

When there is a scarcity of suitable training data, attempts are often made to generate such data artificially from existing data for other conditions. For example, this is often done when models for noisy speech are required [137][138]. Inspired by this, one idea was making existing speech data sets more "song-like" and use these modified datasets to train models for phoneme recognition in singing. The *TIMIT* corpus was once again used as a basis for this.

An overview of the approach is shown in figure 5.2. Five variants of the training part *TIMIT* were generated first. MFCC features were then extracted from these new datasets and used to train models.
Similarly, MFCCs are extracted from the *TIMIT* Test set and from the *ACAP* data set. The previously trained models are used to recognize phonemes on these test datasets. Viterbi decoding can then be used to generate phoneme sequences. Finally, the results are evaluated.

In order to make the training data more "song-like", several variants of this dataset were developed. Table 5.1 shows an overview over the five datasets generated from

TIMIT using three modifications. Dataset N is the original *TIMIT* training set. For dataset P, four of the eight blocks of *TIMIT* were pitch-shifted. For dataset T, five blocks were time-stretched and vibrato was applied to two of them. In dataset TP, the same is done, except with additional pitch-shifting. Finally, dataset M contains a mix of these modified blocks.
In detail, the modifications were performed in the following way:

Time stretching For time stretching, the phase vocoder from [139], which is an implementation of the Flanagan/Dolson phase vocoder [140][141], was used. This algorithm works by first performing a Short-Time Fourier Transform (STFT) on the signal and then resampling the frames to a different duration and performing the inverse Fourier transform.

As demonstrated in section **??**, time variations in singing are mainly performed on vowels and are often much longer than in speech. Therefore, the *TIMIT* annotations were used to only pick out the vowel segments from the utterances. They were modified randomly to a duration between 5 and 100 times the original duration and then re-inserted into the utterance. This effectively leads to more vowel frames in the training data, but since there is already a large amount of instances for each phoneme in the original training data, the effects of this imbalance should be negligible.

Pitch shifting To pitch-shift the signal, code from the freely available Matlab tool *AutoTune Toy*[142], which also implements a phase vocoder, was used. In this case, the fundamental frequency is first detected automatically. The signal is then stretched or expanded to obtain the new pitch and interpolated to retain the original duration.

Using the *TIMIT* annotations, utterances were split up into individual words, and then a pitch-shifted version of each word was generated and the results were concatenated. Pitches are randomly selected from a range between 60% and 120% of the original pitch.

Vibrato The code for vibrato generation was also taken from *AutoTune Toy*. It functions by generating a sine curve and using this as the trajectory for the pitch shifting algorithm mentioned above. A sine of amplitude 0.2 and frequency $6Hz$ was used.

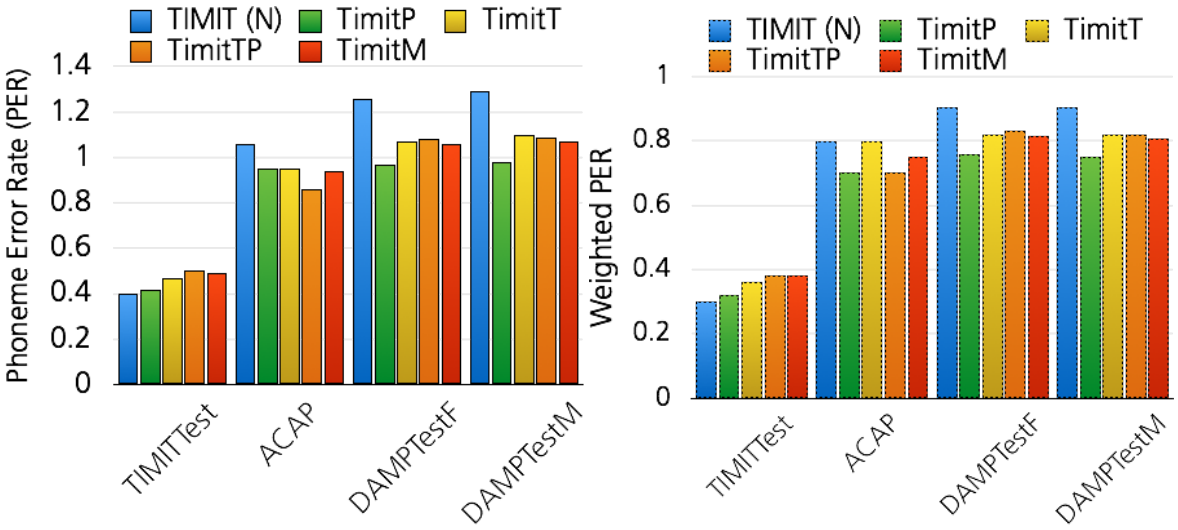
In singing, vibrato is commonly done on long sounds, which are usually vowels. Since spoken vowels are usually very short, vibrato cannot be perceived on

	N	**P**	**T**	**TP**	**M**
DR1	N	N	N	N	N
DR2	N	N	N	N	N
DR3	N	N	N	N	P
DR4	N	N	T	TP	TV
DR5	N	P	T	TP	TPV
DR6	N	P	T	TP	TV
DR7	N	P	TV	TPV	P
DR8	N	P	TV	TPV	TPV

Table 5.1: The five TIMIT variants that were used for training (rows are TIMIT blocks, columns are the five datasets). Symbols: N - Unmodified; P - Pitch-shifted; T - Time-stretched; V - Vibrato

them very well. Therefore, vibrato was only added when time stretching was also applied. Vibrato was then added to the extracted and stretched vowels.

The approach was tested on the various test data sets - namely, the test part of *TIMIT*, the *TIMIT* data set, and the two test sets chosen from the *DAMP* data set. Figure 5.3 shows the results for the DNN models. As it demonstrates, results for singing are generally worse than for speech. The base result for singing is a Weighted Phoneme Error Rate of .8 for *ACAP*, and of .9 for both *DAMPTestF* and *DAMPTestM* (model trained on the original TIMIT dataset). When comparing the models trained on the various *TIMIT* modifications, an improvement is observed for all variants. In contrast, none of the modifications improved the result on the speech data at all. The base result here .3. (see previous section). This makes sense since all of the modifications make the training data less similar to the test data set.
On singing, the Weighted Phoneme Error rate falls by .1 to .15 when using models trained on the pitch-shifted data set (*TimitP*), and by up to .08 when using the time-stretched training data (*TimitT*). The improvements for the corpora with both improvements lie in between. Vibrato does not appear to have a strong influence on the result. The higher error rates for *DAMPTest* can, again, be explained with the fact that this data set has more variation in audio and singing quality.
Pitch shifting might have a stronger effect on the recognition result because it is a stronger modification in the sense that it generates actual new feature values. In contrast, time stretching mostly generates new frames with values similar to the ones in the original data set (and only in between those). Additionally, pitch shifting may introduce more variety that is closer to sung sounds because singers do not usually shape long vowels just by stretching out their short versions.



(a) Phoneme error rate (b) Weighted phoneme error rate

Figure 5.3: Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT* and augmented versions thereof.

Nevertheless, it is interesting to see that both pitch shifting and time stretching improve the result for sung phoneme recognition. This indicates that more variety in the training data is a step in the right direction. However, there is still a lot of room for improvement.

5.3 Phoneme recognition using models trained on a-capella singing

In this section, the most salient tested approach for phoneme recognition in singing is presented. For this method, acoustic models were trained on actual singing recordings. A large set of such recordings was already available from the *DAMP* corpus, but no annotations were included with them. Such annotations were generated automatically from text lyrics available on the internet. In the following subsections, this process is described in detail, some variants are tested, and experiments with these new models are presented.

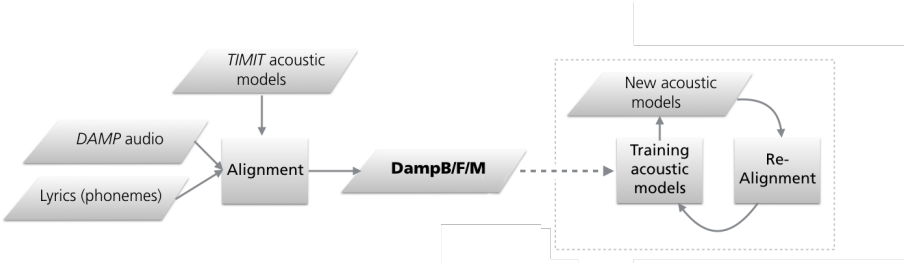


Figure 5.4: An overview of the alignment process. The right-hand part represents the optional bootstrapping.

5.3.1 Corpus construction

As mentioned above, no lyrics annotations are available for the *DAMP* data set, but the textual lyrics can be obtained from the *Smule Sing!* website[1]. All of them were English-language songs. These lyrics were mapped to their phonetic content using the CMU Pronouncing Dictionary[2] with some manual additions of unusual words. As with the other data sets, this dictionary has a phoneme set of 39 phonemes (also see appendix **??**).

An automatic alignment of these lyrics to the *DAMP* audio was then performed using the HMM acoustic models trained on *TIMIT* (see section 5.1). Viterbi alignment was performed on the word and phoneme levels using the HTK framework, using MFCCs and their deltas and double-deltas as features. This is the same principle of so-called "Forced Alignment" that is commonly used in Automatic Speech Recognition [143] (although it is usually done on shorter utterances).

Several different alignment strategies were carried out:

Monophones vs. Triphones Both monophone (i.e. one state per phoneme) and triphone (i.e. three states per triphone modeling the start, middle, and end phases) were tested. Since *TIMIT* only contains monophone annotations, this was done by first splitting the phoneme time frames evenly through three, and then retraining the *TIMIT* acoustic model and re-aligning the data set (with the assumption that the transitions between the triphone states would be "pulled" to the correct times).

One-pass alignment vs. Bootstrapping On top of a one-pass alignment using the Viterbi algorithm, bootstrapping the acoustic models to improve the alignment

[1] `http://www.smule.com/songs`
[2] `http://www.speech.cs.cmu.edu/cgi-bin/cmudict`

was also. To clarify: The alignment was first performed on the *DAMP* data sets using the *TIMIT* models described above, then acoustic models were trained on the resulting phoneme annotations. Then, those models were used to re-align the *DAMP* data, which was again used to train another model. This was done over three iterations.

A modified version of the alignment algorithm was used for doing alignment with the models trained on the **DAMP** data sets. This approach is based on doing Dynamic Time Warping on the generated phoneme posteriorgrams without punishing very long states. This is similar to the approach described in **??**.

A graphical overview of the alignment process is given in figure 5.4.
Of course, errors cannot be avoided when doing automatic forced alignment. All in all, there were now four combinations of these strategies, which were compared. The next section describes how this alignment procedure was validated and what strategies performed best.
Since there is usually a large number of recordings of the same song, an approach using this information to improve the alignment results was also considered, e.g. by averaging time stamps over the alignments of several recordings. This was not done in this work because recordings tend to have different offsets from the beginning (i.e. silence in the beginning), and the singers also do not necessarily pronounce phonemes at the same time. This might be an avenue for future research, though.

5.3.2 Alignment validation

The alignment approach that was used to create the new *DAMP*-based data sets was first tested on the *ACAP* data set. To recap: This approach employs models trained on the *TIMIT* speech corpus, which are used for Viterbi alignment of the known phonemes to the singing. The result of this is then compared to the manual annotations by calculating the difference between each expected and predicted phoneme transition. As mentioned in 5.1 and shown in figure 5.5, the mean alignment error for this first approach is .16 seconds for the triphone case, and .17 for monophones.

Various models trained on the new **DampB**, **DampF**, and **DampM** training data sets were then tested for the same task. The results are also shown in figure 5.5.
Models trained on the monophonic alignments of **DampB** perform slightly better at this task with mean errors of 0.15 seconds. For the gender-specific models, the error was only calculated for the songs of the matching gender in *ACAP*, resulting in the

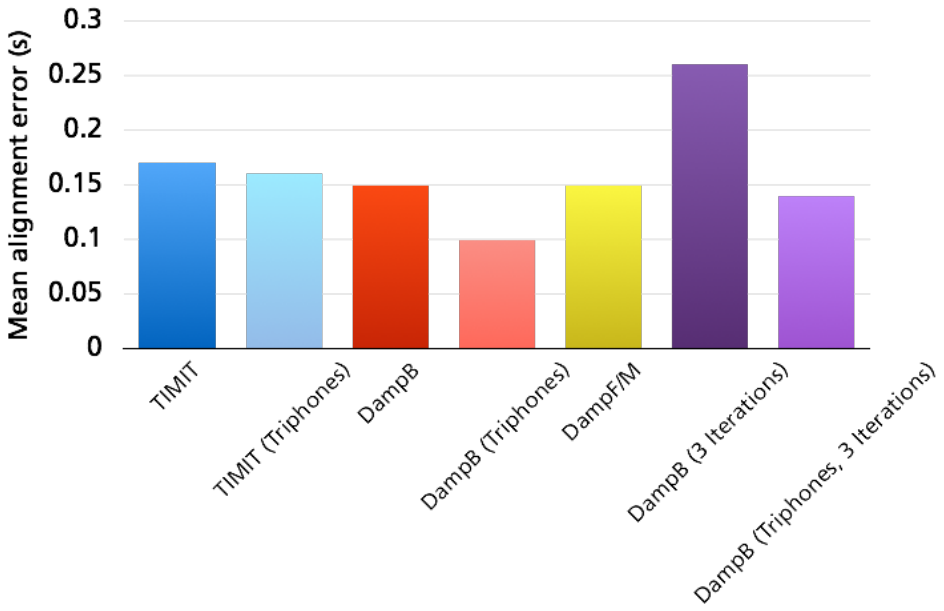


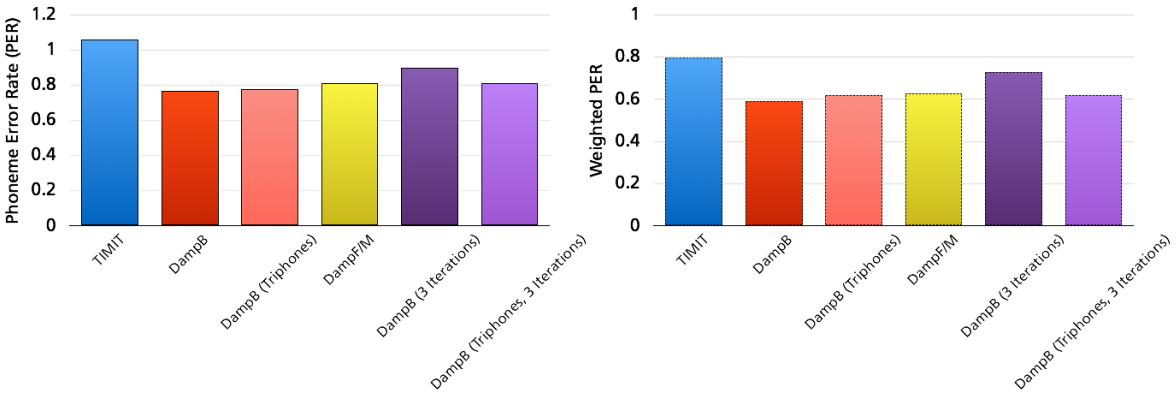
Figure 5.5: Mean alignment error in seconds on the *ACAP* data set. *TIMIT* shows the result for the same models used for aligning the new *DAMP*-based data sets.

same average value. Since the gender-specific models do not produce better results, the other experiments were conducted on the mixed training set only. The triphone version of **DampB** performs even better at alignment, with a mean error of 0.1. This might happen because dedicatedly training the model for the start and end parts of phonemes might make the alignment approach more accurate at finding start and end points.

Finally, a model trained on **DampB** over three iterations was also tested for alignment. This model performs much worse at this task with a mean alignment error of 0.26. This might happen because errors in the original alignment of the phonemes may become amplified over these iterations. The effect is not as pronounced for the triphone version, but it is still present.

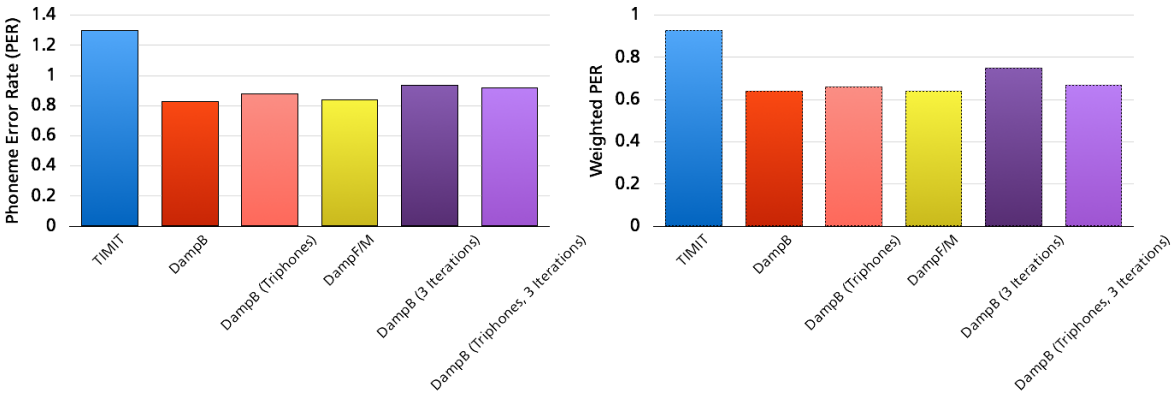
5.3.3 Phoneme recognition

After validating the alignment strategy, phoneme recognition experiments were performed on both the *ACAP* and the *DampTest* corpora. This was possible even though there are no manual annotations for the *DampTest* sets because the expected phonemes are available from the textual lyrics. Again, the phoneme error rate and the weighted phoneme error rate were used as evaluation measures (see **??**).



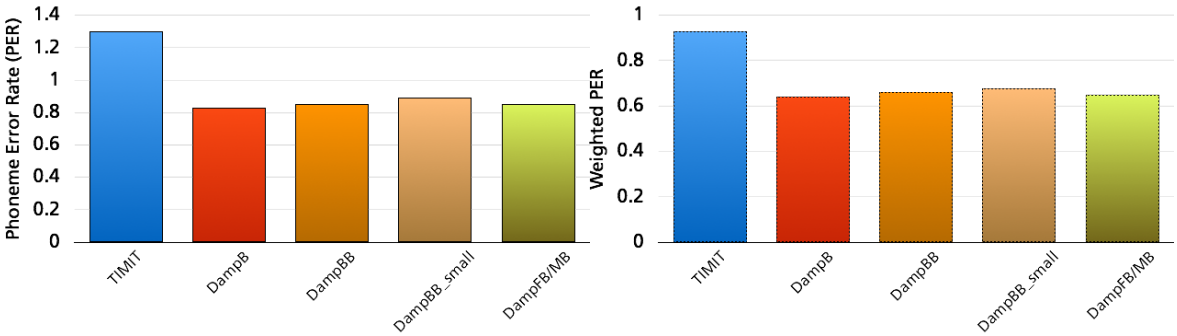
(a) Phoneme error rate (b) Weighted phoneme error rate

Figure 5.6: Mean phoneme recognition results on the *ACAP* data set using acoustic models trained on *Timit* and the new *DAMP*-based data sets.



(a) Phoneme error rate (b) Weighted phoneme error rate

Figure 5.7: Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets.



(a) Phoneme error rate (b) Weighted phoneme error rate

Figure 5.8: Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets.

The results for *ACAP* are shown in figure 5.6. In general, models trained on *DampB* performed much better at phoneme recognition than those trained on *TIMIT*. Compared to these speech-based models, the phoneme error rate falls from 1.06 to 0.77, while the weighted phoneme error rate falls from 0.8 to 0.59. As can be seen from both evaluation measures, using triphone alignments instead of monophones does not improve the results in this case, contrasting with the better alignment results. This might happen because more classes cause more confusion in the model, even though the triphone results were downmapped to monophones for calculating the evaluation measures. As in the alignment results, using gender-specific models does not provide an advantage over a mixed model. (Results for the gender-specific models were only evaluated on songs of the matching gender).
As already seen in the alignment validation results, training models on **DampB** over three iterations actually degrades the result. Again, this might happens because phoneme alignment errors become amplified in this way. Interestingly, the effect is not as strong for the triphone models, perhaps because the three classes per phoneme help to alleviate each other's errors.

The results for the same procedure on the *DampTest* sets are shown in figure 5.7. Results over *DampTestF* and *DampTestM* were averaged. The same general trend can be observed for these results: The phoneme error rate falls from 1.3 to 0.83 when compared to models trained on *TIMIT*, with the weighted phoneme error rate decreasing from 0.93 to 0.64. Using triphones does not contribute to the result, and neither does the three-iteration bootstrapping process for training acoustic models. As in the *ACAP* results, not even the gender-specific models improve the result. This effect might occur because the range of pitch and expressions is much wider in singing than in speech, and therefore gender-specific models may not actually learn as much added helpful information. Other experiments indicated that gender-specific models also do not improve the results when using triphones or three alignment iterations as with the mixed-gender training data.

Going forward, monophonic models trained on mixed-gender data (i.e. *DampB*) appear to be the best and simplest solution. To gain insight into the role of the composition of the training data set, more experiments were conducted with the variants of *DampB* described in section 4.2.3. The results are shown in figure 5.8.

When using the smaller, more balanced version of *DampB* (*DampBB*), the results

become somewhat worse, but not much, with a phoneme error rate of .85, and a weighted phoneme error rate of 0.66. This is particularly interesting because this data set is only 4% the size of the bigger one and training is therefore much faster. With the smallest data set which is only half the size of *DampBB*, the change is similar: The Phoneme Error Rate rises to .89, the Weighted Phoneme Error Rate to .68. Since this data set has roughly the same amount of phonemes as *TIMIT*, this proves that the improvement is actually caused by the acoustic properties of the training data, rather than just the larger amount of data. (Of course, this factor also contributes). The reduced versions of *DampF* and *DampM* were tested as well, but once again, they do not provide an improvement over the mixed gender model.

5.3.4 Error sources

Of course, with an automatic alignment algorithm like this, errors cannot be avoided. To acquire a clearer picture of the reasons for the various misalignments, the audio data where they occurred was analyzed more closely. Some sources of error repeatedly stuck out:

Unclear enunciation Some singers pronounced words very unclearly, often focusing more on musical performance than on the lyrics.

Accents Some singers sung with an accent, either their natural one or imitating the one used by the original singer of the song.

Young children's voices Some recordings were performed by young children.

Background music Some singers had the original song with the original singing running in the background.

Speaking in breaks Some singers spoke in the musical breaks.

Problems in audio quality Some recordings had qualitative problems, especially loudness clipping.

For most of these issues, more robust phoneme recognizers would be helpful. For others, the algorithm could be adapted to be robust to extraneous recognized phonemes (particularly for the speaking problem). If possible, a thorough manual check of the data would be very helpful as well.

5.4 Conclusion

In this chapter, new approaches for phoneme recognition in singing were described. As a starting point, DNN models were trained on the *TIMIT* speech corpus. For verification, these models were evaluated on the test section of *TIMIT*, resulting in a Phoneme Error Rate of .4 and a Weighted Phoneme Error Rate of .3. The results on the *ACAP* singing data set were much worse: A Phoneme Error Rate of .1.06 and a Weighted Phoneme Error Rate of .8, demonstrating the difficulty of phoneme recognition in singing as opposed to speech. Similarly, the Phoneme Error Rate was 1.28 on the *DampTest* data sets, with a Weighted Phoneme Error Rate of .9. Generally, results on the *DampTest* sets are worse than on the *ACAP* test set, which presumably happens because the *DampTest* sets are performed by amateurs, whose enunciation is not as clear as that of the professional singers in the *ACAP* set and who may not always sing the correct lyrics, because the recording quality varies much more, and because the annotations may contain errors due to the automatic phoneme alignment (as opposed to the more realiable manual annotations of *ACAP*).

In order to make the models better adapted to singing, acoustic modifications were performed on the *TIMIT* training data - namely, pitch shifting and time stretching. This increases the Phoneme Error Rate on speech test data (*TIMITTest*, but improves them on sung data. Pitch shifting appears to have a stronger effect than time stretching, which might happen because this modification results in actual changed feature data, whereas time stretching mainly produces more frames with feature values similar or in between the existing ones. On the *ACAP* test data, the lowest Phoneme Error Rate is .95, and the lowest Weighted Phoneme Error Rate is .7 (with the pitch-shifted models). On *DampTest*, those values are decreased to .98 and .76 respectively.

The best-adapted models for recognizing phonemes in singing should be those trained on actual singing data. Unfortunately, there are no big, annotated singing data sets like those for speech. For this reason, the *DAMP* data set, which contains thousands of recordings of unaccompanied amateur singing, was selected, and the text lyrics were obtained from the internet. Then, various strategies for aligning the phonetic content of these lyrics to the audio were tested. In the simplest algorithm, HMM models trained on *TIMIT* were used for Viterbi alignment. This also turned out to be one of the most effective algorithms for the alignment process necessary for constructing this

new singing data set.

The resulting annotations, together with the singing audio data, were then used to train new DNN models for phoneme recognition. Using 6000 of these recordings of singers of both genders, the Phoneme Error Rate was lowered to .77 on the *ACAP* test set, and the Weighted Phoneme Error Rate was lowered to .59. On the *DampTest* sets, those values decreased to .83 and .64 respectively. Neither employing triphones instead of monophones nor training gender-specific models provided additional advantages. Iterating the alignment process decreased the result, possibly because errors in the original alignment become amplified over these iterations.

The influence of training set size was also investigated on the *DampTest* data. Using a phonetically balanced subset of *DAMP* that was just 4% of the size of the originally selected training set worsened the result by only 2 percent points (both in Phoneme Error Rate and Weighted Phoneme Error Rate). Using an even smaller data set that is roughly the size of *TIMIT* increased the Phoneme Error Rate only by a further 4 percent points (and the Weighted Phoneme Error Rate by 2), thereby proving that the better recognition results are caused by the sung content rather than just the size of the training data.

A manual error analysis was performed on cases where the phoneme recognition failed. Errors are frequently caused by linguistically or acoustically difficult segments in the test data, such as accents, children's voices, background music, or additional speaking.

Comparing these results to the state of the art is difficult because of the different testing data used. Considering raw numbers, it can be assumed that the best results are in the range of the best state of the art results, for example those by Mesaros et al. [101] (see section 3.2). In contrast to these approaches, no involved post-processing is employed. In the future, integrating such steps as singer adaptation and language modeling would in all probability serve to improve the approach even more.

6 Sung language identification

Singing language identification is the task of automatically determining the language of a sung recording. For this task, two algorithms were developed: One based on i-Vector extraction, and one based on the statistics of phoneme posteriorgrams. They will be described in detail in this section.
In both cases, the *NIST2003LRE* and *OGIMultilang* corpora were used for testing the algorithms on speech, and the *YTAcap* data set was used for singing (see chapter **??**).

6.1 Language identification in singing using i-vectors

As described in section **??**, i-Vector extraction is a feature dimension reduction algorithm that was originally developed for speaker recognition, but has since then been employed successfully for other tasks, including language identification. After the publication of this approach for i-Vector extraction for singing language identification, they also became used for other MIR tasks, such as artist recognition and similarity calculation [144] [145].

6.2 Proposed system

Figure 6.1 shows a rough overview over the i-vector classification system.

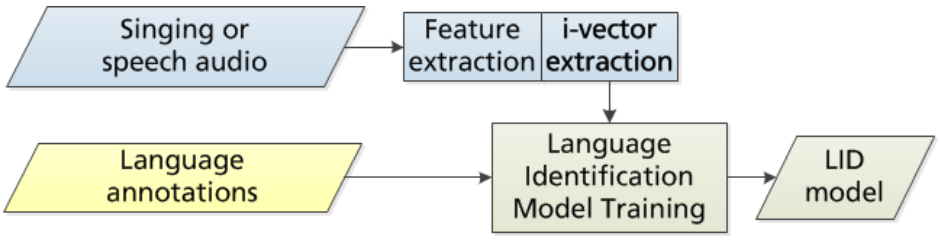


Figure 6.1: Overview of the process for language identification using i-vector extraction.

A number of features were extracted from each audio file. Table 6.1 shows an overview over the various configurations used in training.

Table 6.1: Feature configurations used in training.

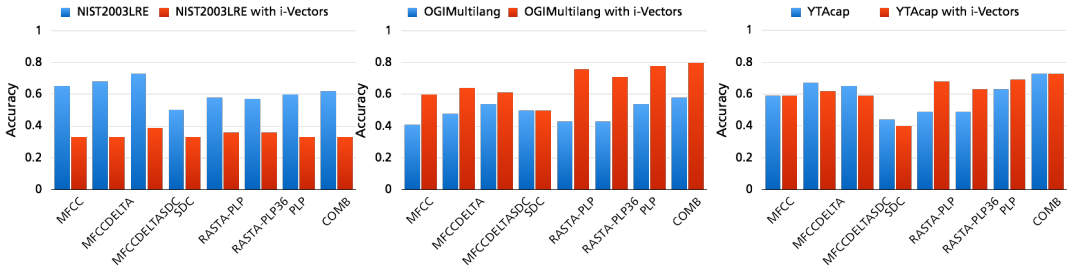
Name	Description	Dimensions
MFCC	MFCC, 20 coefficients	20
MFCCDELTA	MFCC, 20 coefficients, deltas and double deltas	60
MFCCDELTASDC	MFCCDELTA+SDC	117
SDC	SDC with configuration $7-1-3-7$	91
RASTA-PLP	PLP with RASTA processing, model order 13 with deltas and double deltas	39
RASTA-PLP36	PLP with RASTA processing, model order 36 with deltas and double deltas	96
PLP	PLP without RASTA processing, model order 13 deltas and double deltas	39
COMB	PLP+MFCCDELTA	99

For classification, Multi-Layer Perceptrons (MLPs) and Support Vector Machines (SVMs) were tested. The MLPs were fixed at three layers, with the middle layer having a dimension of 256. Additional layers did not seem to improve the result. A larger middle layer only improved it slightly. The SVM parameters were determined using a grid-search. For each of those classifiers and data sets, all combinations of the features listed in table 6.1 were tested directly and with i-vector processing. All results were obtained using five-fold cross validation.

6.2.1 Experiments with known speakers

In the first experiment, MLP and SVM models were trained on randomly selected folds of the training data sets. This means that recordings by the same speaker are spread out between the training and test data sets. In theory, i-vectors could be particularly susceptible to capturing speaker characteristics instead of language characteristics, leading to evaluation results that may not be representative of results for unknown speakers. However, this effect is often ignored in literature [146], and an argument can be made that partially training models on speaker properties is realistic for some use cases (i.e. when many of the expected speakers for each language are already known).

The results for the MLP training on the *NIST2003LRE*, *OGIMultilang*, and *YTAcap* data sets are shown in figure 6.2.
As shown in figure 6.2a, the MLP did not produce good results on the *NIST2003LRE* database for any of the feature combinations. *NIST2003LRE* is the smallest of the data sets by a large margin. Since a relatively high-dimensional model was used, this



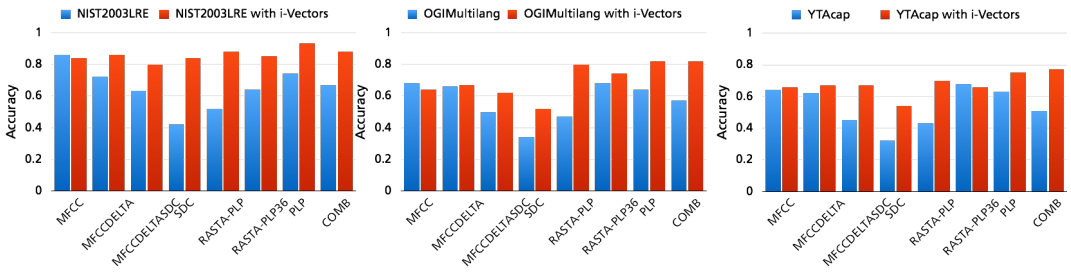
(a) NIST2003LRE (b) OGIMultilang (c) YTAcap

Figure 6.2: Results using MLP models on all three language identification data sets, with or without i-vector processing.

is probably a case of overtraining. The i-vector processing step reduces the training data even further, thus aggravating the problem.

The *OGIMultilang* data set contains roughly 4 times as much data as the *NIST2003LRE* set. With enough data, training an MLP classifier works a lot better. Without i-vector processing, this approach still only reach about 52% accuracy. i-Vector extraction improves the system massively. The best feature configurations are RASTA-PLP (82%), PLP (80%), and COMB (80%).

As with all other experiments, the task becomes harder when attempted on singing data. The results on the *YTAcap* data set are somewhat worse than those on *OGIMultilang*, even though they contain a similar amount of data. The best result without i-vector extraction is still obtained using the COMB feature configuration at 56%. Similar to the *OGIMultilang* experiment, i-vector extraction yields a large improvement. COMB remains the best configuration, now at 77%.



(a) NIST2003LRE (b) OGIMultilang (c) YTAcap

Figure 6.3: Results using SVM models on all three language identification data sets, with or without i-vector processing, with speakers shared between training and test sets.

Figure 6.3 shows the results for the SVM models trained on the same data sets. In general, these models appear to be able to capture the language boundaries better.

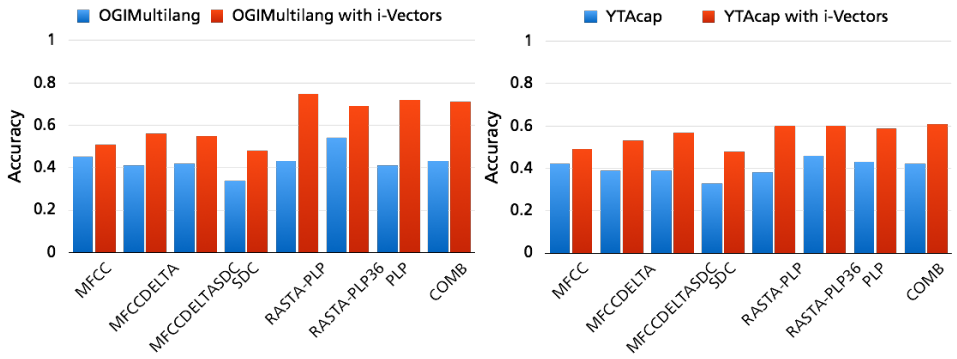
In contrast to the MLP experiment, SVMs produced good results on the *NIST2003LRE* data set for all of the features. They seem to be able to discriminate very well on this small, clean data set. The best result with i-vector processing is 86% for MFCC features. When using i-vectors, 93% are achieved with PLP features. This may, in fact, be close to the upper bound for the classification here, which might be caused by annotation errors or ambiguous data.
The *OGIMultilang* corpus is bigger and more varied than the *NIST2003LRE* corpus, which makes it harder to classify. As shown, the high-dimensional pure features did not perform as well, with a maximum of 68% for MFCCs and RASTA-PLPs with 36 coefficients. Using i-vector extraction improved the result by a large margin. Feature-wise, PLPs without RASTA processing seem to work best at a result of 82%. MFCC and SDC features did not work quite as well, but did not hurt the result either when combined with PLPs (COMB result). It is interesting to see that the i-vector extraction decreased the results for MFCCs, the feature that worked best without it.
Similar to the *OGIMultilang* corpus, the *YTAcap* corpus provides very complex and varied data. The same effects occur on the direct feature training here, too: RASTA-PLPs with 36 coefficients provide the best results, but the accuracy is not very high at 68%. i-vector extraction once again serves to improve the result. The highest results when using i-vector extraction are 75% when using PLP without RASTA processing or 77% for the COMB configuration.

6.2.2 Experiments with unknown speakers

In order to find out what influence the speaker characteristics had on the result, the same experiments were then repeated with training and evaluation sets that strictly separated speakers. This experiment was not performed for the *NIST2003LRE* corpus because no speaker information is available for it. Since the SVM models performed much better in the previous experiment, only these models were tested.

The results are shown in figure 6.4. In general, all configurations performed worse, indicating that some of the characteristics learned by the models come from the speakers rather than the languages. Apart from this, the general trends for the features remain the same, and i-vector extraction still improves the over-all results.

On the *OGIMultilang* corpus, the best result is still obtained with RASTA-PLP features and i-vector processing, but the accuracy falls by around 8 percent points to



(a) OGIMultilang (b) YTAcap

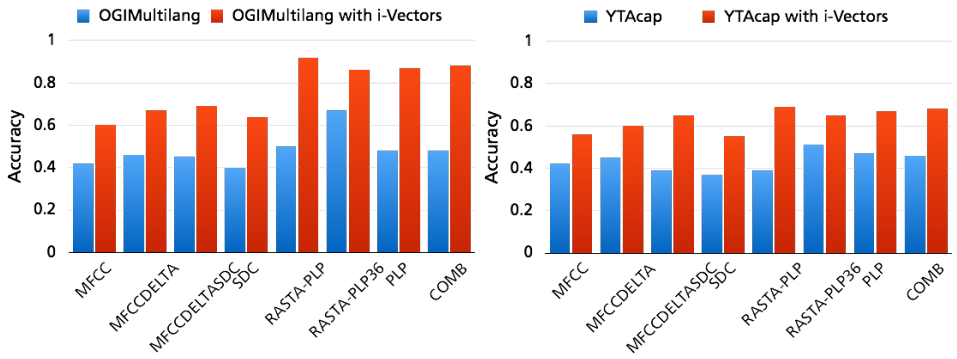
Figure 6.4: Results using SVM models on all three language identification data sets, with or without i-vector processing, with speakers separated between training and test sets.

75%. On *YTAcap*, the effect is even worse: From an accuracy of 77% with the mixed condition, the result decreases to 61% for the separated condition. The reason for this is probably the wider signal variety in singing as opposed to speech; additionally, *YTAcap* also possesses a wider range of recording conditions than the controlled telephone conditions of *OGIMultilang*. Arguably, the solution for this effect would be the use of larger training data sets, which would be able to cover these acoustic and performance conditions better. Conversely, as the previous results show, the approach functions better when an application scenario can be limited to a range of known speakers, or at least recording conditions (as in the *NIST2003LRE* experiment).

6.2.3 Experiments with utterances combined by speakers

All previous experiments were performed on relatively short utterances of a few seconds in duration. In many application scenarios, much more audio data is available to make a decision about the language. In particular, songs are usually a few minutes in length, and in many cases, only one result per document (=song) is required.
For this reason, results for the *YTAcap* data set were taken from the previous experiment and summed up for each song (and therefore also for each singer). For the *OGIMultilang* corpus, results for all utterances by the same speaker were aggregated in the same fashion, resulting in similar durations of audio. (Again, this experiment was not performed with the *NIST2003LRE* corpus due to the lack of speaker information).

The results are shown in figure 6.5. Overall, aggregation of multiple utterances by the same speakers seems to balance out some of the speaker-specific effects seen in the



(a) OGIMultilang (b) YTAcap

Figure 6.5: Document-wise results using SVM models, with or without i-vector processing.

previous experiment. Taking more acoustic information into account, the models are able to determine the language with higher accuracy.
On the *OGIMultilang* corpus, the result is even better than on the condition with known speakers. The best result rises from 75% for short utterances to 92% for the aggregated documents (both with the RASTA-PLP feature). On the *YTAcap* data sets, the aggregated result is 69% (compared to 60% for line segments).

As suggested in the previous section, the approach produces practically usable results when the problem can be narrowed down, e.g. to known speakers or recording conditions. As this experiment shows, useful results can also be obtained when longer sequences are available for analysis.

6.3 LID in singing using phoneme recognition posteriors

Another developed approach is based upon phoneme statistics derived from phoneme posteriorgrams. To obtain representative statistics for model training, relatively long observations are necessary, but, as described in the previous section, this is the case for many applications, for example when considering song material (e.g. songs of 3-4 minutes in duration). On the other hand, phoneme posteriorgrams need to be calculated for a number of other tasks, such as keyword spotting or lyrics-to-audio alignment.

An overview of the approach is shown in figure 6.6. Posteriorgrams were generated

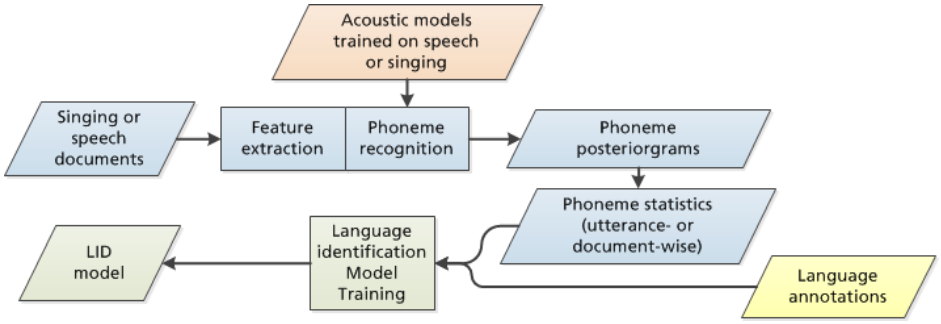


Figure 6.6: Overview of the process for language identification using phoneme statistics.

on the test data sets *YTAcap* and *OGIMultilang* using the acoustic models trained on the *TIMIT* speech data set and on the *DAMP* singing data set as described in section 5.3. To facilitate the following language identification, phoneme statistics were then calculated in two different ways:

Utterance-wise statistics Means and variances of the phoneme likelihoods over each utterance were calculated (or, in the case of *YTAcap*, over each song segment). For further training, the resulting vectors for each speaker/song (=document) were used as a combined feature matrix. As a result, no overlap of speakers/songs was possible between the training and test sets.

Document-wise statistics Mean and variances of the phoneme likelihoods over whole songs or sets of utterances of a single speaker were calculated. This resulted in just two feature vectors per document (one for the means, one for the variances).

Naturally, relatively long recordings are necessary to produce salient statistics. For this reason, the aggregation by speaker/song was done in both cases rather than treating each utterance separately.

Then, Support Vector Machine (SVM) models were trained on the calculated statistics in both variants with the three languages as annotations. Unknown song/speaker documents could then be subjected to the whole process and classified by language. Again, all results in the were obtained using 5-fold cross-validation - i.e., SVMs were trained on 4/5 of each corpus, then the remaining 1/5 was classified with the model. This was done 5 times until each song/speaker document had been classified.

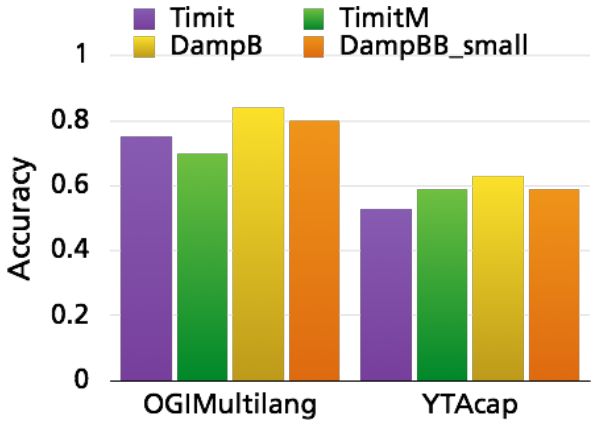


Figure 6.7: Results using document-wise phoneme statistics generated with various acoustic models.

6.3.1 Language identification using document-wise phoneme statistics

In the first experiments, SVM classifiers were trained on the document-wise phoneme statistics, and classification was also performed on a document-wise basis (i.e., only one mean and one variance vector per document). The results are shown in figure 6.7 in terms of accuracy (average retrieval when all documents are classified into exactly one language).

On the singing test set, results are worst when using acoustic models trained on *TIMIT* at just 53% accuracy, and become better when using the model trained on the *TIMIT* variant modified for singing or the small selection of the singing training set (59% each). The best result is achieved when the models are trained on the full singing data set at 63% accuracy.

Surprisingly, the results on the *OGIMultilang* corpus also improve from 75% with the *TIMIT* models to 84% using the *DampB* models. Since *TIMIT* is a very "clean" data set, training on the song corpus might provide some more phonetic variety, acting as a sort of data augmentation. This could be especially important in this context where phonemes are recognized in three different languages.

On both corpora, there is no noticeable bias of the confusion matrix - i.e., the confusions are spread out evenly. This is particularly interesting when considering that the acoustic models were trained on English speech or singing only.

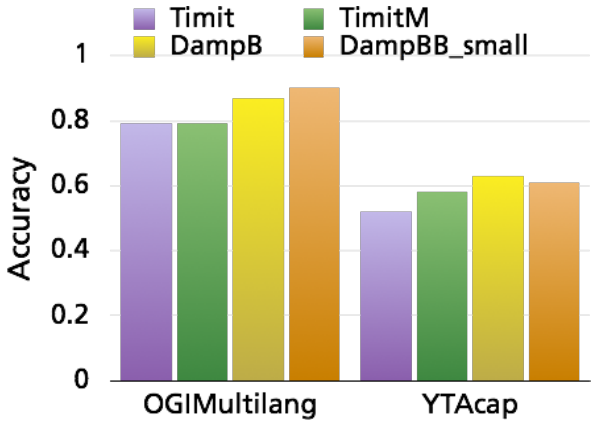


Figure 6.8: Results using utterance-wise phoneme statistics generated with various acoustic models.

6.3.2 Language identification using utterance-wise phoneme statistics

Next, language identification was performed with models trained on the statistics of each utterance contained in the document. The recognition process is still performed on the whole document. The results are reported in figure 6.8.
Phoneme statistics may not be as representative when computed on shorter inputs, but they may provide more information for the backend model training when utilized as a combined feature matrix for a longer document. The results on singing improve slightly to 63% with the acoustic model trained on the small singing corpus (*DampBB_small*) and decrease slightly for the *DampB* model (61%). However, on the speech corpus, the best result rises to 90%.

6.3.3 For comparison: Results for the i-vector approach

For comparison, models from the previous approach were also trained on the same time scales. I-vectors were calculated on the utterance- or the document-wise scale. This was done for PLP and MFCC features. The resulting i-vectors were then used to train SVMs in the same manner as in the previous experiments. (The difference here is that the models are already trained on the aggregated i-vectors, either with those for a whole document or with all i-vectors of the utterances constituting each document aggregated). The results are shown in figure 6.9.
The best result obtained on *YTAcap* corpus is 68% accuracy. This is only 5 percent points higher than the presented approach, which is much easier to implement. On the

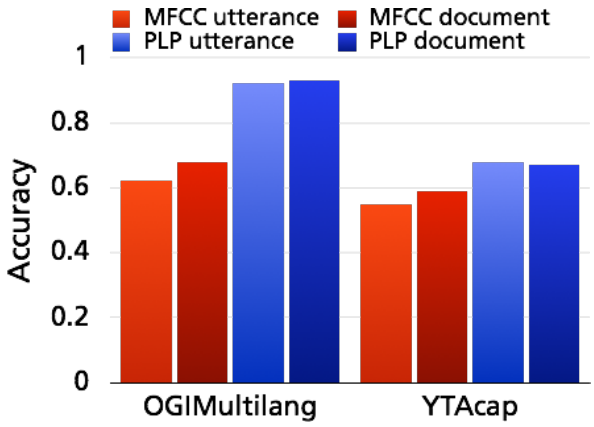


Figure 6.9: Results using utterance- and document-wise i-vectors calculated on PLP and MFCC features.

OGIMultilang data set, the difference is only 3 percent points (93%). Of course, the advantage of the i-vector approach is that it can also be performed on much shorter inputs.

6.4 Conclusion

In this section, two approaches to singing language identification were presented: One based on i-vector processing of audio features, and one based on phoneme statistics generated from posteriorgrams. In both cases, machine learning models were trained on the resulting data.

In the first approach, PLP, MFCC, and SDC features were extracted from audio data, and then run through an i-vector extractor. The generated i-vectors were then used as inputs for MLP and SVM training. The basic idea behind the i-vector approach is the removal of speaker- and channel-dependent components of the signal. This effectively reduces irrelevance to the language identification tasks and also reduces the amount of training data massively.
The smallest data set is the *NIST2003LRE* corpus. No feature configuration achieved good results when using the MLP backend. In this case, the small size of the corpus may lead to overtraining. I-vector processing only amplified this problem by reducing the amount of data even further. The SVM backend, however, produced good results of up to 93% for PLP features with i-vector extraction.

The *OGIMultilang* corpus is a much bigger speech corpus. Training without i-vector extraction did not work well for any feature configuration. The best accuracy for this scenario was 68%. Results of up to 83% were achieved with i-vector processing. There does not seem to be a large difference between SVM and MLP training, with SVM having just a slight advantage.
Language identification for singing was expected to be a harder task than for speech due to the factors described in section 3.1. The results on the *YTAcap* corpus turned out to be somewhat worse than those for the *OGIMultilang* corpus, which is of similar size. As on *OGIMultilang*, i-vector extraction improved the results. In this case, the accuracy improved from 63% to 73%.
The same experiment was performed again, this time with no speaker overlap between training and test sets. The results fell significantly, indicating speaker influence on the model training. In a third experiment, the results were aggregated into documents by each speaker, which again led to improved results. The best accuracy on *OGIMultilang* is 92%, while on *YTAcap*, it is 69%. Both experiments demonstrate that useful results can be obtained when limiting the task, e.g. by training on a set of known speakers or recording conditions, or by analyzing documents of longer durations. Alternatively, a wider range of speakers in the training data could lead to models that generalize better.
Overall, i-vector extraction reduces irrelevance in the training data and there leads to a more effective training. As additional benefits, the training process itself is much faster and less memory is used due to its data reduction properties. Most of the state-of-the-art approaches are based on PPRLM, which requires phoneme-wise annotations and a highly complex recognition system, using both acoustic and language models. In this respect, this system is easier to implement and merely requires language annotations.

The second presented method is a completely new language identification approach for singing. It is based on the output of various acoustic models, from which statistics were generated and SVM models were trained. In contrast to similar approaches for speech, no voice tokenization is performed. Since phoneme recognition on singing is not always reliable, the statistics are calculated directly on the phoneme posteriorgrams, although this does not take any temporal information into account. The acoustic models were trained only on English-language material (speech and singing); it would be interesting to test this with multi-language training data. Due to the statistics-based nature of the approach, it is not suited for language identification of very short audio recordings.

The accuracy of the result for singing is somewhat worse than the results obtained with the i-vector based approach. However, this new approach is much easier to implement and the feature vectors are shorter. For many applications, such posteriors need to be extracted anyway and can then efficiently be used for language identification when long observations are available. The best accuracy of 63% is obtained with acoustic models trained on the *DampB* singing corpus.
Interestingly, the best result on the *OGIMultilang* speech corpus is also obtained with these acoustic models (and is only 3 percent points below the one obtained with the i-vector approach). This possibly happens because the singing corpora provide a wider range of phoneme articulations. It would be interesting to try out these acoustic models for other phoneme recognition tasks on speech where robustness to varied pronunciations is a concern.

7 Sung keyword spotting

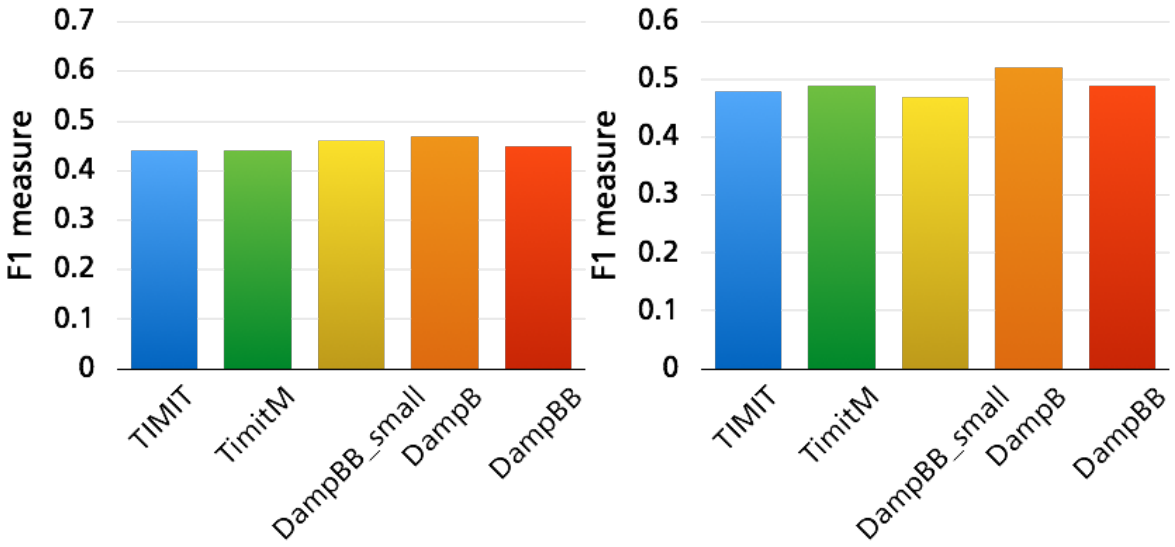
Keyword spotting is another task for which the new acoustic models described in chapter 5 were employed. A keyword-filler HMM algorithm was selected due to its independence on phoneme durations, which is a condition that cannot be fulfilled easily in singing. As described in section **??**, keyword-filler HMMs consist of two sub-HMMs: One to model the keyword and one to model everything else (=filler). The keyword HMM has a simple left-to-right topology with one state per keyword phoneme. The filler HMM is a fully connected loop of all phonemes. When the Viterbi path with the highest likelihood passes through the keyword HMM rather than the filler loop, the keyword is detected. Keyword detection was performed on whole songs, which is a realistic assumption for many practical applications. The *ACAP* and *DampTest* data sets were used for evaluation with the keyword set described in section **??**. Song-wise F_1 measures were calculated for evaluation.

7.1 Keyword spotting using keyword-filler HMMs

7.1.1 Comparison of acoustic models

Phoneme posteriorgrams were generated with the various acoustic models described in section 5.3. The results in terms of F_1 measure across the whole *DampTest* sets are shown in figure 7.1a. Figure **??** show the results of the same experiment on the small *ACAP* data set.

Across all keywords, a document-wise F_1 measure of 0.44 is obtained using the posteriorgrams generated with the *TIMIT* model on the *DampTest* data sets. This result remains the same for the *TimitM* models trained on "songified" speech. In this experiment, using models trained on the *DAMP*-based singing data sets only improves the results slightly, with F_1 measures of 0.47 for the *DampB* model, and 0.46 with the much smaller *DampBB_small* model. Surprisingly, in this case, the model trained on the medium-size balanced data set *DampBB* performs a little worse than the smallest one; however, this might just be due to some statistical fluctuation. In general, results



(a) *DampTest* (b) *ACAP*

Figure 7.1: F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models.

on these test data sets are somewhat inconclusive. There are several reasons for this: First, the annotations were generated automatically and the keywords were picked from the aligned lyrics. However, the singers may not always perform them or might pronounce them wrong. Additionally, the keyword approach can be tuned easily for high recall; then, the precision becomes the deciding factor for F_1 calculation. Considering the size of the data set, keyword occurrences are relatively rare, which makes obtaining a high precision more difficult and blurs the F_1 measures between approaches.

On the hand-annotated *ACAP* test set, the differences are someqhat more pronounced. The F_1 measure is 0.48 for the *TIMIT* model, and rises to 0.52 with the *DampB* model. The *TimitM* and *DampBB* models both produce F_1 measures of 0.49. The higher over-all values could be caused by the more accurate annotations or by the higher-quality singing. Additionally, the data set is much smaller with fewer occurrences of each keyword, which could emphasize both positive and negative tendencies in the detection.

In general, recalls are usually close to 1, and precisions often in the range of 0.2 to 0.5 (with much lower and higher outliers). For this reason, an approach that could exploit a configuration with high recalls and then discard unlikely occurrences could offer an improvement. This idea is explored further in section 7.2.

7.1.2 Gender-specific acoustic models

Keyword spotting was also performed on the posteriorgrams generated with the gender-dependent models trained on *DampF* and *DampM* (also described in section 5.3. The results are shown in figure 7.2.

In contrast to the phoneme recognition results from Experiment C, the gender-dependent models perform slightly better for keyword spotting than the mixed one of the same size, and almost as good as the one trained on much more data (*DampB*). The F_1 measures for the female test set are 0.48 for the *DampB* model, 0.45 for the *DampBB* model, and 0.46 for the *DampFB* model. For the male test set, they are 0.46 and 0.45 for the first two, and 0.46 for the *DampMB* model.

7.1.3 Individual analysis of keyword results

Figure 7.3 shows the individual F_1 measures for each keyword using the best model (*DampB*), ordered by their occurrence in the *DampTest* sets from high to low (i.e. number of songs which include the song). There appears to be a tendency for more frequent keywords to be detected more accurately. This happens because a high recall is often achievable, while the precision depends very much on the accuracy of the input posteriorgrams. The more frequent a keyword, the easier it also becomes to achieve a higher precision for it.

As shown in literature [147], the detection accuracy also depends on the length of the keyword: Keywords with more phonemes are usually easier to detect. This might explain the relative peak for “every”, in contrast to “eyes” or “world”. Since keyword detection systems tend to perform better for longer words and most of the keywords only have 3 or 4 phonemes, this result is especially interesting.

One potential source of error are sequences of phonemes that overlap with keywords, but are not included in the calculation of the precision. Words spelled the same were included, but split phrases or other spellings were not (e.g. “away” as part of “castaway” would be counted, but “a way” would not be counted as “away”). This might artificially lower the results and could be an avenue for future improvement. Additionally, only one pronunciation for each keyword was provided, but there may be several possible.

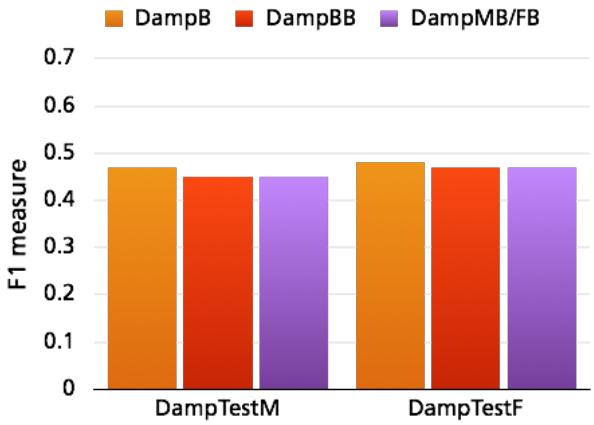


Figure 7.2: F_1 measures for keyword spotting results on the *DampTestM* and *DampTestF* data sets using mixed and gender-dependent models.

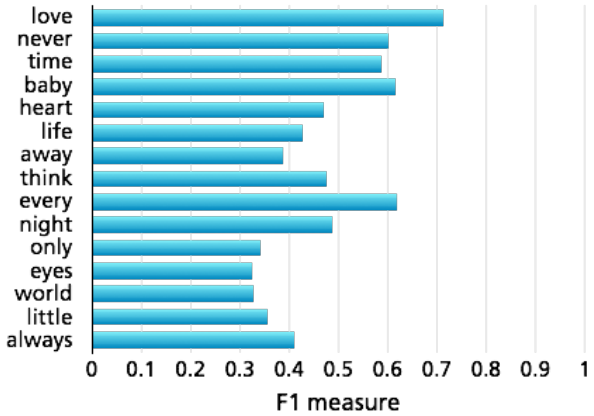


Figure 7.3: Individual F_1 measures for the results for each keyword, using the acoustic model trained on *DampB*.

7.2 Keyword spotting using duration-informed keyword-filler HMMs

7.2.1 Approach

As mentioned above, a high recall is easily achievable with the described approach, but the comparatively low precision decreases the over-all result. Therefore, using additional information to sort out false positives would be a helpful next step.
One such source of information are the durations of the detected phonemes. As shown in figure **??**, each phoneme in the *TIMIT* speech database has a fairly fixed duration. In singing, the vowels' durations vary a lot, but the consonants' are still quite predictable. Standard HMMs do not impose any restrictions on the state durations, resulting in a geometric distribution which does not correspond to naturally observed phoneme durations.
As first described in [148], introducing restrictions on state durations can improve the recognition results. In [149], Juang et al. present two basic approaches for duration modeling in HMMs: Internal duration modeling and Post-processor duration modeling.

In both approaches, parametric state duration models for each phoneme need to be calculated first [150]. Several distributions has been tested for this task (e.g. Gaussian), but Burshtein showed that Gamma distributions are best at modeling naturally occurring phoneme duration distributions [151]:

$$d(\tau) = K \exp\{-\alpha\tau\}\tau^{p-1} \tag{7.1}$$

where $\tau = 0, 1, 2, ...$ are the possible state durations in frames and K is a normalizing factor. The parameters α and p are estimated according to

$$\hat{\alpha} = \frac{E\{\tau\}}{VAR\{\tau\}}, \hat{p} = \frac{E^2\{\tau\}}{VAR\{\tau\}} \tag{7.2}$$

where E is the distribution mean and VAR is the distribution variance. E and VAR are estimated empirically using a small portion of the singing data that has been annotated with phoneme occurrences.

In internal duration modeling, the durations are incorporated directly into the Viterbi alignment. This means that the Viterbi output will already be a state sequence that is optimal with regards to the a-priori phoneme duration knowledge. It is,

however, computationally expensive. In previous experiments [152], this approach did not produce better results than the much easier to implement post-processor duration modeling. Therefore, this section will focus on that approach.

When using post-processor duration modeling, knowledge about plausible phoneme durations is imposed on the result of the Viterbi alignment, the obtained state sequence. This is computationally cheap, but only results in a new likelihood score for the obtained sequence and does not provide better possible state sequences. As described in [149], the state sequence obtained from the Viterbi alignment can afterwards be re-scored according to:

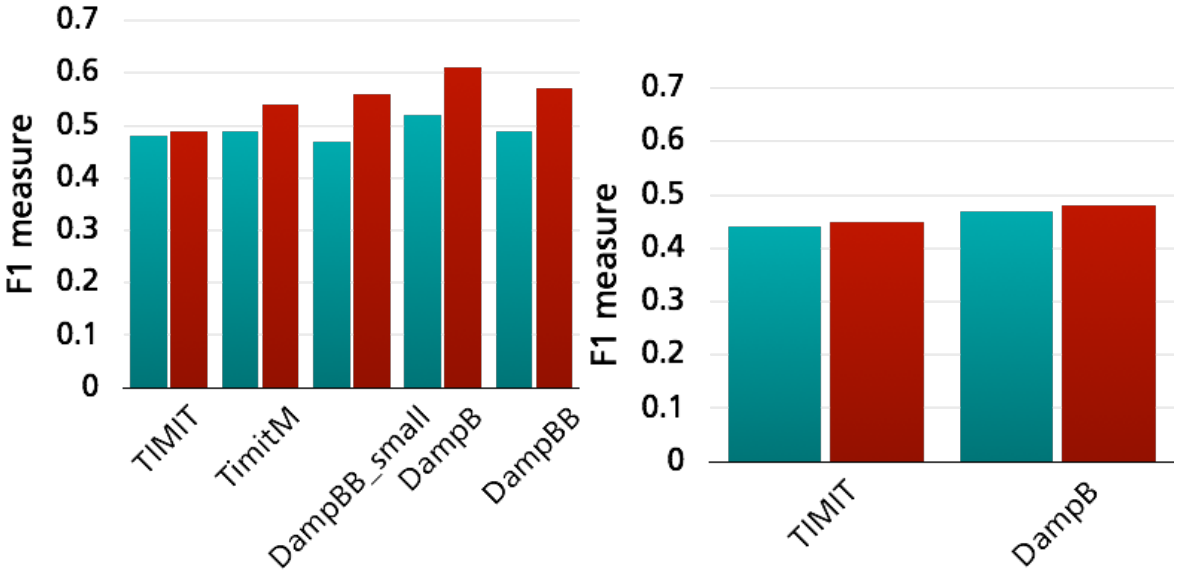
$$\log \hat{f} = \log f + \gamma \sum_{k=1}^{N} d_k(\tau_k) \tag{7.3}$$

where f is the original likelihood of the sequence, γ is a weighting factor, $k = 1...N$ are the discrete states in the state sequence, τ_k are their durations, and $d_k(\tau_k)$ is, again, the probability of state k being active for the duration τ_k.
Using keyword-filler HMMs, only one state sequence per utterance is obtained, which either contains the keyword or not. It is therefore not possible to compare these likelihood scores and Eq. 7.3 cannot be applied directly. To still be able to integrate post-processor duration modeling, the HMM parameters are tuned to obtain a high recall value. Then, the duration likelihood (second half of Eq. 7.3) is calculated for all found occurrences of the keyword and normalized by the number of states taken into account:

$$dl = \frac{1}{N} \sum_{k=1}^{N} d_k(\tau_) \tag{7.4}$$

Then all occurrences where dl is below a certain threshold are discarded.

For the presented results, duration statistics from the *ACAP* data set were used, and only the consonants' durations were taken into account (since vowel durations vary much more as shown in figure 3.1). However, additional experiments showed that the result only varies slightly when using speech statistics instead, and when also discarding unlikely vowel durations. This probably happens because the keywords do not contain many states anyway, and because the duration distribution for vowels has a large variance, allowing for many different durations.



(a) F_1 measures for keyword spotting results on the *ACAP* data set with post-processor duration modeling.

(b) F_1 measures for keyword spotting results on the *DampTest* data sets with post-processor duration modeling.

Figure 7.4: F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models with post-processor duration modeling.

7.2.2 Results

Results with and without post-processor duration modeling for the *ACAP* data set are shown in figure 7.4a. All models were tested. As these results show, F_1 measures improve for all configurations when post-processor duration modeling is employed. The effect is somewhat stronger for the *DAMP* models than for the *TIMIT* model. This confirms that those models are, in principle, able to detect the keyword phonemes, and that exploiting information about plausible phoneme durations can improve the precision. The best result rises from 0.52 to 0.61 with the *DampB* model. Analysis of the detailed results shows that the precision can be improved significantly when detected occurrences with implausible phoneme durations are discarded. However, this often also decreases the recall, resulting in the shown F_1 results.

The approach was also tested on the *DampTest* data sets for two acoustic models. As seen above, F_1 measures on these data sets are generally somewhat blurry for the described reasons. In this case, the results are just a little bit higher with post-processor duration modeling.

7.3 Conclusion

In this chapter, an approach for keyword spotting using the new acoustic models trained on singing was described. This was done by extracting phoneme posteriorgrams generated with these new models from the audio and then running them through a keyword-filler approach to detect 15 keywords. On the *DampTest* data sets, the resulting F_1 measure rises from 0.44 for the models trained on speech (*TIMIT*) to 0.47 for the new models. In general, results on the *DampTest* data sets are somewhat inconclusive because the effect of the different models is shadowed by issues with the test data itself - i.e. automatic and thus possibly inaccurate annotations, amateur singing, and a relatively low frequency of keywords because of the large size of the data sets. On the smaller, hand-annotated *ACAP* test set, the results become a little clearer: The best F_1 measure for the models trained on *TIMIT* is 0.48, and with those trained on *DampB*, it is 0.52.
This result is especially interesting because most of the keywords have few phonemes. Gender-dependent models perform very slightly better than mixed-gender models of the same size. Individual analysis of the keyword results showed that keywords that occur more frequently are detected more accurately. This probably happens because the approach is able to obtain high recall easily, but precision is an issue. The more frequent a keyword, the easier obtaining higher precisions becomes. Additionally, keywords with more phonemes are detected more accurately than short ones because there is more information to base detection on.

The described approach is easy to tune for high recall, but precisions are often not very high. One idea to improve this was to use additional information to discard implausible detections, thus improving precision. This idea was tested in a second approach by integrating knowledge about phoneme durations. Means and standard deviations of phoneme durations were calculated from annotated data (in particular, the *ACAP* singing data set), and then occurrences with phoneme durations outside of their Gamma distributions were ignored. This approach improved F_1 measures by up to 9 percentage points on the *ACAP* test data, with the best result being 0.61. On the *DampTest* data sets, the effect is still existent, but not very pronounced. This is probably because of the blurriness of the result described above.

This approach was only tested with MFCC features. As preliminary experiments suggest [85], other features like TRAP or PLP may work better on singing. So-called

log-mel filterbank features have also been used successfully with DNNs [153]. Another interesting factor is the size and configuration of the classifiers, of which only one was tested (after a small grid search to validate this choice). Other experiments also suggest that incorporating certain phoneme duration information into the recognition can improve the over-all results [152].
As in the phoneme recognition experiments, there is not much of a difference between the acoustic model trained on all 6000 songs of the *DAMP* data set and the one trained only on 4% of this data. It would be interesting to find the exact point at which additional training data does not further improve the models. On the evaluation side, a keyword spotting approach that allows for pronunciation variants or sub-words may produce better results. Language modeling might also help to alleviate some of the errors made during phoneme recognition.
These models have not yet been applied to singing with background music, which would be interesting for practical applications. Since this would probably decrease the result when used on big, unlimited data sets, more specified systems would be more manageable, e.g. for specific music styles, sets of songs, keywords, or specialized applications. Searching for whole phrases instead of short keywords could also make the results better usable in practice.
As shown in [82] and [84] and in the next chapter, alignment of textual lyrics and singing already works well. A combined approach that also employs textual information could be very practical.

8 Lyrics Retrieval and Alignment

Lyrics retrieval and alignment are related tasks: The retrieval task can be interpreted as an alignment of all possible lyrics sequences to the query audio, and a subsequent selection of the best-matching result. This is done in all described algorithms.
As a starting point, a traditional HMM-based algorithm for Forced Alignment was tested. Building on top of the posteriorgrams extracted with the new acoustic models as described in chapter 5, a new algorithm using Dynamic Time Warping (DTW) was developed and evaluated for both alignment and retrieval. Next, another processing step for explicitly extracting phonemes from the posteriorgrams was included; the resulting method was also tested for both alignment and retrieval. All three mentioned algorithms were entered into the 2017 *MIREX* challenge for lyrics-to-audio alignment, in which the algorithms were tested on Hansen's dataset, both in the singing-only and polyphonic conditions (*ACAP* and *ACAP_Poly*, see sections 4.2.2 and 4.3.3), and on Mauch's alignment data set (*Mauch*, see section 4.3.2). The results can be viewed on `http://www.music-ir.org/mirex/wiki/2017:Automatic_Lyrics-to-Audio_Alignment_Results` (last checked: ???).
Finally, a practical application for lyrics alignment is presented: Automatic expletive detection in rap songs.

8.1 HMM-based lyrics-to-audio alignment

For this algorithm, monophone Hidden Markov Models were trained on the *TIMIT* speech data using the HTK framework [136]. Then, Viterbi alignment was run to align the phonemes to the singing audio. These are the same models used to align lyrics to the *DAMP* audio data to generate the training data set described in section 5.3. The results in the *MIREX* challenge are shown in figure 8.1. The detailed results for all tested songs are given in appendix **??**.
On the unaccompanied data set, the mean error is $5.11s$, while the median error is $0.67s$. The large difference comes about because one of the tested songs ("Clocks") produced high error values in every approach. Considering only the other songs, the

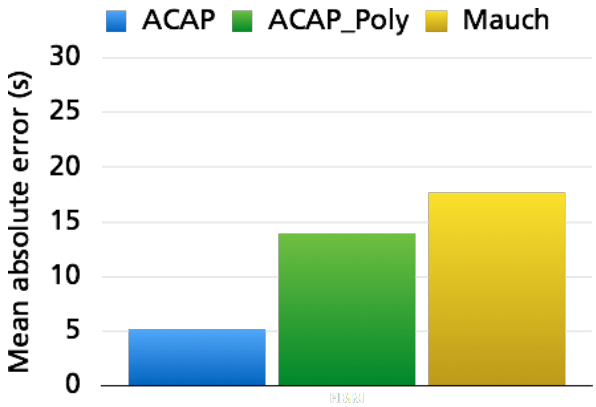


Figure 8.1: *MIREX* results on all three data sets using the HMM-based approach.

highest error is 1.45s, and all others are lower than 1s. This is a good result, especially considering that the approach is relatively simple and trained on speech data only. A manual check suggests that the results are usable in practice.
On the accompanied version of the same data set, the average error is 13.96s and the median error is 7.64. On the *Mauch* data set, the mean error is at 17.7s and the median error is at 10.14s. Once again, the error varies widely across the songs. As expected, the result is worse since no measures were taken make the approach robust to background music. Interestingly, other submitted approaches that employ sources separation did not fare much better on this data set either, suggesting that the core algorithm may be better. It would be interesting to see results for the HMM-based algorithm on polyphonic music with a source separation step (or at least a vocal activity detection step) included.

8.2 Posteriorgram-based retrieval and alignment

A new approach is based on the posteriorgrams generated with the DNN acoustic models described in section 5.3; an overview is illustrated in figure 8.2. In order to align lyrics to these posteriorgrams, binary templates are generated from the text lyrics on the phoneme scale. These can be seen as oracle posteriorgrams, but do not include any timing information.
Between this template and the query posteriogram, a similarity matrix is calculated using the cosine distance. On the resulting matrix, Dynamic Time Warping (DTW) is then performed using the implementation from [24] to obtain the alignment. An

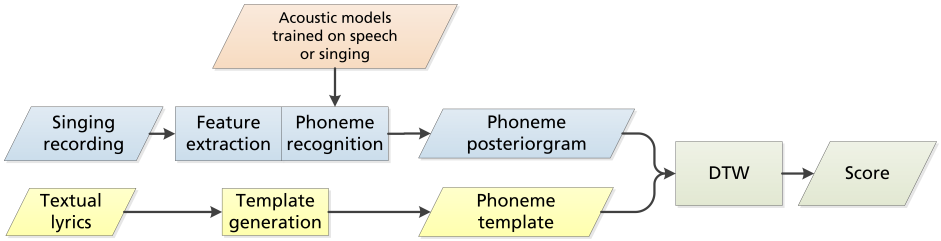


Figure 8.2: Overview of the DTW-based lyrics alignment and retrieval method.

example is shown in figure 8.3.
Two optimizations were made to the algorithm. The first one was a sub-sampling of the phoneme posteriorgrams by the factor 10 (specifically, the mean for 10 consecutive frames was calculated). This increased the speed of the DTW for the individual comparisons and also produced better results. Longer windows were also tested, but this had a negative impact on the result.
Secondly, squaring the posteriorgrams before the similarity calculation produced slightly better results. This makes the posteriorgrams more similar to the binary lyrics templates used for comparison. Binarizing them was also tested, but this emphasized phoneme recognition errors too much.

In the retrieval case, the binary templates are generated for all possible lyrics in the database, and the DTW calculation is performed for each of them with the query posteriorgram. The DTW cost is used as the measure to obtain the best-matching lyrics. Since the phoneme durations in the actual recording and the lyrics templates have different lengths, the length of the warping path should not be a detrimental factor in cost calculation. Therefore, the accumulative cost of the best path is divided by the path length and then retained as a score for each possible lyrics document. In the end, the lyrics document with the lowest cost is chosen as a match (or, in some experiments, the N documents with the lowest costs).
As an additional experiment, both the textual lyrics corpus and the sung inputs were split into smaller segments roughly corresponding to one line in the lyrics each (around 12,000 lines). The retrieval process was then repeated for these inputs. This allowed an evaluation as to how well lyrics could be retrieved from just one single sung line of the song. (Sub-sequence DTW could also be used for this task instead of splitting

both corpora.)

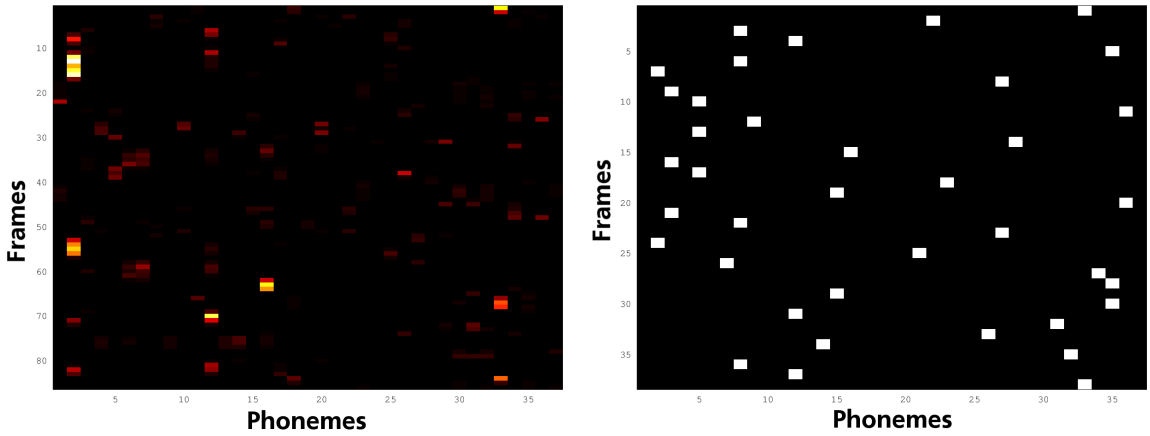
8.2.1 Alignment experiments

The results for this approach in the *MIREX* challenge are displayed in figure 8.4, with the detailed results given in appendix **??**. Overall, errors are much higher than with the HMM-based approach: The mean error is 9.77s for the *ACAP* data set, 27.94s on the *ACAP_Poly* data set, and 22.23s on the *Mauch* data set. This presumably happens because in contrast to the HMM approach, the algorithm does not have any information about phoneme priors and transition probabilities, neither implicitly nor explicitly. DTW on the posteriorgram is a relatively rudimentary method for performing alignments. Nevertheless, a manual check of the results suggests that the algorithm is still able to produce usable results in many cases, with many song-wise results for the *ACAP* data set still below 1s. In particular, it works acceptably when the posteriorgram is not too noisy. This is also corroborated by the retrieval results for unaccompanied queries presented in the following.
As a future step, incorporating prior phoneme information (e.g. in the form of DNN-HMMs) could serve to mitigate some of the noise in the posteriorgram caused by model inaccuracies or by background music.

8.2.2 Retrieval experiments on whole song inputs

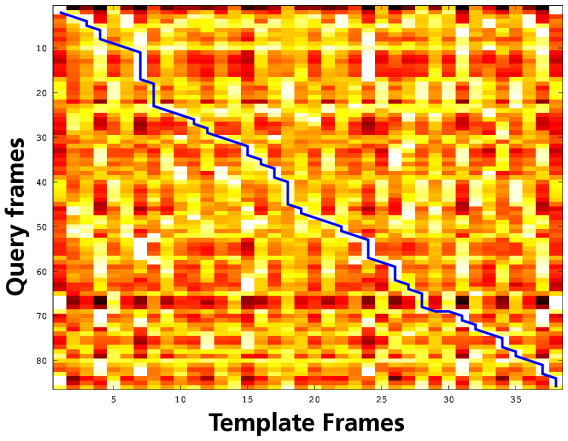
In the first experiment, similarity measures were calculated between the lyrics and recordings of whole songs using the described process. This was tested with phoneme posteriorgrams obtained with all five acoustic models on the female and the male test sets (*DampTestF* and *DampTestM*). The accuracy was then calculated on the 1-, 3-, and 10-best results for each song (i.e., how many lyrics are correctly detected when taking into account the 1, 3, and 10 lowest distances?). The results on the female test set are shown in figure 8.5a, the ones for the male test set in figure 8.5b.

These results show that phoneme posteriorgrams obtained with models trained on speech data (*TIMIT*) generally produce the lowest results in lyrics retrieval. The difference between the two test sets is especially interesting here: On the male test set, the accuracy for the single best result is 58%, while on the female set it is only 39%. Previous experiments showed that the phoneme recognition itself performs somewhat worse for female singing inputs. This effect is compounded in these lyrics retrieval results. This may happen because the frequency range of female singing is even fur-



(a) Phoneme posteriorgram

(b) Phoneme template



(c) Similarity matrix with cheapest path (blue)

Figure 8.3: Example of a similarity calculation: Phoneme posteriorgrams are calculated for the audio recordings **(a)**. Phoneme templates are generated for the textual lyrics **(b)**. Then, a similarity matrix is calculated using the cosine distance between the two, and DTW is performed on it **(c)**. The accumulated cost divided by the path length is the similarity measure.

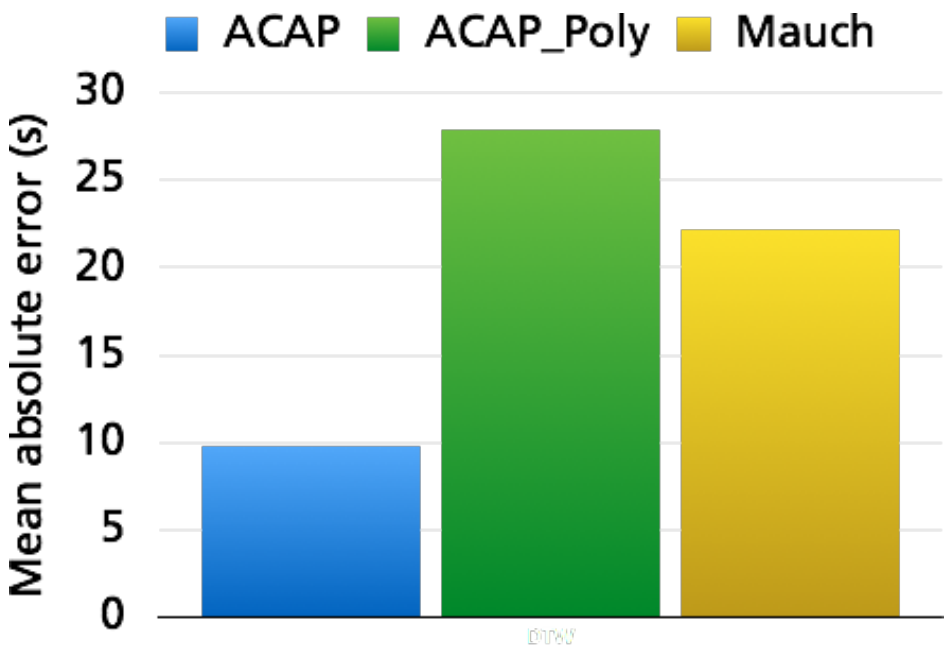
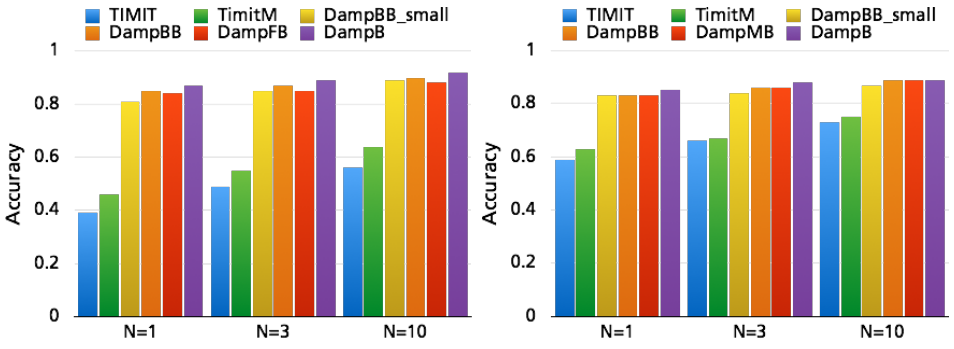


Figure 8.4: *MIREX* results on all three data sets using the DTW-based approach.



(a) DampTestF (b) DampTestM

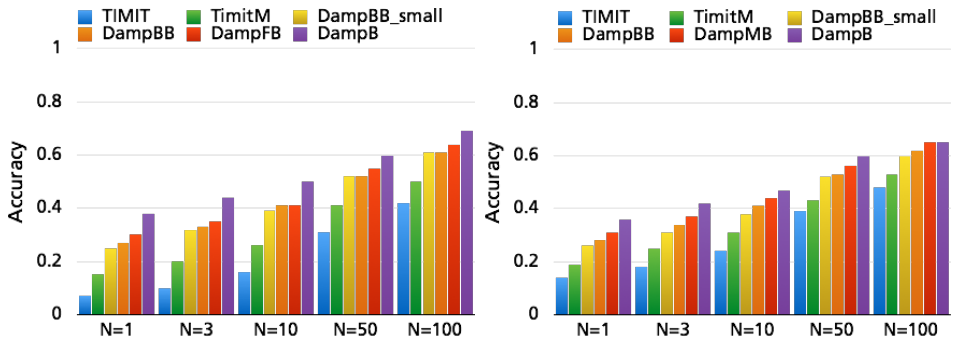
Figure 8.5: Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the 1-, 3-, and 10-best results.

ther removed from that of speech than the frequency range of male singing is [154]. Even female speech is often performed at the lower end of the female singing frequency range. The frequency range of male singing is better covered when training models on speech recordings (especially when speech recordings of both genders are used).
This effect is still visible for the *TimitM* models, which is the variant of *Timit* that was artificially made more "song-like". However, the pitch range was not expanded too far in order to keep the sound natural.
The results improve massively when acoustic models trained on any of the *DAMP* singing corpora are used. The difference between the male and female results disappears, which supports the idea that the female pitch range was not covered well by the models trained on speech. Using the models trained on the smallest singing data set (*DampBB_small*), which is slightly smaller than *Timit*, the results increase to 81% and 83% for the single best result on the female and the male test set respectively. With the models trained on the *DampBB* corpus, which is about twice as big, they increase slightly more to 85% on the female test set. Gender-specific models of the same size do not improve the result in this case.
Finally, the results obtained with the acoustic models trained on the largest singing corpus (*DampB*) provide the very best results at accuracies of 87% and 85%.
For some applications, working with the best N instead of just the very best result could be useful (e.g. for presenting a selection of possible lyrics to a user). When the best 3 results can be taken into account, the accuracies on the best posteriorgrams rise to 89% and 88% on the female and male test sets respectively. When the best 10 results are used, they reach 92% and 89%.

8.2.3 Lyrics retrieval experiments on line-wise inputs

In the second retrieval experiment, the same process was performed on single lines of sung lyrics as inputs (usually a few seconds in duration). Costs were then calculated between the posteriorgrams of these recordings and all 12,000 available lines of lyrics. Lines with fewer than 10 phonemes were not taken into account.
Then, evaluation was performed as to whether a line from the correct song was retrieved in the N-best results. In this way, confusions between repetitions of a line in the same song did not have an impact on the result. However, repetitions of lyrical lines across multiple songs are a possible source of confusion. The results for the female test set are shown in figure 8.6a, the ones for the male test set in figure 8.6b.

Again, a difference between both test sets is visible when generating posteriograms with the *TIMIT* models. The accuracy on the best result is 14% for the male test set,



(a) DampTestF (b) DampTestM

Figure 8.6: Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the 1-, 3-, 10-, 50-, and 100-best results.

but just 7% for the female test.
The results for the *DAMP* models show the same basic tendencies as before, although naturally much lower. For the single best result, the accuracies when using the *DampB* model are 38% and 36% on the female and male test sets respectively. For this task, gender-dependent models produce slightly higher results than the mixed-gender ones of the same size.

8.3 Phoneme-based retrieval and alignment

Next, another step was introduced to improve the previous approach and make it more flexible. This step serves to compress the posteriorgrams down to a plausible sequence of phonemes, which can then be used to search directly on a textual lyrics database. Text comparison is much cheaper than the previous comparison strategy, and enables quick expansion of the lyrics database.
In parallel, the lyrics database is prepared by converting it into phoneme sequences as described in section **??**. The key algorithms are (a) how to generate plausible phoneme sequences from the posteriorgrams, and (b) how to compare these against the sequences in the database. These parts will be described in more detail in the following. An overview is given in figure 8.7.

Symbolic mapping As described before, starting from the sung recording used as a query, MFCC features are extracted and run through the acoustic model trained on singing to recognize the contained phonemes. This produces a phoneme posteriorgram,

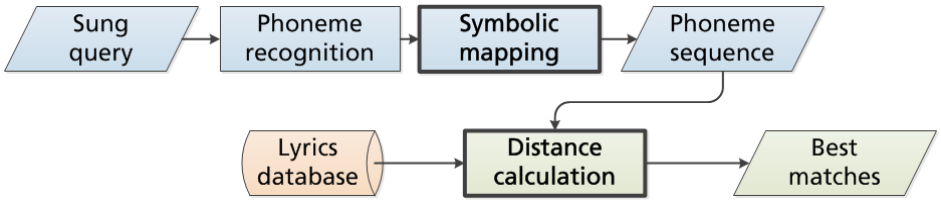


Figure 8.7: Overview of the phoneme-based lyrics retrieval process.

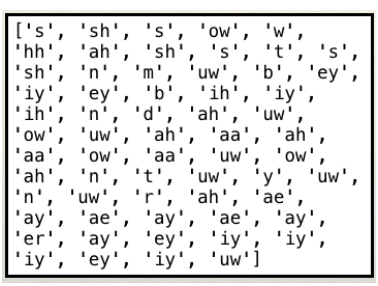
such as the one shown in figure 2.24; i.e., probabilities of each phone over each time frame. These probabilities contain some noise, both due to inaccuracies in the model and due to ambiguities or actual noise in the performance.
The following steps are undertaken to obtain a plausible phoneme sequence from the posteriorgram:

Smoothing First, the posteriorgram is smoothed along the time axis with a window of length 3 in order to remove small blips in the phoneme probabilities.

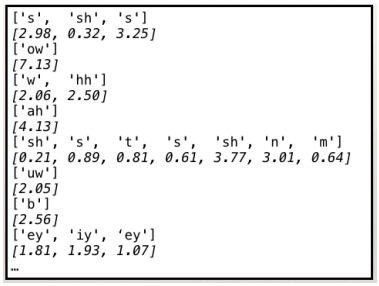
Maximum selection and grouping Then, the maximum bin (i.e. phoneme) per frame is selected, and consecutive results are grouped. An example is given in figure 8.8a.

Filtering by probability and duration These results can then be pre-filtered to discard those that are too short or to improbable. This is done with different parameterizations for vowels and consonants since vowels are usually longer in duration. This yields a first sequence of phonemes, each with duration and sum probability information, which is usually too long and noisy. In particular, a lot of fluctuations between similar phonemes occur.

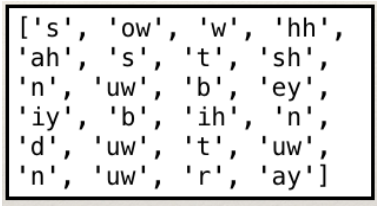
Grouping by blocks and filtering through confusion matrix This problem is solved by first grouping the detected phonemes into blocks, in this case vowel and consonant blocks (shown in figure **??**). Then, a decision needs to be made as to which elements of these blocks are the "true" phonemes and which ones are noise. This is done by taking each phoneme's probability as well as the confusion between phonemes into account. The confusion is calculated in advance by running the classifier on an annotated test set; the result covers both the confusion by inaccuracies in the classifier as well as perceptual or performance-based confusions (e.g. transforming a long *[ay]* sound into *[aa - ay - iy]* during singing). An example is shown in figure 8.9. The product of the probabilities and the confusions are calculated for the highest combinations up to a certain threshold, and all



(a) Original phoneme sequence



(b) Grouped into blocks (with probabilities in italics)



(c) Filtered result

Figure 8.8: Example of the block grouping of the phoneme sequence and subsequent filtering by probabilities and confusions.

other detected phonemes are discarded. This results in a shorter, more plausible phoneme sequence (figure 8.8c).

Distance calculation Then, the distances between the extracted phoneme sequence and the ones provided in the lyrics database are calculated.

First, an optional step to speed up the process is introduced. Each sequence's number of vowels is counted in advance, and the same is done for the query sequence. Then, only sequences with roughly the same amount of vowels are compared (with some tolerance). This slightly decreases accuracies, but drastically speeds up the calculation time.

The similarity calculation itself is implemented with a modified Levenshtein distance. Again, the classifier's confusion between phonemes is taken into account. These confusions are used as the Levenshtein weights for substitutions. Surprisingly, using them for insertion weights improves the results as well. This probably happens because of the effect described above: A singer will in many cases vocalize an phoneme as multiple different ones, particularly for vowels with long durations. This will result in an insertion in the detected phoneme sequence, which is not necessarily a "wrong" classification by the acoustic model, but does not correspond to the expected sequence for this line of lyrics. For this reason, such insertions should not be harshly penalized. For

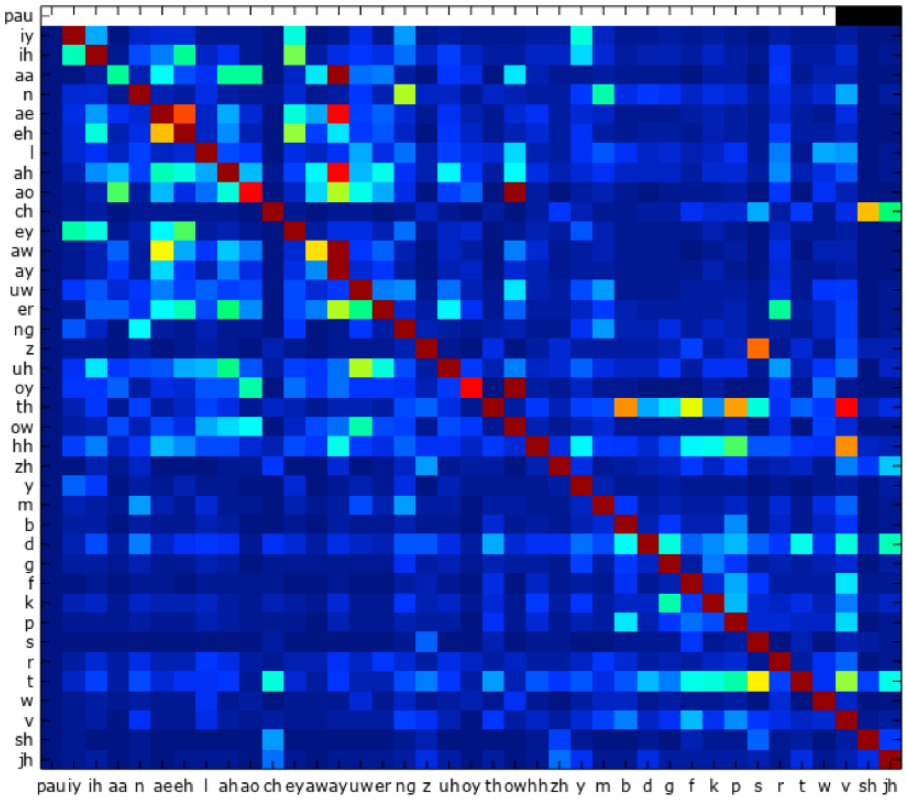
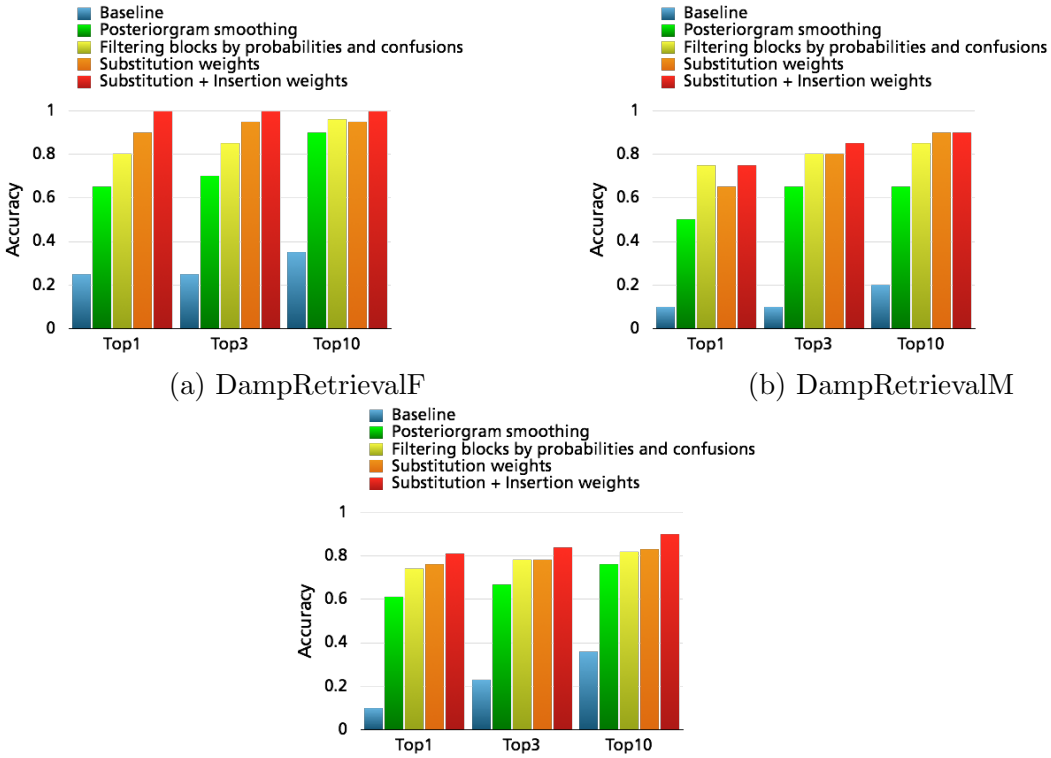


Figure 8.9: Example of a confusion matrix for an acoustic model. (Note that *[pau]* confusions are set to 0).



(a) DampRetrievalF

(b) DampRetrievalM

(c) AuthorRetrieval

Figure 8.10: Results of the phoneme-based retrieval algorithm with various improvement steps for three small calibration data sets.

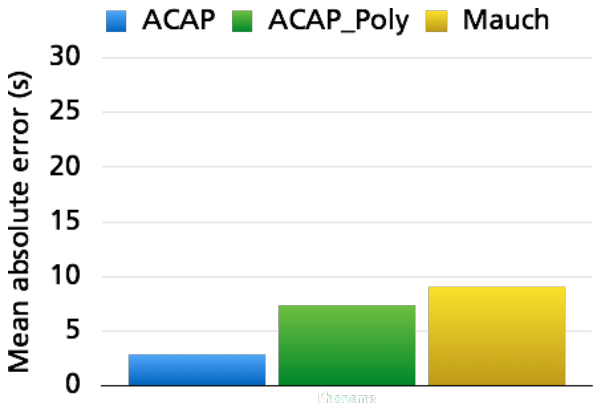


Figure 8.11: *MIREX* results on all three data sets using the phoneme-based approach.

deletions, the weight is set to .5 to balance out the lower insertion and substitution weights.

8.3.1 Alignment experiments

MIREX results for this approach are shown in figure 8.11; the detailed results can be found in appendix **??**. Over-all, this approach produces much lower error values than the other two, and was in fact the winning algorithm in the competition. This confirms that the phoneme detection strategy is a feasible alternative to the HMM-based approach. On the *ACAP* data set, the mean error is at 2.87s and the median is 0.26s. On *ACAP_Poly*, these values are 7.34s and 4.55s respectively, and on *Mauch*, they are 9.03s and 7.52s. Once again, the results are much higher on polyphonic data than on unaccompanied singing, for which the models were trained. However, the error is still lower than with the submitted methods that include source separation. Therefore, future experiments that also employ pre-processing of this kind would be very interesting. Looking at the song-wise results, songs with heavier and noisier accompaniment generally receive higher errors than others.

8.3.2 Retrieval experiments: Calibration

For calibrating the algorithm's parameters quickly, two smaller data sets were used: *DampRetrieval*, which consists of 20 female and 20 male hand-selected sung segments with clear pronunciation and good audio quality, and *AuthorRetrieval*, which consists of 90 small performances of phrases in the *DAMP* data set by the author. Both are described in sections **??** and **??**. This was done to ensure that the results were not

influenced by flaws in the data set (such as unclear or erroneous pronunciation or bad quality). Lyrics can still come from the whole *DAMP* lyrics set, resulting in 12,000 lines of lyrics from 300 songs.
Figure 8.10 shows an overview over the experimental results. Retrieval rates are reported for the Top 1 result (i.e. the one with the lowest Levenshtein distance), the Top 3, and the Top 10 results. Queries were allowed to be one to three consecutive lines of songs, increasing the number of "virtual" database entries to around 36,000 (12,000 lines as 1-, 2-, and 3-grams). It should be noted that a result counts as correct when the correct song (out of 300) was detected; this was done as a simplification because the same line of lyrics is often repeated in songs. For possible applications, users are most probably interested in obtaining the correct full song lyrics, rather than a specific line. Picking random results would therefore result in a retrieval rate of .003.
The various improvement steps of the algorithm were tested as follows:

Baseline This is the most straightforward approach: Directly pick the phonemes with the highest probabilities for each frame from the posteriorgram, group them by consecutive phonemes, and use the result of that for searching the lyrics database with a standard Levenshtein implementation. This results in retrieval rates of .25, .1, and .23 for the Female, Male, and Author test sets respectively.

Posteriorgram smoothing This is the same as the baseline approach, but the posteriorgram is smoothed along the time axis as described in paragraph 8.3. This already improves the result by around .4 for each test set.

Filtering blocks by probabilities and confusions This includes the last step described in paragraph 8.3. The result is improved further by .13 to .25.

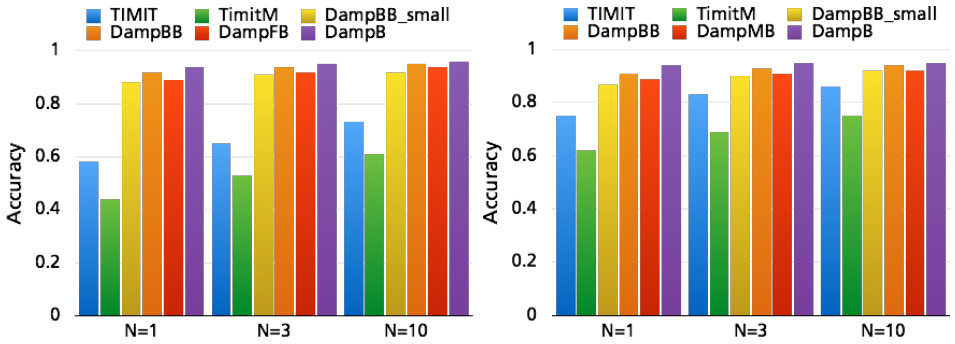
Substitution and insertion weights Finally, the modified Levenshtein distance as described in paragraph 8.3 is calculated. When using the confusion weights for phoneme substitutions only, the result increases further, and even more so when they are also used for the insertion weights. The final Top 1 retrieval rates are 1, .75, and .81 for the three test sets respectively.

8.3.3 Retrieval experiments: Full data set

The full algorithm was then also run on the full *DampTest* data sets in the same way as the DTW-based approach.
The results for retrieval using whole songs as queries are shown in figure 8.12. All

the steps presented in the previous section were included in the calculation of these results. As in the DTW-based approach, the material on which the acoustic model is trained plays a huge role. When using models trained on *TIMIT*, the retrieval rate of the Top-1 result is just 0.58 on the female test set, and 0.75 on the male one (this mirrors the discrepancy between female and male results already seen in the previous approach). Interestingly, the model trained on *TimitM* actually performs worse for this approach with retrieval rates of 0.44 and 0.62 for the female and male test sets respectively. Since this approach is based on extracting salient phonemes from the posteriorgrams, it is possible that the time-stretched and pitch-shifted training data causes too much noise in the model's results, or a stronger bias towards vowels, which did not throw off the DTW approach.
As previously seen, the models trained on the *DAMP* data sets perform much better at this task. On both the female and male test sets, the best result is a 0.94 retrieval rate with the model trained on *DampB*. The models trained on less data gradually perform a few percentage points worse. Interestingly, training on gender-specific data once again does not improve the result. As suggested before, this could be due to the higher variety in timbre and pitch across songs and singers compared to variation between genders.
The retrieval rates for the Top-3 and Top-10 results follow a similar pattern. For the *DAMP*-based models, the increase is not very high because the Top-1 result is already close to an upper bound. Analysis of the non-retrieved songs mainly shows discrepancies between the expected lyrics and the actual performance, such as those described in section 5.3.4. This means that the most effective way to improve results would lie in making the algorithm more robust to such variances (versus improving the detection itself). Over-all, the results are significantly higher than with the DTW approach.

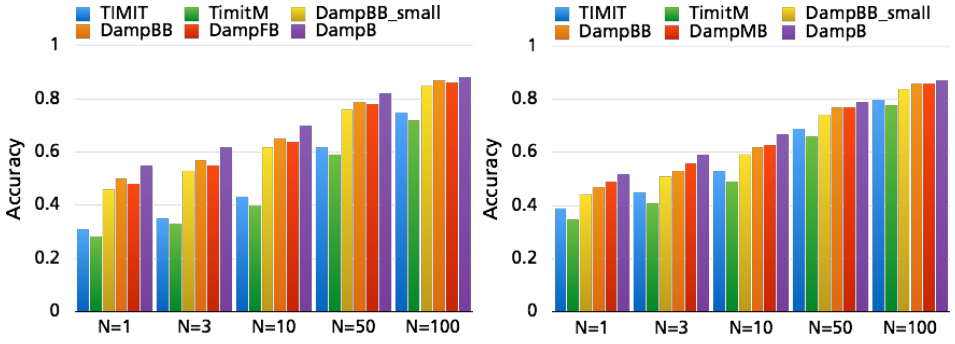
Figure 8.13 shows the analogous results when using single line queries for retrieval. The trend across the different classifiers runs parallel to the whole song results, with the *TimitM* models performing worse than the *TIMIT* models, and both being outperformed by the *DAMP*-based models. The best Top-1 result is 0.55 for the female test set, and 0.52 for the male one. For the Top-10 results, the retrieval results are 0.7 and 0.67 respectively. Once again, lines with fewer than 10 phonemes were excluded; however, lines may still occur in more than one song. Considering this fact and the previously mentioned interfering factors, this result is already salient. When comparing these results to the ones for the calibration data set, it becomes clear that



(a) DampTestF (b) DampTestM

Figure 8.12: Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the 1-, 3-, and 10-best results.

.



(a) DampTestF (b) DampTestM

Figure 8.13: Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the 1-, 3-, 10-, 50-, and 100-best results.

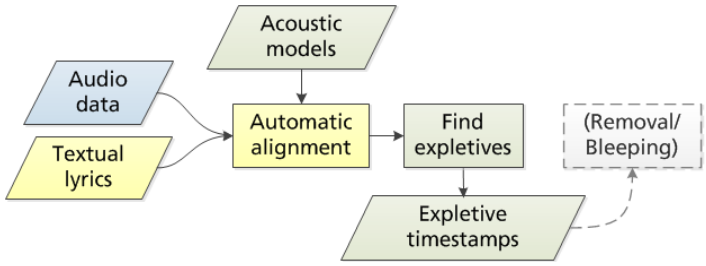


Figure 8.14: Data flow in the expletive detection approach.

performance is highly dependent on the quality of the queries. In a practically usable system, users could be asked to enunciate clearly or to perform long segments.

8.4 Application: Expletive detection

Lots of song lyrics contain expletives. There are many scenarios in which it is necessary to know when these words occur, e.g. for airplay and for the protection of minors. In the case of airplay, they are commonly "bleeped" or acoustically removed. The alignment strategies described previously are employed for the practical scenario of finding such expletives automatically.
The test data set is the one compiled by Queen Mary University, described in section 4.3.1.

A direct keyword spotting approach was also considered, but this did not generate sufficient results since most of the expletives only consist of 2 or 3 phonemes. Keyword spotting becomes notoriously hard for such short keywords as described in chapter 7. Since textual lyrics are usually easily available on the internet, a new approach utilizing those was developed:

1. Automatically align textual lyrics to audio (as described above)
2. Search for pre-defined expletives in the result
3. If necessary, remove those expletives. A stereo subtraction approach was used, which works adequately for this case since the removed timespans are short. Alternatively, keywords can be masked with a bleep or similar.

The data flow is shown in figure 8.14.

The alignment was performed using the HMM-based alignment described in section 8.1, and the DTW alignment from section 8.2 using the DNN acoustic models trained

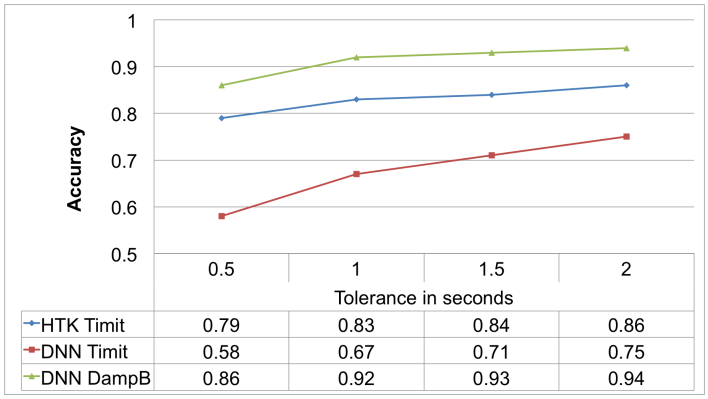


Figure 8.15: Results for the expletive detection approach at various tolerances.

on *TIMIT* and on *DampB*. Accuracy was then calculated by evaluating how many of the annotated expletives were recognized at their correct timestamps (with various tolerances). The results are shown in figure 8.15.

In this small practical example, only two alignment strategies were tested, but there are others that could provide better results. Whenever the alignment failed, it was mostly due to solo instruments. In order to remedy this, vocal detection (and possibly source separation) could be employed prior to alignment. Additionally, a more sophisticated removal approach could be implemented to remove the expletives.
Lyrics collected from the internet are often incorrect, e.g. because repetitions are not spelled out, because of spelling errors, or because they refer to different versions of a song. This approach could be expanded to allow for some flexibility in this respect. At the moment, these lyrics need to be provided manually. In the future, those could be retrieved automatically (e.g. using the approaches described above).

8.5 Conclusion

In this section, three approaches to lyrics-to-audio alignment and retrieval and one practical application were presented. The first one uses HMMs trained on speech for alignment. The mean alignment error in the 2017 *MIREX* challenge was 5.11s on unaccompanied singing, and 13.96s and 17.7s on two polyphonic data sets. A manual check on some examples suggests that the algorithm is already practically usable in many cases, even for polyphonic music. It was also used to generate the annotations

for the *DAMP* training data sets as described in section 5.3.

The second algorithm is based on Dynamic Time Warping. In its first step, phoneme posteriorgrams are generated with the various acoustic models described in chapter 5, in particular those trained on *TIMIT* speech data, the time-stretched and pitch-shifter version *TimitM*, and several of the *DAMP*-based data sets whose annotations were generated with the previous approach. Then, a one-hot binary template is generated from the lyrics to be aligned, and an optimal alignment between this and the posteriorgram is computed via DTW. In the *MIREX* challenge, this approach achieved a mean alignment error of 9.77s on the unaccompanied singing data, and of 27.94s and 22.23s on the two polyphonic data sets.
The same approach can also be used for retrieving lyrics from a text database with a sung query. To this end, binary templates are generated for all possible lyrics, and the alignment is performed for all of them. Then, the result with the lowest DTW cost is selected as the winner. When the whole song is used as the input, an accuracy of 86% for the single best result is obtained. If the 10 best results are taken into account, this rises to 91%. When using only short sung lines as input, the mean Top-1 accuracy for retrieving the correct song lyrics of the whole song is 37%. For the best 100 results, the accuracy is 67%. An interesting result was the difference between the female and the male test sets: On the female test set, retrieval with models trained on speech was significantly lower than on the male set (39% vs. 58% on the song-wise task). This may happen because the frequency range of female singing is not covered well with speech data only. When using acoustic models trained on singing, this difference disappears and the results become significantly higher in general. Even for a model trained on less data than that contained in *TIMIT*, the average accuracy is 82%.

The third approach is also based on posteriorgrams generated with the models from chapter 5. However, instead of operating directly on them, phoneme sequences are extracted. This is done by selecting the phoneme with the highest probability in each frame and compressing this sequence down to discard unlikely occurrences. This process takes the known phoneme confusions of the classifier into account. Several improvement steps were found. The extracted phoneme sequence is then aligned to the expected one (from the known lyrics) using the Levenshtein distance. In the *MIREX* challenge, this algorithm produced the best results with a mean error of 2.87s on unaccompanied singing, and 7.34s and 4.55s on polyphonic music.
For lyrics retrieval, the extracted phoneme sequence is compared to all the sequences

in the lyrics database with the Levenshtein distance, and the result with the lowest distance is selected. On the same test data as above, the approach achieves a retrieval rate of 0.94 for whole-song inputs. With line-wise inputs, the retrieval rate is 0.54 for the Top-1 result, and 0.88 for the Top-100. Once again, there is a significant difference between the female and male results with models trained on speech, which disappears with singing-based models.
Tests on a smaller database of hand-selected clean audio samples resulted in much higher retrieval rates for single-line queries. This suggests that the system can perform much better when provided with clean inputs; this would be feasible in a real-world application.

Finally, the HMM and DTW alignments were tested in a real-world scenario: The extraction of expletives from popular music. To this end, known lyrics were aligned to polyphonic song recordings containing such words. Then, the found time segments were masked with other sounds, or stereo subtraction was performed to remove the singing voice. At a tolerance of $1s$, 92% of the expletives were detected in their correct positions, making the system usable for practical applications.

In all experiments, analysis of the errors showed the same possible sources as described in section 5.3.4. Many of them had to do with enunciation issues (clarity, accents, or children's voices) or issues with the recording itself (background music, clipping, extraneous speaking). These problems would not be as prevalent in professional recordings. However, some of them could be fixed with adaptations to the algorithm. In polyphonic alignment experiments, errors also occurred due to prominent instruments, in particular during instrumental solos. As described in section 3.1, this is also an issue in other MIR tasks related to singing, such as Vocal Activity Detection.

As described, the developed algorithms are in many cases ready for practical applications. In the future, it would be interesting to see them integrated into existing systems. There are also ways to make them more robust, e.g. to errors in the lyrics transcriptions or mistakes or variations by the singers. No special steps have been taken so far to adapt them to polyphonic music; Vocal Activity Detection or Source Separation could be performed in the pre-processing.
So far, the retrieval approaches were only tested on a relatively small lyrics database containing 12,000 lines of lyrics across 300 songs. For larger databases, scalability might become an issue. One partial solution was already integrated into the phoneme-based

method. To take this one step further, both retrieval algorithms could first perform a rough search with smaller lyrical "hashes" to find possible matches, and then perform a refinement as presented. This is similar to techniques that are already used in audio fingerprinting [155].
Alternatively, the phoneme-based approach could be expanded to retrieve lyrics from the internet instead of from a fixed database, e.g. with Semantic Web techniques.

9 Conclusion

10 Future work

Bibliography

[1] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition," *Speech Communication*, vol. 54, no. 4, pp. 543–565, May 2012.

[2] M. Müller, *Information Retrieval for Music and Motion*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[3] P. Mermelstein, "Distance measures for speech recognition: Psychological and instrumental," in *Pattern Recognition and Artificial Intelligence*, C. H. Chen, Ed., pp. 374–388. Academic Press, New York, 1976.

[4] S. B. Davis and P. Mermelstein, "Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences," *IEEE Transactions on acoustics, speech and signal processing*, pp. 357–366, 1980.

[5] J. S. Bridle and M. D. Brown, "An experimental automatic word recognition system," Tech. Rep., Joint Speech Research Unit, Ruislip, England, 1974.

[6] D. D. O'Shaughnessy, "Invited paper: Automatic speech recognition: History, methods and challenges.," *Pattern Recognition*, vol. 41, no. 10, pp. 2965–2979, 2008.

[7] B. Bielefeld, "Language identification using shifted delta cepstrum," in *Fourteenth annual speech research symposium*, Baltimore, MD, USA, 1994.

[8] P. A. Torres-Carrasquillo, E. Singer, M. A. Kohler, R. J. Greene, D. A. Reynolds, and J. J. R. Deller, "Approaches to language identification using Gaussian mixture models and shifted delta cepstral features," in *International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, USA, 2002, pp. 89–92.

[9] W. M. Campbell, J. P. Campbell, D. A. Reynolds, E. Singer, and P. A. Torres-Carrasquillo, "Support vector machines for speaker and language recognition," *Computer Speech and Language*, vol. 20, pp. 210–229, 2006.

[10] F. Allen, E. Ambikairajah, and J. Epps, "Language identification using Warping and the Shifted Delta Cepstrum," in *2005 IEEE 7th Workshop on Multimedia Signal Processing*, Shanghai, China, 2006, pp. 1–4.

[11] H. Hermansky, "Perceptual linear predictive (PLP) analysis of speech," *J. Acoust. Soc. Am.*, vol. 57, no. 4, pp. 1738–52, Apr. 1990.

[12] F. Hönig, G. Stemmer, C. Hacker, and F. Brugnara, "Revising perceptual linear prediction (plp).," in *INTERSPEECH*. 2005, pp. 2997–3000, ISCA.

[13] P. C. Woodland, M. J. F. Gales, and D. Pye, "Improving environmental robustness in large vocabulary speech recognition," in *ICASSP*. 1996, pp. 65–68, IEEE Computer Society.

[14] J. Makhoul and L. Cosell, "Lpcw: An lpc vocoder with linear predictive spectral warping.," in *ICASSP*. 1976, pp. 466–469, IEEE.

[15] S. S. Stevens, "On the psychophysical law," *Psychological Review*, vol. 64, no. 3, pp. 153–181, 1957.

[16] J. Makhoul, "Spectral linear prediction: Properties and applications," *IEEE Transactions on acoustics, speech and signal processing*, vol. 23, no. 3, pp. 283–296, 1975.

[17] H. Hermansky, N. Morgan, A. Bayya, and P. Kohn, "{RASTA-PLP} Speech Analysis," Tech. Rep. TR-91-069, ICSI, 1991.

[18] H. Hermansky and S. Sharma, "Traps – classifiers of temporal patterns," in *Proceedings of the 5th International Conference on Spoken Language Processing (ICSLP)*, Sydney, Australia, 1998, pp. 1003–1006.

[19] H. Hermansky and S. Sharma, "Temporal patterns (TRAPS) in ASR of noisy speech," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Phoenix, AZ, USA, 1999, pp. 289–292.

[20] P. Matejka, I. Szoeke, P. Schwarz, and J. Cernocky, "Automatic language identification using phoneme and automatically derived unit strings," in *Proceedings of 7th International Conference on Text, Speech, and Dialogue (TSD)*, Brno, Czech Republic, 2004, pp. 147–154.

[21] J. K. Hansen, "Recognition of Phonemes in A-cappella Recordings using Temporal Patterns and Mel Frequency Cepstral Coefficients," in *9th Sound and Music Computing Conference (SMC)*, Copenhagen, Denmark, 2012, pp. 494–499.

[22] J. Li, L. Deng, Y. Gong, and R. Häb-Umbach, *Robust Automatic Speech Recognition: A Bridge to Practical Applications*, Academic Press, 2015.

[23] L. Rabiner and B.-H. Juang, *Fundamentals of Speech Recognition*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[24] D. P. W. Ellis, "Dynamic Time Warp (DTW) in Matlab," 2003, Web resource, Last checked: 03/30/16.

[25] V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions and Reversals," *Soviet Physics – Doklady*, vol. 10, no. 8, 1966.

[26] G. Navarro, "A guided tour to approximate string matching," *ACM Comput. Surv.*, vol. 33, no. 1, pp. 31–88, Mar. 2001.

[27] R. Lee, Ed., *Applied Computing and Information Technology*, Studies in Computational Intelligence. Springer, 2017.

[28] D. Jurafsky, "Minimum Edit Distance," University Lecture, 2014.

[29] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, vol. 39, no. 1, pp. 1–38, 1977.

[30] D. A. Reynolds and R. C. Rose, "Robust text-independent speaker identification using Gaussian mixture speaker models," *IEEE Transactions on Speech and Audio Processing*, vol. 3, no. 1, pp. 72–83, Jan. 1995.

[31] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press, New York, NY, USA, 3 edition, 2007.

[32] H. Permuter, J. Francos, and I. Jermyn, "Gaussian mixture models of texture and colour for image database retrieval.," in *IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Piscataway, NJ, April 2003, vol. 3, pp. 569–572, IEEE.

[33] C. Alexander, "Normal mixture diffusion with uncertain volatility: Modelling short- and long-term smile effects," *Journal of Banking and Finance*, vol. 28, no. 12, pp. 2957–2980, 2004.

[34] J. Chen, O. E. Adebomi, O. S. Olusayo, and W. Kulesza, "The evaluation of the gaussian mixture probability hypothesis density approach for multi-target tracking," in *IEEE International Conference on Imaging Systems and Techniques (IST)*, 2010, pp. 182–185.

[35] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *Ann. Math. Statist.*, vol. 37, no. 6, pp. 1554–1563, 1966.

[36] L. E. Baum and J. A. Eagon, "An inequality with applications to statistical estimation for probabilistic functions of markov processes and to a model for ecology," *Bull. Amer. Math. Soc.*, vol. 73, no. 3, pp. 360–363, 05 1967.

[37] L. E. Baum and G. R. Sell, "Growth transformations for functions on manifolds.," *Pacific J. Math.*, vol. 27, no. 2, pp. 211–227, 1968.

[38] L. E. Baum, T. Petrie, G. Soules, and N. Weiss, "A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains," *Ann. Math. Statist.*, vol. 41, no. 1, pp. 164–171, 02 1970.

[39] L. E. Baum, "An inequality and associated maximization technique in statistical estimation of probabilistic functions of a markov process," *Inequalities*, vol. 3, pp. 1–8, 1972.

[40] L. R. Rabiner and B. H. Juang, "An introduction to hidden markov models," *IEEE ASSp Magazine*, Jan. 1986.

[41] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[42] F. Jelinek, L. R. Bahl, and R. L. Mercer, "Design of a linguistic statistical decoder for the recognition of continuous speech," *IEEE Transactions on Information Theory*, vol. 21, no. 3, pp. 250–256, 1975.

[43] M. Gales and S. Young, "The application of hidden markov models in speech recognition," *Found. Trends Signal Process.*, vol. 1, no. 3, pp. 195–304, Jan. 2007.

[44] C. D. Manning and H. Schütze, *Foundations of Statistical Natural Language Processing*, MIT Press, Cambridge, MA, USA, 1999.

[45] R. Nag, K. Wong, and F. Fallside, "Script recognition using hidden Markov models," in *ICASSP*. 1986, pp. 2071–2074, IEEE.

[46] A. Krogh, M. Brown, I. S. Mian, K. Sjölander, and D. Haussler, "Hidden Markov models in computational biology: Applications to protein modeling," *Journal of Molecular Biology*, vol. 235, pp. 1501–1531, Feb. 1994.

[47] R. Durbin, S. Eddy, A. Krogh, and G. Mitchison, *Biological Sequence Analysis.*, Cambridge University Press, 1998.

[48] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[49] C.-W. Hsu, C.-C. Chang, and C.-J. Lin, "A practical guide to support vector classification," Tech. Rep., Department of Computer Science and Information Engineering, National Taiwan University, Taipei 106, Taiwan, 2010.

[50] D. Boswell, "Introduction to support vector machines," 2002.

[51] C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," *Data mining and knowledge discovery*, vol. 2, no. 2, pp. 121–167, 1998.

[52] N. Dehak, P. J. Kenny, R. Dehak, P. Dumouchel, and P. Ouellet, "Front-End Factor Analysis for Speaker Verification," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 4, pp. 788–798, may 2011.

[53] D. Martinez, O. Plchot, and L. Burget, "Language Recognition in iVectors Space," in *Interspeech*, Florence, Italy, August 2011, pp. 861–864.

[54] D. A. Reynolds, T. F. Quatieri, and R. B. Dunn, "Speaker verification using adapted gaussian mixture models," in *Digital Signal Processing*, 2000, p. 2000.

[55] C. Charbuillet, D. Tardieu, and G. Peeters, "GMM Supervector for content based music similarity," *... Conference on Digital ...*, , no. 1, pp. 1–4, 2011.

[56] W. Mcculloch and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 127–147, 1943.

[57] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[58] A. G. Ivakhnenko and V. G. Lapa, "Cybernetic Predicting Devices," Tech. Rep., CCM Information Corporation, 1965.

[59] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, MIT Press, 1969.

[60] P. J. Werbos, *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*, Ph.D. thesis, Harvard University, 1974.

[61] A. H. Waibel, T. Hanazawa, G. E. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks.," *IEEE Trans. Acoustics, Speech, and Signal Processing*, vol. 37, no. 3, pp. 328–339, 1989.

[62] S. Srinivas, R. K. Sarvadevabhatla, K. R. Mopuri, N. Prabhu, S. S. S. Kruthiventi, and R. V. Babu, "A taxonomy of deep convolutional neural nets for computer vision," *Frontiers in Robotics and AI*, vol. 2016, 2016.

[63] Y. Goldberg, "A primer on neural network models for natural language processing," *J. Artif. Intell. Res.*, vol. 57, pp. 345–420, 2016.

[64] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, no. 7, pp. 1527–1554, July 2006.

[65] J. Markoff, "Scientists see promise in Deep Learning programs," *The New York Times*, Nov. 23, 2012.

[66] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 2015, Published online 2014; based on TR arXiv:1404.7828 [cs.NE].

[67] L. Deng, G. Hinton, and B. Kingsbury, "New types of deep neural network learning for speech recognition and related applications: An overview," in *Proc. Int. Conf. Acoust., Speech, Signal Process*, 2013.

[68] D. J. C. MacKay, *Information Theory, Inference & Learning Algorithms*, Cambridge University Press, New York, NY, USA, 2002.

[69] R. Campbell, "Demystifying Deep Neural Nets," *Medium*, Mar. 31, 2017, `https://medium.com/manchester-futurists/demystifying-deep-neural-nets-efb726eae941`.

[70] S. Hochreiter, "The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 06, no. 02, pp. 107–116, 1998.

[71] S. Hochreiter and J. Schmidhuber, "Long short-term memory.," *Neural computation*, vol. 9, no. 8, pp. 1735–80, 1997.

[72] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation," *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 1724–1734, 2014.

[73] H. Sak, A. W. Senior, and F. Beaufays, "Long short-term memory recurrent neural network architectures for large scale acoustic modeling," in *INTERSPEECH*. 2014, pp. 338–342, ISCA.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, USA, 2012, NIPS'12, pp. 1097–1105, Curran Associates Inc.

[75] K. J. Piczak, "Environmental sound classification with convolutional neural networks," in *Proceedings of the IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*. 2015, IEEE.

[76] O. Abdel-Hamid, A.-R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional neural networks for speech recognition," *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, vol. 22, no. 10, pp. 1533–1545, Oct. 2014.

[77] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016, `http://www.deeplearningbook.org`.

[78] M. J. Hunt, "Figures of merit for assessing connected-word recognisers.," *Speech Communication*, vol. 9, no. 4, pp. 329–336, 1990.

[79] M. A. Zissman, "Comparison of four approaches to automatic language identification of telephone speech," *IEEE Transactions on Speech and Audio Processing*, vol. 4, no. 1, pp. 31–44, Jan. 1996.

[80] E. Singer, P. A. Torres-Carrasquillo, T. P. Gleason, W. M. Campbell, and D. A. Reynolds, "Acoustic, phonetic, and discriminative approaches to automatic language identification," in *Proceedings of Eurospeech*, Geneva, Switzerland, 2003, pp. 1345–1348.

[81] "The 2015 NIST Language Recognition Evaluation Plan (LRE15)," Tech. Rep., NIST, 2015.

[82] A. Moyal, V. Aharonson, E. Tetariy, and M. Gishri, *Phonetic Search Methods for Large Speech Databases*, chapter 2: Keyword spotting methods, Springer, 2013.

[83] I. Szöke, P. Schwarz, P. Matejka, L. Burget, M. Karafiát, and J. Cernocky, "Phoneme based acoustics keyword spotting in informal continuous speech.," in *TSD*, V. Matousek, P. Mautner, and T. Pavelka, Eds. 2005, vol. 3658 of *Lecture Notes in Computer Science*, pp. 302–309, Springer.

[84] A. Jansen and P. Niyogi, "An experimental evaluation of keyword-filler hidden markov models," Tech. Rep., Department of Computer Science, University of Chicago, 2009.

[85] E. I. Chang and R. P. Lippmann, "Figure of merit training for detection and spotting," in *Proceedings of the 6th International Conference on Neural Information Processing Systems*, San Francisco, CA, USA, 1993, NIPS'93, pp. 1019–1026, Morgan Kaufmann Publishers Inc.

[86] A. Mesaros and T. Virtanen, "Automatic alignment of music audio and lyrics," in *DaFX-08*, Espoo, Finland, 2008.

[87] A. Loscos, P. Cano, and J. Bonada, "Low-delay singing voice alignment to text," in *Proceedings of the ICMC*, 1999.

[88] H. Fujihara and M. Goto, *Multimodal Music Processing*, chapter Lyrics-to-audio alignment and its applications, Dagstuhl Follow-Ups, 2012.

[89] A. M. Kruspe, "Keyword spotting in a-capella singing," in *15th International Conference on Music Information Retrieval (ISMIR)*, Taipei, Taiwan, 2014.

[90] J. Schlüter, "Learning to Pinpoint Singing Voice from Weakly Labeled Examples," in *Proceedings of the 17th International Society for Music Information Retrieval Conference (ISMIR 2016)*, New York, USA, 2016.

[91] C. kai Wang, R.-Y. Lyu, and Y.-C. Chiang, "An automatic singing transcription system with multilingual singing lyric recognizer and robust melody tracker.," in *INTERSPEECH*. 2003, ISCA.

[92] T. Hosoya, M. Suzuki, A. Ito, and S. Makino, "Lyrics recognition from a singing voice based on finite state automaton for music information retrieval," *Proceedings of the International Society for Music Information Retrieval Conference (ISMIR)*, pp. 532–535, 2005.

[93] M. Gales and P. Woodland, "Mean and variance adaptation within the mllr framework," *Computer Speech & Language*, vol. 10, pp. 249–264, 1996.

[94] M. Gruhne, K. Schmidt, and C. Dittmar, "Phoneme recognition in popular music," *ISMIR*, 2007.

[95] M. Gruhne, K. Schmidt, and C. Dittmar, "Detecting phonemes within the singing of polyphonic music," *Proceedings of ICoMCS ...*, , no. December, pp. 60–63, 2007.

[96] K. Dressler, "Sinusoidal Extraction using an efficient implementation of a multi-resolution {FFT}," in *Proc. of the 9th Int. Conference on Digital Audio Effects (DAFx-06)*, sep 2006, pp. 247–252.

[97] H. Hermansky, N. Morgan, A. Bayya, and P. Kohn, "Rasta-plp speech analysis technique," in *Proceedings of the 1992 IEEE International Conference on Acoustics, Speech and Signal Processing - Volume 1*, Washington, DC, USA, 1992, ICASSP'92, pp. 121–124, IEEE Computer Society.

[98] A. Härmä and U. K. Laine, "A comparison of warped and conventional linear predictive coding," *IEEE Trans. Speech and Audio Processing*, vol. 9, pp. 579–588, 2001.

[99] H. Fujihara, T. Kitahara, M. Goto, K. Komatani, T. Ogata, and H. Okuno, *Singer identification based on accompaniment sound reduction and reliable frame selection*, pp. 329–336, 2005.

[100] G. Szepannek, M. Gruhne, B. Bischl, S. Krey, T. Harczos, F. Klefenz, C. Dittmar, and C. Weihs, *Classification as a tool for research*, chapter Perceptually Based Phoneme Recognition in Popular Music, Springer, Heidelberg, 2010.

[101] H. Fujihara, M. Goto, and H. G. Okuno, "A novel framework for recognizing phonemes of singing voice in polyphonic music.," in *WASPAA*. 2009, pp. 17–20, IEEE.

[102] G. Fant, "The source filter concept in voice production," *Quarterly Progress and Status Report*, vol. 22, no. 1, pp. 021–037, 1981.

[103] A. Mesaros and T. Virtanen, "Adaptation of a speech recognizer for singing voice," *European Signal Processing Conference*, , no. 1, pp. 1779–1783, 2009.

[104] A. Mesaros and T. Virtanen, "Recognition of phonemes and words in singing," *Acoustics Speech and Signal ...*, pp. 1–4, 2010.

[105] A. Mesaros and T. Virtanen, "AUTOMATIC UNDERSTANDING OF LYRICS FROM SINGING," *Akustiikkapäivät*, pp. 1–6, 2011.

[106] T. Virtanen, A. Mesaros, and M. Ryynänen, "Combining pitch-based inference and non-negative spectrogram factorization in separating vocals from polyphonic music," in *ISCA Tutorial and Research Workshop on Statistical and Perceptual Audition, SAPA 2008, Brisbane, Australia, September 21, 2008*, 2008, pp. 17–22.

[107] A. Mesaros and T. Virtanen, "Automatic Recognition of Lyrics in Singing," *EURASIP Journal on Audio, Speech, and Music Processing*, vol. 2010, pp. 1–11, 2010.

[108] M. McVicar, D. Ellis, and M. Goto, "LEVERAGING REPETITION FOR IMPROVED AUTOMATIC LYRIC TRANSCRIPTION IN POPULAR MUSIC," *mattmcvicar.com*, pp. 3141–3145, 2014.

[109] J. G. Fiscus, "A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (rover)," 1997, pp. 347–352.

[110] M. Goto, H. Hashiguchi, T. Nishimura, and R. Oka, "Rwc music database: Popular, classical, and jazz music databases," in *In Proc. 3rd International Conference on Music Information Retrieval*, 2002, pp. 287–288.

[111] A. Loscos, P. Cano, and J. Bonada, "Low-delay singing voice alignment to text," *Proceedings of the ICMC*, 1999.

[112] Y. Wang, M.-Y. Kan, T. L. Nwe, A. Shenoy, and J. Yin, "LyricAlly: automatic synchronization of acoustic musical signals and textual lyrics," *Proceedings of the 12th annual ACM international conference on Multimedia*, vol. 0, pp. 212–219, 2004.

[113] M. Kan, Y. Wang, D. Iskandar, T. L. Nwe, and A. Shenoy, "Lyrically: Automatic synchronization of textual lyrics to acoustic music signals," *IEEE Trans. Audio, Speech & Language Processing*, vol. 16, no. 2, pp. 338–349, 2008.

[114] D. Iskandar, Y. Wang, M.-Y. Kan, and H. Li, "Syllabic level automatic synchronization of music signals and text lyrics," in *Proceedings of the 14th ACM International Conference on Multimedia*, New York, NY, USA, 2006, MM '06, pp. 659–662, ACM.

[115] A. Sasou, M. Goto, S. Hayamizu, and K. Tanaka, "An auto-regressive, non-stationary excited signal parameter estimation method and an evaluation of a singing-voice recognition," in *ICASSP*, 2005.

[116] K. Chen, S. Gao, Y. Zhu, and Q. Sun, "Popular song and lyrics synchronization and its application to music information retrieval," *SPIE*, 2006.

[117] H. Fujihara, M. Goto, J. Ogata, K. Komatani, T. Ogata, and H. G. Okuno, "Automatic synchronization between lyrics and music CD recordings based on viterbi alignment of segregated vocal signals," in *Eigth IEEE International Symposium on Multimedia (ISM 2006), 11-13 December 2006, San Diego, CA, USA*, 2006, pp. 257–264.

[118] H. Fujihara and M. Goto, "Three techniques for improving automatic synchronization between music and lyrics: Fricative detection, filler model, and novel feature vectors for vocal activity detection," in *IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, Las Vegas, NV, USA, 2008, pp. 69–72.

[119] H. Fujihara, M. Goto, J. Ogata, and H. G. Okuno, "LyricSynchronizer: Automatic Synchronization System Between Musical Audio Signals and Lyrics," *J. Sel. Topics Signal Processing*, vol. 5, no. 6, pp. 1252–1261, 2011.

[120] M. Mauch, H. Fujihara, and M. Goto, "Lyrics-to-audio alignment and phrase-level segmentation using incomplete internet-style chord annotations," in *Proceedings of the 7th Sound and Music Computing Conference (SMC 2010)*, 2010.

[121] M. Mauch, H. Fujihara, and M. Goto, "Integrating additional chord information into hmm-based lyrics-to-audio alignment," *Trans. Audio, Speech and Lang. Proc.*, vol. 20, no. 1, pp. 200–210, Jan. 2012.

[122] C. H. Wong, W. M. Szeto, and K. H. Wong, "Automatic lyrics alignment for cantonese popular music," *Multimedia Syst.*, vol. 12, no. 4-5, pp. 307–323, 2007.

[123] K. Lee and M. Cremer, "Segmentation-based lyrics-audio alignment using dynamic programming," in *ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*, 2008, pp. 395–400.

[124] R. Gong, P. Cuvillier, N. Obin, and A. Cont, "Real-time audio-to-score alignment of singing voice based on melody and lyric information," in *INTERSPEECH 2015, 16th Annual Conference of the International Speech Communication Association, Dresden, Germany, September 6-10, 2015*. 2015, pp. 3312–3316, ISCA.

[125] G. Dzhambazov, A. Srinivasamurthy, S. Sentürk, and X. Serra, "On the use of note onsets for improved lyrics-to-audio alignment in turkish makam music," in *Proceedings of the 17th International Society for Music Information Retrieval Conference, ISMIR 2016, New York City, United States, August 7-11, 2016*, M. I. Mandel, J. Devaney, D. Turnbull, and G. Tzanetakis, Eds., 2016, pp. 716–722.

[126] M. Suzuki, T. Hosoya, A. Ito, and S. Makino, "Music information retrieval from a singing voice based on verification of recognized hypotheses," in *Proceedings of the 7th International Conference on Music Information Retrieval*, R. Dannenberg, K. Lemstrom, and A. Tindale, Eds., Victoria, BC, Canada, Oct. 2006, University of Victoria, pp. 168–171, University of Victoria.

[127] M. Suzuki, T. Hosoya, A. Ito, and S. Makino, "Music information retrieval from a singing voice using lyrics and melody information," *EURASIP J. Adv. Sig. Proc.*, 2007.

[128] C.-C. Wang, J.-s. R. Jang, and W. Wang, "An Improved Query by Singing/Humming System Using Melody and Lyrics Information," *11th ISMIR*, , no. Ismir, pp. 45–50, 2010.

[129] A. Mesaros and T. Virtanen, "Recognition of phonemes and words in singing.," in *ICASSP*. 2010, pp. 2146–2149, IEEE.

[130] A. Mesaros and T. Virtanen, "Automatic recognition of lyrics in singing.," *EURASIP J. Audio, Speech and Music Processing*, vol. 2010, 2010.

[131] W.-H. Tsai and H.-M. Wang, "Towards Automatic Identification Of Singing Language In Popular Music Recordings," in *5th International Conference on Music Information Retrieval (ISMIR)*, Barcelona, Spain, 2004, pp. 568–576.

[132] J. Schwenninger, R. Brueckner, D. Willett, and M. E. Hennecke, "Language Identification in Vocal Music," in *7th International Conference on Music Information Retrieval (ISMIR)*, Victoria, Canada, 2006, pp. 377–379.

[133] V. Chandraskehar, M. E. Sargin, and D. A. Ross, "Automatic language identification in music videos with low level audio and visual features," in *Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, Prague, Czech Republic, 2011, pp. 5724–5727.

[134] H. Fujihara, M. Goto, and J. Ogata, "Hyperlinking lyrics: A method for creating hyperlinks between phrases in song lyrics," in *ISMIR 2008, 9th International Conference on Music Information Retrieval, Drexel University, Philadelphia, PA, USA, September 14-18, 2008*, 2008, pp. 281–286.

[135] G. Dzhambazov, S. Sentürk, and X. Serra, "Searching lyrical phrases in a-capella turkish makam recordings," in *Proceedings of the 16th International Conference on Music Information Retrieval (ISMIR)*, Malaga, Spain, 2015.

[136] J. S. Garofolo et al., "TIMIT Acoustic-Phonetic Continuous Speech Corpus," Tech. Rep., Linguistic Data Consortium, Philadelphia, 1993.

[137] V. Zue, S. Seneff, and J. Glass, "Speech database development at MIT: Timit and beyond," *Speech Communication*, vol. 9, no. 4, pp. 351–356, 1990.

[138] A. Martin and M. Pryzbocki, "2003 NIST Language Recognition Evaluation," Tech. Rep., Linguistic Data Consortium, Philadelphia, 2006.

[139] J. C. Smith, *Correlation analyses of encoded music performance*, Ph.D. thesis, Stanford University, 2013.

[140] S. J. Young, D. Kershaw, J. Odell, D. Ollason, V. Valtchev, and P. Woodland, *The HTK Book Version 3.4*, Cambridge University Press, 2006.

[141] C. Jankowski, A. Kalyanswamy, S. Basson, and J. Spitz, "NTIMIT: A phonetically balanced, continuous speech telephone bandwidth speech database," *ICASSP*, pp. 109–112, 1990.

[142] H.-G. Hirsch and D. Pearce, "The Aurora experimental framework for the performance evaluation of speech recognition systems under noisy conditions," in *ISCA ITRW ASR*, 2000, pp. 29–32.

[143] D. P. W. Ellis, "A phase vocoder in Matlab," 2002, Web resource, Last checked: 04/29/15.

[144] R. M. G. J. L. Flanagan, "Phase vocoder," *Bell System Technical Journal*, pp. 1493–1509, Nov. 1966.

[145] M. Dolson, "The phase vocoder: A tutorial," *Computer Music Journal*, vol. 10, no. 4, pp. 14–27, 1986.

[146] C. Arft, "AutoTune Toy," 2010, Web resource, Last checked: 4/29/15.

[147] D. Jurafsky and J. H. Martin, *Speech and language processing: An introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*, Prentice Hall, 2009.

[148] H. Eghbal-zadeh, B. Lehner, M. Schedl, and G. Widmer, "I-vectors for timbre-based music similarity and music artist classification," in *Proceedings of the 16th International Society for Music Information Retrieval Conference, ISMIR 2015, Málaga, Spain, October 26-30, 2015*, M. Müller and F. Wiering, Eds., 2015, pp. 554–560.

[149] H. Eghbal-zadeh, M. Schedl, and G. Widmer, "Timbral modeling for music artist recognition using i-vectors," in *European Signal Processing Conference (EUSIPCO)*, Nice, France, 2015.

[150] A. Martin and C. Greenberg, "The 2009 NIST Language Recognition Evaluation," *Proceedings of Odyssey*, , no. July, pp. 165–171, 2010.

[151] A. J. K. Thambiratnam, *Acoustic keyword spotting in speech with applications to data mining*, Ph.D. thesis, Queensland University of Technology, 2005.

[152] J. D. Ferguson, "Variable duration models for speech," in *Proc. Symp. Applications Hidden Markov Models Text Speech*, Princeton, NJ, 1980.

[153] B. H. Juang, L. R. Rabiner, S. E. Levinson, and M. M. Sondhi, "Recent developments in the application of hidden markov models to speaker-independent isolated word recognition," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1985.

[154] S. E. Levinson, "Continuously variable duration hidden markov models for speech analysis," in *IEEE Int. Conf. on Acoustics, Speech, and Signal Processing (ICASSP)*, 1986.

[155] D. Burshtein, "Robust parametric modeling of durations in Hidden Markov Models," *IEEE Trans. ASSP*, vol. 4, no. 3, May 1996.

[156] A. M. Kruspe, "Keyword spotting in a-capella singing with duration-modeled HMMs," in *EUSIPCO*, Nice, France, 2015.

[157] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, and B. Kingsbury, "Deep Neural Networks for Acoustic Modeling in Speech Recognition," *Signal Processing Magazine*, 2012.

[158] J. Sundberg, *The Psychology of Music*, chapter 6. Perception of singing, Academic Press, 3 edition, 2012.

[159] A. L. Wang, "An industrial-strength audio search algorithm," in *Proceedings of the 4th International Conference on Music Information Retrieval (ISMIR)*, Baltimore, MD, USA, 2003, pp. 7–13.

List of Figures

2.1 Schematic of the training procedure used in the considered tasks. . . . 3
2.2 Schematic of the classification procedure used in the considered tasks (the model is the one created during training - see figure 2.1). 3
2.3 Example of a Mel filterbank. [4] 5
2.4 The SDC calculation at frame t. [8] 6
2.5 Comparison of the processing steps in MFCC (left) and PLP (right) calculation. [12] 7
2.6 Mel-scale (top) and Bark-scale filterbank. [12] 8
2.7 Perceptually motivated equal-loudness weighting function. (The dashed function is used for signals with a Nyquist frequency $> 5kHz$). [12] . . 8
2.8 The temporal paradigm for TRAP extraction versus conventional features (e.g. MFCC). [18] 10
2.9 TRAP extraction process. [21] 11
2.10 Mean TRAPs of various phonemes in the 5th critical band. [18] 11
2.11 Example of a DTW alignment. Alignment between points is represented by the arrows. [2] 12
2.12 Example of a matrix of costs between two sequences X and Y using the Manhattan distance as the local cost measure. [2] 13
2.13 Example of a Levenshtein distance calculation between the two strings "INTENTION" and "EXECUTION". Found operations (**d**eletions, **i**nsertions, **s**ubstitutions) are shown at the bottom. [28] 15
2.14 Visualization of a GMM. The Gaussian mixture is the weighted sum of several Gaussian distributions, where p_i are the mixture weights and b_i are the Gaussians. [30] 16

2.15 Example of a GMM in two dimensions. Crosses signify the data points, ellipses the three multivariate Gaussians. On the left-hand side, the evolution of the estimated means and covariances of these Gaussians during the EM process is shown; the right-hand side shows the converged mixture. [31] 17
2.16 A HMM as it is commonly used in ASR, with phonemes as hidden states and acoustic feature vectors as the observations. $a_12, a_22, a_23, ...$ are elements of the transition matrix A; $b_2(y1_)), b_2(y_2), b_3(y_3)$ are elements of the output probability matrix B. [43] 20
2.17 A set of data points which cannot be separated linearly in their original form (left), but can be separated after transformation into another space (right) [50] 22
2.18 A set of data points which cannot be separated using a linear kernel (left), but can be separated with a polynomial kernel (right) [51] 22
2.19 Functionality of a single neuron in an ANN. The neuron computes a weighted sum of its inputs, and then applies an activation function, resulting in output y. [68] 25
2.20 Activation functions used for ANN neurons. [69] 26
2.21 Schematic of a feed-forward neural network with an input layer with four neurons, two hidden layers with four neurons each, and an output layer with two neurons. [69] 27
2.22 Schematic of the training procedure for phoneme recognition. 30
2.23 Schematic of the classification procedure for phoneme recognition. . . . 30
2.24 Example of a phoneme posteriorgram, the result of classification with an acoustic model. The posteriorgram represents the posterior probabilities of each phoneme over time. The time resolution in this example is $10ms$. 32
2.25 Schematic of the training procedure for language identification. 32
2.26 Schematic of the classification procedure for language identification. . . 32
2.27 Keyword-filler HMM for the keyword "greasy" with filler path on the left hand side and two possible keyword pronunciation paths on the right hand side. The parameter β determines the transition probability between the filler HMM and the keyword HMM. [80] 35
2.28 Schematic of the procedure for keyword spotting. 35
2.29 Schematic of the procedure for lyrics-to-audio alignment. 36
2.30 Schematic of the procedure for lyrics retrieval. 36

3.1 Standard deviations of phoneme durations in the *TIMIT* and *ACAP* data sets. 39

5.1 Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT*. 61
5.2 Overview of the "songified" phoneme recognition system 62
5.3 Mean phoneme recognition results on the test data sets using acoustic models trained on *TIMIT* and augmented versions thereof. 65
5.4 An overview of the alignment process. The right-hand part represents the optional bootstrapping. 66
5.5 Mean alignment error in seconds on the *ACAP* data set. *TIMIT* shows the result for the same models used for aligning the new *DAMP*-based data sets. 68
5.6 Mean phoneme recognition results on the *ACAP* data set using acoustic models trained on *Timit* and the new *DAMP*-based data sets. 69
5.7 Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets. 69
5.8 Mean phoneme recognition results on the *DampTest* data sets using acoustic models trained on *TIMIT* and the new *DAMP*-based data sets. 69

6.1 Overview of the process for language identification using i-vector extraction. 74
6.2 Results using MLP models on all three language identification data sets, with or without i-vector processing. 76
6.3 Results using SVM models on all three language identification data sets, with or without i-vector processing, with speakers shared between training and test sets. 76
6.4 Results using SVM models on all three language identification data sets, with or without i-vector processing, with speakers separated between training and test sets. 78
6.5 Document-wise results using SVM models, with or without i-vector processing. 79
6.6 Overview of the process for language identification using phoneme statistics. 80
6.7 Results using document-wise phoneme statistics generated with various acoustic models. 81

6.8 Results using utterance-wise phoneme statistics generated with various acoustic models. 82
6.9 Results using utterance- and document-wise i-vectors calculated on PLP and MFCC features. 83

7.1 F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models. 87
7.2 F_1 measures for keyword spotting results on the *DampTestM* and *DampTestF* data sets using mixed and gender-dependent models. 89
7.3 Individual F_1 measures for the results for each keyword, using the acoustic model trained on *DampB*. 89
7.4 F_1 measures for keyword spotting results using posteriorgrams generated with various acoustic models with post-processor duration modeling. . . 92

8.1 *MIREX* results on all three data sets using the HMM-based approach. 96
8.2 Overview of the DTW-based lyrics alignment and retrieval method. . . 97
8.3 Example of a similarity calculation: Phoneme posteriorgrams are calculated for the audio recordings **(a)**. Phoneme templates are generated for the textual lyrics **(b)**. Then, a similarity matrix is calculated using the cosine distance between the two, and DTW is performed on it **(c)**. The accumulated cost divided by the path length is the similarity measure. 99
8.4 *MIREX* results on all three data sets using the DTW-based approach. 100
8.5 Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the 1-, 3-, and 10-best results. 100
8.6 Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the DTW-based approach with five different acoustic models, and evaluated on the 1-, 3-, 10-, 50-, and 100-best results. 102
8.7 Overview of the phoneme-based lyrics retrieval process. 103
8.8 Example of the block grouping of the phoneme sequence and subsequent filtering by probabilities and confusions. 104
8.9 Example of a confusion matrix for an acoustic model. (Note that *[pau]* confusions are set to 0). 105
8.10 Results of the phoneme-based retrieval algorithm with various improvement steps for three small calibration data sets. 106
8.11 *MIREX* results on all three data sets using the phoneme-based approach. 107

8.12 Accuracies of the results for lyrics detection on the whole song for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the 1-, 3-, and 10-best results. 110
8.13 Accuracies of the results for lyrics detection on separate lines of sung lyrics for the *DampTest* sets using the phoneme-based approach with five different acoustic models, and evaluated on the 1-, 3-, 10-, 50-, and 100-best results. 110
8.14 Data flow in the expletive detection approach. 111
8.15 Results for the expletive detection approach at various tolerances. . . . 112

List of Tables

4.1 Amounts of data in the three used data sets: Sum duration on top, number of utterances in italics. 54
4.2 Overview of the structure of the *DAMP*-based phonetically annotated data sets, number of recordings in brackets, duration in italics (dd:hh:mm:ss). Note that no number of recordings is given for some data sets because their content was selected phoneme-wise, not song-wise. For retrieval, the recordings are short phrases instead of full songs. 58
4.3 All 15 tested keywords, ordered by number of phonemes. 59

5.1 The five TIMIT variants that were used for training (rows are TIMIT blocks, columns are the five datasets). Symbols: N - Unmodified; P - Pitch-shifted; T - Time-stretched; V - Vibrato 64

6.1 Feature configurations used in training. 75

List of Abbreviations

ANN	Artificial Neural Network
AR-HMM	Auto-Regressive HMM
ASR	Automatic Speech Recognition
CNN	Convolutional Neural Network
DAMP	Digital Archive of Mobile Performances
DCT	Discrete Cosine Transform
DFT	Discrete Fourier Transform
DNN	Deep Neural Network
DTW	Dynamic Time Warping
EM	Expectation Maximization
F_0	Fundamental frequency
F_1	F_1 measure
FFT	Fast Fourier Transform
FSA	Finite State Automaton
FSM	Finite State Machine
GMM	Gaussian Mixture Model
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
LPC	Linear Predictive Coding
LRE	Language Recognition Evaluation
LSTM	Long Short-Term Memory
LVCSR	Large-Vocabulary Continuous Speech Recognition
MAP	Mean Average Precision
MFC	Mel-Frequency Cepstrum
MFCC	Mel-Frequency Cepstral Coefficient(s)
MIR	Music Information Retrieval
MIREX	Music Information Retrieval Evaluation eXchange
MLLR	Maximum Likelihood Linear Regression
MLP	Multilayer Perceptron

NIST	National Institute for Standards and Technology
NN	Neural Network
OGI	Oregon Institute for Science and Technology
PER	Phoneme Error Rate
PLP	Perceptual Linear Prediction
PPR	Parallel Phoneme Recognition
PPRLM	Parallel Phoneme Recognition followed by Language Modeling
QMUL	Queen Mary University of London
RASTA	RelAtive SpecTrA
RBF	Radial Basis Function
ReLU	Rectified Linear Unit
RNN	Recursive Neural Network
ROVER	Recogniser Output Voting Error Reduction
SAI	Stabilized Auditory Images
SDC	Shifted Delta Cepstrum
STFT	Short-term Fourier Transform
SVM	Support Vector Machine
TRAP	TempoRAl Pattern
UBM	Universal Background Model
VAD	Vocal Activity Detection
VTHMM	Variable Time HMM
WLPC	Warped Linear Predictive Coding

A Appendix

A lot of stuff that didn't fit into the main part ...

B Eigenständigkeitserklärung

Die vorliegende Arbeit habe ich selbstständig ohne Benutzung anderer als der angegebenen Quellen angefertigt.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten Quellen entnommen wurden, sind als solche deutlich kenntlich gemacht. Die Arbeit ist in gleicher oder ähnlicher Form oder auszugsweise im Rahmen einer oder anderer Prüfungen noch nicht vorgelegt worden.

Ilmenau, 17.12.2013

Sheldon Cooper

Thesis Summary

1. Scissors cuts paper, paper covers rock, rock crushes lizard, lizard poisons Spock, Spock smashes scissors, scissors decapitates lizard, lizard eats paper, paper disproves Spock, Spock vaporizes rock, and as it always has, rock crushes scissors.
2. I'm not insane, my mother had me tested!
3. All I need is a healthy ovum and I can grow my own Leonard Nimoy!

Thesen

1. These 1
2. These 2
3. These 3