

KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Leuven

DISTANCE MEASURES FOR TEMPLATE BASED SPEECH AND PATTERN RECOGNITION

Promotor:
Prof. dr. ir. R. Cools
Prof. dr. ir. D. Van Compernolle

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Mike MATTON

9 november 2009



KATHOLIEKE UNIVERSITEIT LEUVEN
FACULTEIT INGENIEURSWETENSCHAPPEN
DEPARTEMENT COMPUTERWETENSCHAPPEN
AFDELING NUMERIEKE ANALYSE EN
TOEGEPASTE WISKUNDE
Celestijnenlaan 200A – B-3001 Leuven

DISTANCE MEASURES FOR TEMPLATE BASED SPEECH AND PATTERN RECOGNITION

Jury:
Prof. dr. ir. P. Van Houtte, voorzitter
Prof. dr. ir. R. Cools, promotor
Prof. dr. ir. D. Van Compernolle, co-promotor
Prof. dr. ir. P. Wambacq,
Prof. dr. ir. S. Vandewalle,
Prof. dr. ir. H. Blockeel,
Dr. L. ten Bosch, Radboud Universiteit Nijmegen

Proefschrift voorgedragen tot
het behalen van het doctoraat
in de ingenieurswetenschappen

door

Mike MATTON

9 november 2009

© Katholieke Universiteit Leuven — Faculteit Ingenieurswetenschappen
Arenbergkasteel, B-3001 Leuven, Belgium

Alle rechten voorbehouden. Niets uit deze uitgave mag worden vermenigvuldigd en/of openbaar gemaakt worden door middel van druk, fotokopie, microfilm, elektronisch of op welke andere wijze ook zonder voorafgaande schriftelijke toestemming van de uitgever.

All rights reserved. No part of the publication may be reproduced in any form by print, photoprint, microfilm or any other means without written permission from the publisher.

D/2009/7515/122 ISBN 978-94-6018-138-2

Woord Vooraf

En dan is het eindelijk zover. Na zes jaar is dit doctoraatsproefschrift afgewerkt. Dat het maken van een doctoraat niet altijd over rozen gaat, daar zal niemand aan twijfelen. Het maken van een doctoraat heeft vele ups en downs. Het afronden van dit doctoraat zou echter niet mogelijk zijn zonder de hulp en steun van vele mensen. Alle namen opnoemen zou mij echter veel te ver leiden. Daarom beperk ik mij tot de dichtste vrienden en mensen die een steentje bijgedragen hebben aan deze tekst. Voor hen dit oprecht woord van dank.

De eerste persoon die ik wil bedanken is mijn promotor, Prof. Ronald Cools. In de eerste plaats wil ik hem bedanken voor de kans die ik gekregen hebben om een doctoraat te mogen maken en het nadien ook af te ronden. Bovendien moet ik hem bedanken voor de vele hints en tips die hij gegeven heeft, het nalezen van papers en verslagen en nog zoveel meer... Hij volgde ook de praktische kant van het doctoraat goed op en herinnerde er altijd op tijd aan om het nodige papierwerk in orde te brengen.

Ook mijn co-promotor Dirk Van Compernolle wil ik bedanken voor de vele suggesties die hij tijdens de loop van het doctoraat gegeven heeft. Ook wil ik hem bedanken voor het nalezen van deze verhandeling en de vele suggesties ter verbetering die hij gegeven heeft.

Naast de promotoren wil ik graag ook alle juryleden bedanken: Prof. Paul Van Houtte om als voorzitter te fungeren, Prof. Patrick Wambacq, Prof. Stefan Vandewalle, Prof. Hendrik Blockeel en Dr. Louis Ten Bosch voor de bereidheid om deel uit te maken van mijn doctoraatsjury en voor het nalezen van de tekst.

Mijn voormalige collega Mathias De Wachter wil ik bedanken voor

de vele input die hij gegeven heeft voor het onderzoek. Hij heeft mij op weg geholpen bij het begin van mijn doctoraat en heeft ook heel dikwijls nuttige tips gegeven.

Verder wil ik mijn bureaugenoten Bart Vandewoestyne en Yves Frederix danken voor de aangename werksfeer en voor hun aanspreekbaarheid als ik weer eens een technisch probleem had. Yves wil ik in het bijzonder danken voor het nalezen van de Nederlandse samenvatting. Ook collega Dirk Nuyens wil ik bedanken voor de suggesties die hij gegeven heeft ter verbetering van de tekst.

Een volgende groep mensen die ik wil bedanken zijn de tafelgenoten in alma, Ares, Celine, Pieter, Bart, Toon & co. wil ik bedanken voor de boeiende, interessante, maar dikwijls ook absurde discussies die de middagpauze opvrolijkten. Ik zal de discussies missen ;-). Ook de vrienden die van de maandagavond een ontspannende avond maakten wil ik niet vergeten: bedankt Andy, Ares, Celine, Erwin, Isabelle, Lotte, Olivier, Pieter B., Pieter V. en Stephanie!

Werk en ontspanning gaan samen, dat is algemeen geweten. Bij mij staat ontspanning gelijk aan het bespelen van een muziekinstrument. Ik ben dan ook heel blij dat ik mijn momenten van vrije tijd kon doorbrengen in verschillende muziekverenigingen. Ik wil hierbij alle mede-muzikanten uit Muziekvereniging Bellegem en het Universitair Harmonieorkest bedanken voor de aangename afwisseling en de leuke sfeer. Eén persoon heeft een steentje bijgedragen aan het tot stand komen van dit werk. Bedankt Sanne voor het kiezen en samenstellen van de kaft.

Tot slot wil ik mijn ouders bedanken om mij de mogelijkheid te geven hogere studies aan te vatten en de vele dingen die ik daardoor heb bijgeleerd.

List of Symbols

$\ell(.)$	A loss function
γ_0, γ_1, γ_2	DTW local path costs
Λ	Weight matrix, or set of weighting parameters
λ	A single weight parameter
$\mathbb{S}$	Set of HMM states
$\mathbf{A} = \{a_{ij}\}$	HMM transition probabilities
$\mathbf{B} = \{b_i(\mathbf{x})\}$	HMM observation probabilities
$\mathbf{s}, \mathbf{s}_1^T$	Sequence of HMM states (of length T)
$\mathbf{t}$	A template sequence
$\mathbf{w}$	A word sequence
$\mathbf{w}_i$	A single word
$\mathbf{x}$	Sequence of observations or single observation, depending on the context
$\mathbf{x}_1^T$	Sequence of acoustic observations (feature vectors) of length T
$\mathbf{x}_{\text{nn}}^{=}$	Nearest neighbor of $\mathbf{x}$ in the same class

$\mathbf{x}_{\text{nn}}^{\neq}$	Nearest neighbor of $\mathbf{x}$ in a different class
$\mathbf{x}_i$	Single observation
$\mathcal{X}$	A feature (observation) space
$\mathcal{X}_k$	Part of feature space that causes misclassification for class ω_k
Ω	HMM feature space
$\Omega(\mathbf{x})$	Classification decision made for $\mathbf{x}$
ω_i	A class label in the classification problem
ω_k	A class label for the classification problem
Φ	The parameter set of a Hidden Markov Model
$\phi(.)$	Parzen window
$\phi_x(k)$	DTW warping function
$\boldsymbol{\mu}$	Mean
$\boldsymbol{\mu}_k$	Mean vector of observations in class ω_k
Σ	Covariance matrix
C_p	A class label for the distance measure
D	Number of dimensions
$D(\mathbf{x};\mathbf{y})$	Distance between templates, usually DTW
$d(\mathbf{x};\mathbf{y})$	Distance between single observations
$d_k(\mathbf{x},\Lambda)$	Misclassification measure for feature vector $\mathbf{x}$ of class ω_k with parameters Λ
$E\,[.]$	Expected value

$g_k(.)$	A discriminant function
$H(.)$	Entropy
$I(.)$	Mutual information
$J(\Lambda)$	Optimization criterion for the parameter set Λ
K	Number of classes of the classification problem
$K(.)$	Generalized Parzen window
$L(.)$	Total loss function
N	Number of elements, prototypes, observations, depending on the context
N_K	Number of classes / clusters for the distance measure
N_k	Number of elements in class k
N_S	Number of HMM states
$P(.)$	Probability
$p(.)$	Probability density or likelihood function
R	Overall risk
S	Sample covariance matrix (scatter matrix)
T, T_1, T_2	Number of observations in a sequence
Y	Set of training examples

Glossary

k-NN k-nearest neighbors, 83

ASR Automatic speech recognition, 1

CD Context dependent, 24
CV Cross validation, 138

DTW Dynamic time warping, 29

EM Expectation maximization, 12

GMM Gaussian mixture model, 11
GPD Generalised probabilistic descent, 101

HMM Hidden Markov Model, 5
HTK Hidden Markov Model Toolkit, 145

LDA Linear discriminant analysis, 57
LM Language model, 15
LVSR Large vocabulary speech recognition, 34

MCE Minimum classification error, 92
MFCC Mel frequency cepstral coefficients, 4
MIDA Mutual information discriminant analysis, 58

ML Maximum likelihood, 12
MLE Maximum likelihood estimation, 87
MMI Maximum mutual information, 89
MPE Minimum phone error, 98

PCA Principal component analysis, 56

SLP Spoken language processing, 1
SME Soft margin estimation, 97

TBSR Template based speech recognition, 23

WER Word error rate, 141

List of Figures

1.1 Basic structure of a continuous ASR system 4
1.2 Preprocessing in speech recognition: schematic overview . . 6
1.3 A typical HMM used in speech recognition 8
1.4 The forward algorithm [41] 10
1.5 Trellis for the forward algorithm 11
1.6 The Viterbi algorithm [89] 12

2.1 Template alignment: time warping 28
2.2 DTW local path constraints: Sakoe and Itakura 29
2.3 DTW: global path constraints in the Itakura case 30
2.4 A template "model" 32
2.5 Template based speech recognition: system lay-out 33
2.6 The search space for connected digit recognition 34
2.7 The time filter algorithm [20] 37
2.8 Example run of the time filter algorithm 38
2.9 Example activation graph 39
2.10 Token passing algorithm in one single model [93] 41

3.1 Gaussian data distribution and its normalized form 47
3.2 Prototype classification, two classes with Gaussian data distribution 50
3.3 Sparse weight matrices: weight selection 54
3.4 A block-diagonal weight matrix 55

4.1 A binary decision tree 65

4.2 The k-means clustering algorithm 68
4.3 The ISODATA clustering algorithm 69
4.4 Combined clustering 72

5.1 A typical classification system 76
5.2 Some loss functions 96

6.1 The kd-tree construction algorithm 117
6.2 Point cloud and its corresponding kd-tree 117
6.3 The kd-tree search algorithm 120
6.4 Kd-tree standard search algorithm 121
6.5 The roadmap training algorithm 125
6.6 Searching (approximate) NN with a kd-tree for varying ϵ. . 128
6.7 Scale-up experiment with kd-trees 129
6.8 Comparison of kd-tree and bbd-tree search algorithms . . . 130
6.9 Comparison between standard kd-tree search and priority search. 131
6.10 Comparison of splitting rules. top line is standard split, bottom line is sliding midpoint split 132
6.11 (a) Mean execution time as function of number of requested nearest neighbors. (b) Log-plot of (a) 133
6.12 (a) Mean execution times for increasing k. (b) Log-plot showing k on the x-axis and $t(k)/k$ on the y-axis. 134
6.13 Plot showing k on the x-axis and $t(k)/k$ on the y-axis . . . 135

A.1 Sparse BLAS distance calculation 157

Contents

List of Symbols **iii**

List of Abbreviations **vii**

List of Figures **ix**

Contents **xi**

1 Introduction **1**
1.1 Introduction: (automatic) spoken language processing . . . 1
1.2 Basic equation . . . 3
1.3 Feature extraction . . . 5
1.4 HMM-based acoustic modeling . . . 5
1.4.1 A Hidden Markov Model (HMM) . . . 5
1.4.2 Evaluating Hidden Markov Models . . . 8
1.4.3 Decoding Hidden Markov Models . . . 10
1.4.4 Training Hidden Markov Models . . . 11
1.5 Language Modeling . . . 15
1.6 Motivation . . . 16
1.6.1 Shortcomings of HMM-based Speech Recognition . . 16
1.6.2 Availability of computing power . . . 18
1.7 Template based speech recognition . . . 19
1.8 Relation to other work . . . 20
1.9 Outline . . . 21

2 Template based speech recognition **23**
2.1 The basic unit: templates 23
2.2 TBSR: a statistical approach 25
2.3 Dynamic Time Warping 26
2.3.1 Distance between input and template 27
2.3.2 Distance calculation: Dynamic Time Warping 29
2.3.3 DTW and templates 31
2.4 A template based speech recognition system 32
2.4.1 Top-down or bottom-up? 33
2.4.2 Template selection: the time filter algorithm 35
2.4.3 Template matching: token passing search algorithm 40
2.4.4 Concatenating templates: transition costs 42

3 Distance measures **45**
3.1 Definition 45
3.2 The Euclidean distance measure 46
3.3 Scaling the distance measure 48
3.3.1 Global scaling of the distance 48
3.3.2 Class-dependent distance measures 49
3.4 Selecting the weight matrices 51
3.4.1 Diagonal weight matrices 51
3.4.2 Full weight matrices 52
3.4.3 Block diagonal weight matrices 53
3.4.4 Implementation 55
3.5 Global data decorrelation 56
3.5.1 Principal component analysis 56
3.5.2 Linear discriminant analysis 57
3.5.3 Mutual information discriminant analysis 58

4 Defining the classes **61**
4.1 Supervised clustering 62
4.1.1 Context-independent phoneme-based classes 62
4.1.2 Context-dependent phoneme-based classes 63
4.2 Unsupervised clustering methods 67
4.2.1 k-Means Clustering 68

4.2.2 Hierarchical clustering 70
4.2.3 Other clustering methods 71
4.3 Combining clustering methods 71
4.3.1 Combination on the context-independent phoneme-level 73
4.3.2 Combination on the context-dependent phoneme-level 73
4.3.3 Experimental results 74

5 Training Distance measures 75
5.1 Introduction: pattern classification 75
5.1.1 Description 75
5.1.2 Training a classifier 76
5.1.3 Classification in speech recognition 77
5.2 Decision theory 77
5.2.1 The Bayes classifier 77
5.2.2 Bayesian decision theory 78
5.2.3 Discriminant functions 79
5.2.4 The ideal classifier 80
5.3 Parametric vs. non parametric 80
5.3.1 Parametric likelihood functions 80
5.3.2 Non-parametric likelihood functions 82
5.4 Parameter estimation 85
5.4.1 Maximum likelihood parameter estimation 86
5.4.2 Discriminative training 88
5.4.3 Maximum Mutual Information 89
5.4.4 Minimum Classification Error 92
5.4.5 Margin-based estimation strategies 97
5.4.6 Minimum Phone Error 98
5.4.7 Unified view 99
5.4.8 Discriminative training and speech recognition . . . 99
5.4.9 Other discriminative criteria 100
5.5 Optimization of criteria 101
5.5.1 Gradient descent 101
5.5.2 Generalized probabilistic descent 102
5.6 Frame-based training 103

5.6.1 Maximum Likelihood based distance measures . . . 103
5.6.2 Discriminative training of frame-based distance measures . . . 104
5.7 Unit-based training of distance measures . . . 108
5.7.1 Phone-based distance measures . . . 109
5.7.2 Sentence-based distance measures . . . 110

6 k-nearest neighbors 113
6.1 Introduction . . . 113
6.1.1 Classification of nearest neighbor methods. . . . 114
6.1.2 The Curse of Dimensionality . . . 114
6.2 Tree Structures . . . 115
6.2.1 Kd-trees . . . 116
6.2.2 Bbd-trees . . . 122
6.3 Graph structures . . . 123
6.3.1 The Roadmap algorithm . . . 123
6.3.2 The extended roadmap algorithm . . . 124
6.4 Experiments . . . 127
6.4.1 Exact vs Approximate k-nearest neighbors . . . 127
6.4.2 Scale-up . . . 129
6.4.3 Kd-trees vs Bbd-trees . . . 130
6.4.4 Standard vs Priority Search . . . 130
6.4.5 Splitting rules and the cell aspect ratio . . . 131
6.4.6 Empirical complexity of the search . . . 131
6.5 Discussion and conclusion . . . 136

7 Experimental results 139
7.1 Pattern classification . . . 139
7.1.1 Experiment setup . . . 139
7.1.2 Experimental results . . . 140
7.2 Speech recognition . . . 142
7.2.1 Experimental set-up . . . 143
7.2.2 Frame-based distance measures . . . 144
7.2.3 Unit-based distance measures . . . 145
7.2.4 Comparison with HMM based speech recognition . . . 147

8 Conclusions **149**
8.1 Main contributions 149
8.2 Main experimental results 152
8.3 Pointers for further research 153

A The Sparse BLAS **155**

B Benchmark databases **159**
B.1 Pattern classification 159
B.2 Speech recognition 163

Bibliography **165**

Nederlandse samenvatting **179**

Chapter 1

Introduction

1.1 Introduction: (automatic) spoken language processing

Speech recognition is a vital part of our human society, because most communication occurs through speech production and human speech recognition. The ability of correctly recognizing and understanding spoken word sequences is an ability that almost all people possess. Cognitive scientists try to unravel the mysteries of human speech processing by constructing models to represent the functioning of the human brain. These insights could then be transferred to automatic spoken language processing by attempting to implement these insights. The field of human language processing however is not a subject of this PhD. We will restrict ourselves to automatic spoken language processing in this dissertation.

Automatic spoken language processing (SLP) tries to achieve the capability of recognizing spoken word sequences automatically using a computing device. The field of spoken language processing incorporates a lot of different subfields, including speech recognition, speaker recognition or verification, keyword spotting, speech synthesis etc... The research performed for this dissertation is restricted to automatic speech recognition, however, as explained in § 1.8, influences from other subfields are always present.

Automatic speech recognition (ASR) itself can also be subdivided into different categories. The most important one is between isolated word recognition and continuous speech recognition. In isolated word recognition, words are uttered separately from each other, without any form of grammar. In continuous speech recognition the uttered words are connected and usually, a grammar describing possible word combinations is present.

A second distinction is made between speaker dependent and speaker independent speech recognition. In speaker dependent speech recognition, example data from the target speaker is available and can be used to train models used during recognition. In speaker independent speech recognition, this example data is not present and models must be trained with example data from other speakers.

Another classification that is usually made in speech recognition systems is the distinction between clean and noisy speech. In clean speech, background noise is negligible, whereas in noisy speech, background noise (a driving car, music, a noisy fan, ...) is present.

In this dissertation, we are concerned with creating an automatic speech recognition system for continuous, speaker independent speech recognition on clean speech.

The remainder of this introductory chapter is organized as follows. In § 1.2, the basic speech recognition equation is explained, leading to the structure of a typical automatic speech recognition system. Section 1.3 explains the conversion from the speech signal to sequences of feature vectors. In § 1.4, the basics of the mainstream approach for speech recognition (HMM-based SR) are explained. Section 1.6 discusses some drawbacks of HMM-based speech recognition and a motivation for the research presented in this work. Section 1.7 introduces the concept of template based speech recognition as an alternative to HMM-based speech recognition. Finally, § 1.8 presents a relation between the work performed for this research and work performed by other people.

1.2 Basic equation

Formally, speech recognition tries to find the optimal word sequence $\hat{\mathbf{w}}$ given a sequence of acoustic observations $\mathbf{x} = \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \ldots, \mathbf{x}_T$, with T the lengths of the acoustic observation. This leads to the fundamental speech recognition equation:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{x}), \tag{1.1}$$

where $P(\mathbf{w}|\mathbf{x})$ denotes the probability that the word sequence $\mathbf{w}$ was uttered given the acoustic observation sequence $\mathbf{x}$. Applying Bayes' rule transforms this equation to the form:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \frac{p(\mathbf{x}|\mathbf{w})P(\mathbf{w})}{p(\mathbf{x})}. \tag{1.2}$$

The factor $p(\mathbf{x}|\mathbf{w})$ is a probability density function of $\mathbf{x}$ given $\hat{\mathbf{w}}$ and $p(\mathbf{x})$ is the probability density function of $\mathbf{x}$. Since $p(\mathbf{x})$ is constant under the $\arg\max$ operator, this can be reduced to

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} p(\mathbf{x}|\mathbf{w})P(\mathbf{w}). \tag{1.3}$$

Equation (1.3) is often referred to as the basic speech recognition equation because it describes the separation of acoustic and language model which is typical in most speech recognition systems:

- $p(\mathbf{x}|\mathbf{w})$ is estimated by the acoustic models. The acoustic model describes the probability of an acoustic observation $\mathbf{x}$ given a word sequence $\mathbf{w}$. To solve this part of equation (1.3) acoustic models are trained on available labeled speech data. Some descriptions of acoustic models are presented in §1.4 for HMM-based speech recognition and §2.1 and 2.3 for template based speech recognition. Usually, one acoustic model per phoneme is created.

- $P(\mathbf{w})$ is estimated by the language model. It represents the probability of a word sequence $P(\mathbf{w})$. A small introduction on typical language models is presented in §1.5.

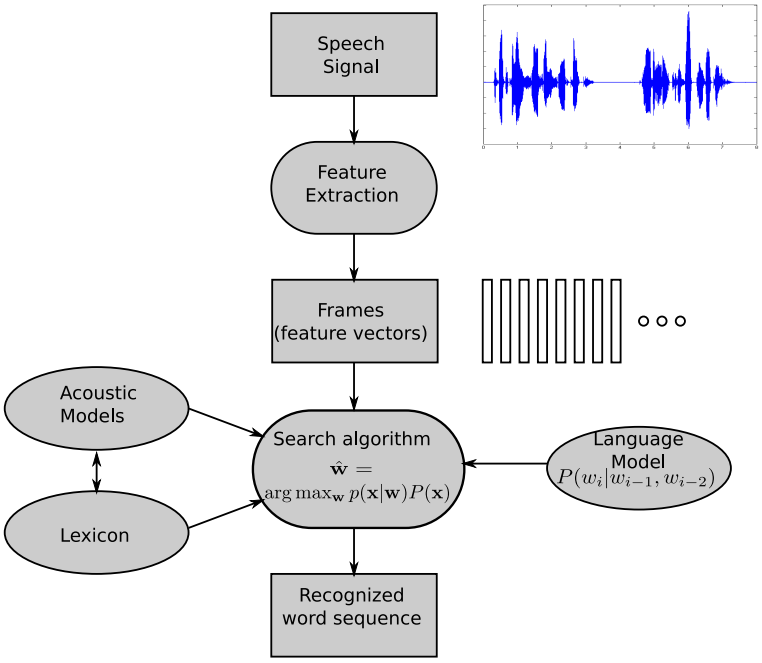


Figure 1.1: Basic structure of a continuous ASR system

Another important part of typical speech recognition systems is the lexicon (also called dictionary). The lexicon describes how to combine acoustic models (phonemes) to form words. It contains all words known to the ASR system and the series of phonemes that must be encountered to form that word.

Fig. 1.1 shows the basic structure of a typical continuous ASR system. The only part that has not yet been introduced is the preprocessing of the speech signal into a series of feature vectors. It is introduced in the next section.

1.3 Feature extraction

The first step in almost all speech recognition systems is the extraction of features from the acoustic data. Most systems make use of the Mel Frequency Cepstral Coefficients (MFCC) [19] to describe the speech signal. This procedure works as follows. First, the input signal is windowed. Typically, this is done with Hamming windows of 30 ms long and with a 20 ms overlap. Next, the spectrum is computed by taking the Fourier transform. These coefficients then are mapped onto the Mel scale by using a set of triangular-shaped filters. After taking the log of the powers (phase information is omitted because it contains no useful information and moreover the human ear is also phase-deaf), the resulting coefficients are treated as a signal and the inverse discrete cosine transform is taken. The resulting spectrum is called the Mel Frequency Spectrum and the resulting coefficients are called Mel Frequency Cepstral Coefficients. Usually the first 12 coefficients are used to describe the part of speech signal under the Hamming window, forming a feature vector. Next the energy of the signal, which also contains useful information is added to the feature vector. The window then shifts (10 ms) and a new feature vector is calculated. This procedure creates a time series of feature vectors from the continuous speech signal. Because speech is transient in nature, also first and second order time derivatives of the MFCC features are added to every feature vector. A schematic overview of this preprocessing step is shown in fig. 1.2.

1.4 HMM-based acoustic modeling

1.4.1 A Hidden Markov Model (HMM)

A Hidden Markov Model (HMM) is a statistical model that is able to characterize an input sample that consists of a discrete time series of observed data. In speech recognition, this discrete time series comes from the sequence of feature vectors generated by the preprocessing step.

An HMM consists of the following elements:

- a set of states $\mathbb{S} = \{1, 2, \ldots, N_S\}$,

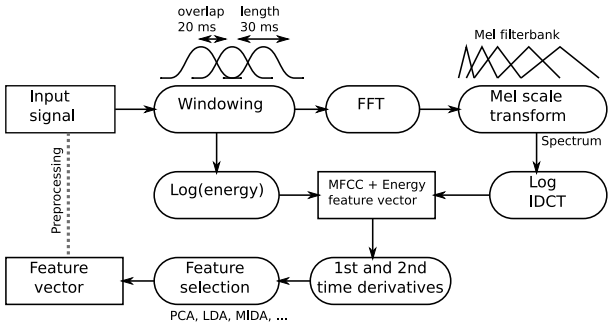


Figure 1.2: Preprocessing in speech recognition: schematic overview

- a feature space Ω, produced by the feature vectors of the observation,

- a set of transition probabilities $\mathbf{A} = \{a_{ij}\}$ representing the probability of making the transition from state i to state j; $i, j \in \mathbb{S}$

- a set of output probabilities $\mathbf{B} = \{b_i(\mathbf{x})\}$, with $b_i(\mathbf{x})$ representing the probability of observing $\mathbf{x}$ when entering state i, $i \in \mathbb{S}$

The Hidden Markov Model uses two sequences of random variables: the observation sequence $\mathbf{x}_1^T = \mathbf{x}_1, \ldots, \mathbf{x}_T$ and a sequence of states $\mathbf{s}_1^T = s_1, \ldots, s_T$, with $s_i \in \mathbb{S}, i = 1, \ldots, T$. The term "hidden" comes from the fact that this state sequence $\mathbf{s}$ is not observable. To complete the specification of an HMM, an initial state distribution π is needed: the probability that the first state of the alignment is state i: $\pi = \{\pi_i\}$ where $\pi_i = P(s_1 = i), 1 \leq i \leq N_S$.

Since an HMM is a probabilistic model, all probabilities must satisfy

the following constraints:

$$a_{ij} \geq 0, \quad b_i(\mathbf{x}) \geq 0, \quad \pi_i \geq 0, \quad \forall i, j, k \tag{1.4}$$

$$\sum_{j=1}^{N_S} a_{ij} = 1 \tag{1.5}$$

$$\int_{\Omega} b_i(\mathbf{x}) d\mathbf{x} = 1 \tag{1.6}$$

$$\sum_{i=1}^{N_S} \pi_i = 1. \tag{1.7}$$

Furthermore, the Hidden Markov Model must satisfy the first order Markov assumption and the output independence assumption. The first order Markov assumption can be described as follows:

$$P(s_t|\mathbf{s}_1^{t-1}) = P(s_t|s_{t-1}), \tag{1.8}$$

where $\mathbf{s}_1^{t-1}$ denotes the sequence of states $s_1, \ldots, s_{t-1}$. The first order Markov assumption states that the probability of entering a specific state depends only on the previous state. The second assumption is the output independence assumption. It can be formulated as follows:

$$p(\mathbf{x}_t|\mathbf{x}_1^{t-1}, \mathbf{s}_1^t) = p(\mathbf{x}_t, s_t). \tag{1.9}$$

In speech recognition, HMM's are used to define acoustic models. Usually, one three-state "linear" HMM is created for every phoneme. The main idea of the tree-state structure is that the first state models the influence from the previous phoneme to the current one. The second state models the "stable" part of a phoneme and the third state models the dependency between the current phoneme and the next one. Naturally, in this case, the alignment of the acoustic observation x_1^T with the HMM only makes sense in a "forward direction". The HMM thus is "linear" in the sense that there are only non-zero transition probabilities between successive states in a a forward direction.

The HMM of a particular phoneme is used to calculate the likelihood that that particular phoneme is uttered based on the acoustic evidence

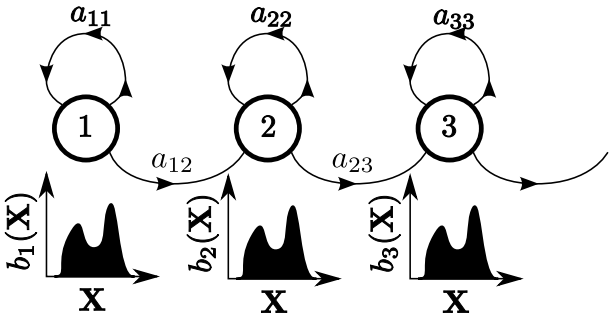


Figure 1.3: A typical HMM used in speech recognition

(series of feature vectors) $\mathbf{x}_1^T$. Such a three-state HMM is represented in fig. 1.3. The transitions with zero probability are omitted.

In order to fully apply these HMM's for speech recognition, three problems have to be solved:

- How to evaluate an HMM? In the context of ASR evaluation means calculating the probability of observing acoustic evidence $\mathbf{x_1^T}$ given a specific HMM. This is called the *evaluation problem*
- How to find the most likely state sequence? Here we are interested in finding the most likely path $\mathbf{s}_1^T$ through an HMM based on the acoustic evidence $\mathbf{x}_1^T$. This is called the *decoding problem*.
- How to find the optimal parameter values of the HMM? The HMM has an observation distribution for every state and transition probabilities between states. These parameters have to be estimated based on available labeled acoustic evidence. This is the *training problem*.

These three problems are discussed in the next few paragraphs.

1.4.2 Evaluating Hidden Markov Models

In the evaluation problem, we are interested in the probability that the acoustic observation sequence $\mathbf{x}$ is observed given the HMM Φ: $p(\mathbf{x}|\Phi)$.

The most straightforward method to compute this is to sum up all possible state sequences $\mathbf{s}$:

$$p(\mathbf{x}|\Phi) = \sum_{\mathbf{s}} P(\mathbf{s}|\Phi)p(\mathbf{x}|\mathbf{s},\Phi), \tag{1.10}$$

where

$$P(\mathbf{s}|\Phi) = P(s_1|\Phi)\prod_{t=2}^{T} P(s_t|s_{t-1},\Phi) = \pi_{s_1}a_{s_1s_2}\ldots a_{s_{T-1}s_T} \tag{1.11}$$

and

$$\begin{aligned} p(\mathbf{x}|\mathbf{s},\Phi) = p(\mathbf{x}_1^T|s_1^T,\Phi) &= \prod_{i=1}^{T} p(\mathbf{x}_t|s_t,\Phi) \\ &= b_{s_1}(\mathbf{x}_1)b_{s_2}(\mathbf{x}_2)\ldots b_{s_T}(\mathbf{x}_T). \end{aligned} \tag{1.12}$$

Combining equations (1.10), (1.11) and (1.12) leads to the following HMM probability:

$$p(\mathbf{x}|\Phi) = \sum_{\mathbf{s}} \pi_{s_1}b_{s_1}(\mathbf{x}_1)a_{s_1s_2}b_{s_2}(\mathbf{x}_2)\ldots a_{s_{T-1}s_T}b_{s_T}(\mathbf{x}_T). \tag{1.13}$$

It is easy to imagine that a brute force evaluation of this equation has an exponential time complexity in T: $\mathcal{O}(N_S^T)$. Therefore, it is not preferable to calculate it in this way. However, the first order Markov assumption allows for a more efficient calculation of the HMM likelihood based on a dynamic programming approach: the forward algorithm [41]. The basic principle of this forward algorithm is the fact that intermediate results can be stored and reused for subsequent state-sequence calculations. The basic step of the algorithm is the computation of the forward probability

$$\alpha_t(i) = p(\mathbf{x}_1^t, s_t = i|\Phi). \tag{1.14}$$

The forward probability is the probability of being in state i at time step t. This can be computed by induction over t because it is a weighted sum of probabilities of arriving in one of the possible previous states at time

1: **for** $i = 1$ to N_S **do**
2: $\quad \alpha_1(i) = \pi_i b_i(\mathbf{x}_1)$
3: **end for**
4: **for** $t = 2$ to T **do**
5: $\quad$ **for** $j = 1$ to N_S **do**
6: $\quad\quad \alpha_t(j) = b_j(\mathbf{x}_t) \sum_{i=1}^{N_S} \alpha_{t-1}(i) a_{ij}$
7: $\quad$ **end for**
8: **end for**
9: $p(\mathbf{x}|\Phi) = \sum_{i=1}^{N} \alpha_T(i)$

Figure 1.4: The forward algorithm [41]

step $t-1$ multiplied by the observation probability of the t-th observation in the current state, i.e.

$$\alpha_t(j) = b_j(\mathbf{x}_t) \sum_{i=1}^{N_S} \alpha_{t-1}(i) a_{ij}. \tag{1.15}$$

The main algorithm is shown in fig. 1.4. Obviously, this forward algorithm has a much better time complexity: $\mathcal{O}(N_S^2 T)$

The forward algorithm can be perceived as the construction of a trellis. A trellis is a graph structure that is ordered in vertical slices (time steps) and where every node is connected to at least one node of the previous time steps and to at least one node of a later time step. In the forward algorithm trellis, each row corresponds to a state i and each column corresponds to a time step t. In speech recognition, this trellis has a very simple structure, because the conventional HMMs used in speech recognition only permit two possibilities: remaining in the same state or jumping to the next state, as is displayed in fig. 1.3. An example of such a simple trellis is shown in fig. 1.5.

1.4.3 Decoding Hidden Markov Models

Using the forward algorithm, it is possible to calculate the probability by summing over all possible paths. It is not possible however to trace the

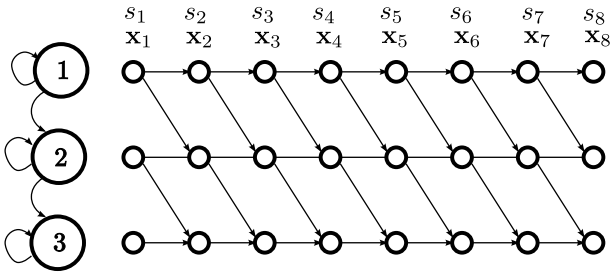


Figure 1.5: Trellis for the forward algorithm

most likely state sequence through the HMM, which is the problem at hand in the decoding problem. For this purpose, the Viterbi algorithm has been designed [89] and used for speech recognition. It is still widely used.

The structure of the Viterbi algorithm is highly similar to the forward algorithm. However, instead of taking a sum over all states, a maximum over all states is taken. At the end, this results in the probability of the most likely state sequence. In order to be able to reconstruct this state sequence, back pointers are put into place. The algorithm is shown in fig. 1.6

Note that this algorithm is also useful for computing an estimation of the HMM score. Instead of all possible paths, only the most likely path is used for computing the likelihood of an HMM given an observation. The Viterbi algorithm is widely used for evaluation of HMM's because the maximum operator can be implemented more efficiently than the sum operation of the forward algorithm.

1.4.4 Training Hidden Markov Models

The third problem is the task of estimating the parameter values for the HMM. These parameters depend on the structure of the HMM and on the type of observation distribution that is used. In speech recognition, these observation distributions are often represented by a Gaussian Mixture Model (GMM), which consists of a weighted sum of different Gaus-

1: **for** $i = 1$ to N **do**
2: $V_1(i) = \pi_i b_i(\mathbf{x}_1)$
3: $B_1(i) = 0$
4: **end for**
5: **for** $i = 2$ to T **do**
6: **for** $j = 1$ to N **do**
7: $V_t(j) = b_j(\mathbf{x}_t) \max_{1 \leq i \leq N_S} V_{t-1}(i) a_{ij}$
8: $B_t(j) = \arg\max_{1 \leq i \leq N_S} V_{t-1}(i) a_{ij}$
9: **end for**
10: **end for**
11: $V_T = \max_{1 \leq i \leq N_S} V_T(i)$
12: $S_T = \arg\max_{1 \leq i \leq N_S} B_T(i)$

Figure 1.6: The Viterbi algorithm [89]

sian distributions. The parameter set of an HMM will be denoted by $\Phi = (\mathbf{A}, \mathbf{B}, \pi)$, with $\mathbf{A}$ the matrix of transition probabilities, $\mathbf{B}$ the set of output probability density functions and π the initial state distribution. The goal of the training problem is to find the parameter set $\hat{\Phi}$ that maximized the log-likelihood of the observation under the model parameters:

$$\begin{aligned} \hat{\Phi} &= \arg\max_{\Phi} \log p_{\Phi}(\mathbf{x}|\mathbf{s}) \\ &= \arg\max_{\Phi} \log p(\mathbf{x}|\mathbf{w}, \Phi), \end{aligned} \tag{1.16}$$

where $\mathbf{w}$ denotes a series of words. Word models are formed by concatenating the individual phoneme HMM's using the dictionary.

To train the parameters, a set of example data must be available, transcribed with the corresponding words uttered in the data and a dictionary containing the sequence(s) of phonemes for each word has to be available. Training the HMM parameters is an unsupervised learning problem, since the state sequence in HMM's is hidden and thus unavailable for training.

Several algorithms were designed for training these HMM parameters. Usually one distinguishes between maximum likelihood training (ML) and discriminative training.

Maximum likelihood training

Maximum likelihood training is based on the principle of expectation maximization (EM). EM is an iterative procedure that repeatedly computes the expected value of the log-likelihood of the data and maximizes it over the model parameters. In this way, a series of HMM parameters sets $\Phi_0, \Phi_1, \Phi_2, \ldots, \Phi_N$ is generated where the log-likelihood of Φ_i is larger than that of Φ_{i-1}. Mathematically, this procedure can be written as follows. Suppose Φ_i is the current estimate and Φ_{i+1} is the one to compute. The log-likelihood of $p(\mathbf{x}|\Phi_{i+1})$ can be rewritten as follows:

$$\log p(\mathbf{x}|\Phi_{i+1}) = \log p(\mathbf{x}, \mathbf{s}|\Phi_{i+1}) - \log P(\mathbf{s}|\mathbf{x}, \Phi_{i+1}). \tag{1.17}$$

If S denotes the stochastic variable of the state sequence, taking the conditional expectation of $\log p(\mathbf{x}|\Phi_{i+1})$ over S with parameter vector Φ_i, we obtain the following:

$$E_{\Phi_i}\left[\log p(\mathbf{x}|\Phi_{i+1})\right]_{S|\mathbf{x}} = \sum_{\mathbf{s}} \left(P(\mathbf{s}|\mathbf{x}, \Phi_i) \log p(\mathbf{x}|\Phi_{i+1})\right) = \log p(\mathbf{x}|\Phi_{i+1}) \tag{1.18}$$

By combining equations (1.17) and (1.18), $\log p(\mathbf{x}|\Phi_{i+1})$ can be rewritten as follows:

$$\begin{aligned} \log p(\mathbf{x}|\Phi_{i+1}) &= E_{\Phi}\left[\log P(S, \mathbf{x}|\Phi_{i+1}\right]_{S|\mathbf{x}} \\ &\quad - E_{\Phi}\left[\log P(S|\mathbf{x}, \Phi_{i+1})\right]_{S|\mathbf{x}} \\ &= \sum_{\mathbf{s}} P(\mathbf{s}|\mathbf{x}, \Phi_i) \log p(\mathbf{s}, \mathbf{x}|\Phi_{i+1}) \\ &\quad - \sum_{\mathbf{s}} P(\mathbf{s}|\mathbf{x}, \Phi_i) \log P(\mathbf{s}|\mathbf{x}, \Phi_{i+1}) \\ &= Q(\Phi_i, \Phi i + 1) - H(\Phi_i, \Phi_{i+1}). \end{aligned} \tag{1.19}$$

It can be shown that if $Q(\Phi_i, \Phi_{i+1}) > Q(\Phi_i, \Phi_i)$, then $\log p(\mathbf{x}|\Phi_{i+1}) > \log p(\mathbf{x}|\Phi_i)$. The EM algorithm then iteratively computes the next parameter vector by maximizing this Q-function over the current parameter vector:

$$\Phi_{i+1} = \arg\max_{\Phi} \sum_{\mathbf{s}} P(\mathbf{s}|\mathbf{x}, \Phi_i) \log p(\mathbf{s}, \mathbf{x}|\Phi). \tag{1.20}$$

It can be shown that the Q-function can be written as a sum of two components: one in $\mathbf{A}$ and one in $\mathbf{B}$. Since $\mathbf{A}$ and $\mathbf{B}$ are subject to constraints as presented in equations (1.5) and (1.6), maximizing this expected value is a constraint optimization of those two components. The optimization can for instance be performed using the technique of Lagrange multipliers. For a more in-depth review of EM-training in speech recognition, the reader is referred to Huang et al. [37].

An alternative to the EM-algorithm is *Viterbi-training*, in which the sum over all possible state sequences is replaced by a maximum operator

$$\hat{\mathbf{s}} = \arg\max_{\mathbf{s}} \log p(\mathbf{x}, \mathbf{s}|\mathbf{w}, \Phi_{i-1}). \tag{1.21}$$

The state sequence $\hat{\mathbf{s}}$ is used to estimate the next model parameters:

$$\Phi_i = \arg\max_{\Phi} \log p(\mathbf{x}, \hat{\mathbf{s}}|\mathbf{w}, \Phi). \tag{1.22}$$

Discriminative training

Maximum Likelihood suffers from the problem that it only works optimal when the underlying model is correct and the amount of available training data is sufficiently high. In speech recognition however, both assumptions are false: the common GMM model clearly is a simplification and the amount of available training data is not always as high as necessary.

ML training only takes into account data for the correct "class". I.e. it only uses examples of the phoneme "a" to train the parameters of the HMM model representing that "a" without worrying about the scores of the other HMM models. In other words, ML training tries to train the evaluation score of the correct HMM as high as possible, it does not try to train the evaluation scores of the other incorrect HMM's as low as possible.

To overcome this, discriminative training techniques were proposed. Discriminative training techniques try to overcome the limitations of ML training by estimating the parameters of the correct and incorrect models simultaneously. A wide variety of these techniques have been proposed in speech and pattern recognition. The most common discriminative training

procedure are Maximum Mutual Information (MMI), Minimum Classification Error (MCE), and Minimum Phone Error (MPE). These discriminative training procedures are one of the main topics of this thesis and are extensively introduced in §5.1

1.5 Language Modeling

The second large component of mainstream ASR systems is the language model (LM). The language model defines how individual words can be combined into sentences. Language models are not studied in this research but for completeness, a brief introduction to stochastic N-grams, the type of language models used in this thesis, is presented in the next paragraphs. For more information about language modeling, we refer to Huang et al. [37]

Stochastic N-grams

Stochastic N-grams represent the likelihood of a given word sequence $\mathbf{w} = w_1, w_2, \ldots, w_n$. This likelihood is modelled by $P(\mathbf{w})$ that this word sequence is uttered. The probability $P(\mathbf{w})$ can be rewritten as follows

$$\begin{aligned} P(\mathbf{w}) &= P(w_1, \ldots, w_n) \\ &= P(w_1)P(w_2|w_1)\ldots P(w_n|w_1, w_2, \ldots, w_{n-1}) \\ &= \prod_{i=1}^{n} P(w_i|w_1, w_2, \ldots, w_{i-1}). \end{aligned} \tag{1.23}$$

However, evaluating this formula would mean that the probability of the next word has to be evaluated for every possible word sequence $w_1, w_2, \ldots, w_{i-1}$. This is nearly impossible because even for moderate i the number of combinations is huge and the number of occurrences in the available data is low (if any). Therefore, the concept of N-grams was introduced. In N-grams, the probability of the occurrence of word w_i is only dependent on the previous N words, i.e. $w_{i-N+1}, w_{i-N+2}, \ldots, w_{i-1}$. Typical values for N are 1 (unigram), 2 (bigram) or 3 (trigram). One special case occurs

when $N = 2$ and $P(w_i|w_{i-1})$ is a 0-1 function: 1 if the sequence is possible and 0 otherwise. This type of LM is called a word pair grammar.

1.6 Motivation

In this section, the motivation for the template based speech recognition approach studied in this research is explained. First, the shortcomings of conventional HMM models are discussed. Next to these shortcomings, some other motivations for exploring template based speech recognition are presented.

1.6.1 Shortcomings of HMM-based Speech Recognition

HMM-based speech recognition systems have proven to be very fast and efficient in decoding spoken language, as they severely limit the search space. They consist of only one model per acoustic unit instead of hundreds of examples of these acoustic units. However, these models also make some assumptions about speech recognition that are not entirely correct. These assumptions will be discussed in the next paragraphs.

Duration modeling

One of the weaknesses of traditional HMM's is the lack of accurate duration modelling within states. When evaluating an HMM, the probability that t consecutive observations are in the same state i looks as follows:

$$d_i(t) = a_{ii}^t(1 - a_{ii}). \tag{1.24}$$

As shown in this formula, the probability decreases exponentially with time. The probability of staying t consecutive frames in state i thus has exponential distribution. It also means that given a transition probability a_{ii}, there is a fixed optimal value of the duration of that state and that both longer or shorter durations are less likely. It has been argued that such an exponential distribution is not very useful to represent linguistically useful components [50]. I.e. there is no reason why an "E" which lasts 10 frames should be more or less likely than an "E" which lasts 13 frames.

There is an intrinsic variation in the duration of phonemes in real speech. This is poorly modelled by an HMM.

Several attempts have been made to overcome this limitation of classical HMM's. Levinson [50] for instance introduced the concept of *state duration models*. In these models, the self-loop is replaced by a duration probability distribution. If the exponential state occupancy distribution from equation (1.24) is inserted into these models, a classical HMM is obtained. These duration models however did not result in significant improvements of the error rates for speaker-independent speech recognition.

The Markov assumption

The first-order Markov assumption was introduced because of the easiness in computing the model parameters.

Mari et al. [55] introduced the use of second order HMM's to make the model more realistic. In this case, the evaluation of a state sequence is computed as follows:

$$P(\mathbf{s}) = \prod_t a_{s_{t-2}s_{t-1}s_t}. \tag{1.25}$$

This extension however suffers from an increased number of states resulting in more expensive calculations. This enlargement of the state space however did not result in improved performance of speech recognition.

The conditional independence assumption

Another assumption made by the HMM is the conditional independence assumption shown in equation (1.9), i.e. the assumption that speech frames are only dependent on the state that generated them and not on their neighboring frames, i.e. the frames are independent of each other. These frames however are often very correlated such that this independence assumption is invalid. Several attempts have been made to include these dependencies in the models. Ming et al. for instance proposed to make the probability distribution also dependent on the previous frame [64]. In

this case, the probability of state sequence $\mathbf{s}$ looks as follows:

$$p(\mathbf{x}|\mathbf{s}, \Phi) = \prod_{t=1}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}, s_t, \Phi). \tag{1.26}$$

Of course, this expands the parameter space by a huge factor. Techniques such as vector quantization have been used to keep the parameter space size under control. Efficient parameter training algorithms for these conditional dependent HMM's have been created based on a small adaption of the forward-backward algorithm.

1.6.2 Availability of computing power

Prior to 1980, non probabilistic speech recognition based on dynamic time warping (DTW)[1] was the main approach to perform speech recognition. However, using this approach for large-sized (and even medium-sized) vocabulary continuous speech recognition was not possible, due to the limited computational resources available at that time, both in processing power and available memory.

The lack of available computing power was one of the motivations for the transition to HMM-based speech recognition, which occurred in the early 80's.

However, the amounts of available processing and memory capacity have increased exponentially over the past decades, as stated by Moore's Law, which is applicable since 1970[2]:

> **Moore's Law**: The number of transistors on an integrated circuit doubles every twenty months.

Moore's law is very general and applies to the available RAM memory, hard disk capacity as well as to the computing performance of processors.

According to this law, the amount of computing capacity has increased by a factor of 2^{18} during the last three decades. The general availability of supercomputing, cluster computing and the more decentralized grid

[1] DTW is explained in § 2.3.

[2] Before 1970, the formulation mentioned a doubling every year.

computing has boosted research, with lots of important scientific results in different research fields as a result, with topics as the decoding of the genome, medical research and very recently, the inauguration of the large hadron collider at CERN in Geneva. All these applications require vast amounts of computing capacity and would not have been possible without the current state of the art in computing devices.

This evolution, together with the disadvantages of HMM-based speech recognition has inspired us to continue research on the abandoned track of DTW-based speech recognition.

1.7 Template based speech recognition

In contrast to HMM based speech recognition, template based speech recognition (TBSR) makes no use of statistical models. Instead of replacing each acoustic unit with an appropriate HMM model, the "model" is the preprocessed data itself. I.e. instead of using HMM's as acoustic models, new speech is directly compared with available examples of these acoustic units (phonemes). Such examples of acoustic units are called "templates". These templates thus are represented by a sequence of feature vectors generated by the preprocessing unit.

So, instead of only a single model for each acoustic unit (usually a context-dependent phoneme), template based speech recognition works with every available example of this acoustic unit directly. The basic algorithm to perform the comparison between input and example data is the dynamic time warping algorithm (DTW), explained in the next chapter. Template based speech recognition is a form of DTW-based speech recognition. Of course, this method increases the branching factor of the search space with a factor equal to the number of available examples for each unit.

Even with the current advances in computing power, this data driven approach is not feasible for medium and large vocabulary speech recognition problems if the search space is traversed in a brute force way. Instead, some algorithms will have to be devised to cope with this search space. For this purpose, De Wachter et al. constructed a template selection strategy which performs a preselection of templates before performing the main

search. This preselection is given the name "time filter" [22]. Details about the template preselection is presented in chapter 2.

This template based approach overcomes several of the disadvantages of HMM models. The modeling of duration of templates is implicit in the number of feature vectors in the sequence, whereas modeling of duration in a three-state HMM model is a lot more complex. Also, the independence assumptions of HMM are not really present any more, since these feature vectors are by nature dependent on the previous observations and as a result, the dependence is present in the data itself.

In the previous paragraphs, template based speech recognition was only briefly explained and motivated. A more detailed discussion of TBSR is presented in chapter 2.

1.8 Relation to other work

Though the oldest recognition systems date as far back as the fifties [18], it was only in the seventies, with the introduction of time warping and the DTW algorithm that automatic speech recognition became popular [35].

DTW-based speech recognition is a very old approach to speech recognition. The technique has been used for speech recognition since the seventies by (amongst others) Sakoe and Chiba [84] and Itakura et al. [38]. Initial experiments on speech recognition were restricted to spoken digit recognition and isolated word recognition because the necessary computing power to perform continuous speech recognition was unavailable at that time.

This limitation in availability of computing power has forced researchers to look for other strategies that need less computing power. HMM-based speech recognition was one of those proposed approaches leading to acceptable recognition results whilst reducing the complexity. The interest in DTW-based speech recognition has diminished during this period and until today, HMM-based speech recognition still is the mainstream approach for speech processing.

Recently, interest in DTW-based speech recognition revived. Aradilla et. al used a DTW-based recognition system for number recognition [2]. Ramasubramanian et al. proposed a template based recognition system

much similar to the system presented in this dissertation [81]. Axelrod et al. combined HMMs and nearest neighbor classification using model selection [8].

The application of DTW is not restricted to speech recognition. It has been applied to many different subfields of speech processing including speaker recognition and keyword spotting. In recent work, Aronowitz et al. performed word spotting using the DTW algorithm to create a speaker recognition system [3]. Monsieur et al. illustrated the usefulness of a DTW system for keyword spotting [65].

The template based approach to DTW-based speech recognition itself is also related to work by other people. A template based approach is very common in speech synthesis, more specifically concatenative speech synthesis. In this type of speech synthesis, speech is generated by concatenating and smoothing a set of samples available in a database. These examples are acoustic units derived from speech generated by a single speaker. This type of speech synthesis shows a high similarity with template based speech recognition, where input speech is recognized by searching for matching units in the database and concatenating them into words and sentences.

1.9 Outline

The remainder of this text is structured as follows. In chapter 2, template based speech recognition (TBSR) is extensively introduced and the TBSR system developed and used for our research is presented.

In our research, we were mainly interested in defining distance measures and applying them to template based speech recognition. These distance measures are the main topic of this dissertation and are discussed in chapters 3, 4 and 5.

In chapter 3, we discuss the concept of distance measures with extensions on the Euclidean distance measure as an approach to model this space more accurately. More specific, class dependent distance measures are introduced. Our main contributions described in chapter 3 are the introduction of class dependent distance measures for dynamic time warping and the use of block-diagonal weight matrices for scaling these distance

measures.

Next, chapter 4 discusses methods to partition the data into classes for these class-dependent distance measures. For our research, we have employed a broad spectrum of existing techniques and combined them to partition the training data and defining the classes of the distance measure. The chapter explains these techniques and shows how we applied them for our purposes.

The final chapter on distance measures, chapter 5, is concerned with procedures to train the weights of the distance measures. We explain the concept of classification and parameter estimation and introduce the different approaches based on maximum likelihood and discriminative training objectives. We also explain non-parametric density estimation. Finally, we have combined these different techniques in order to train the parameters of the distance measures in § 5.6 and § 5.7. This is an important contribution of our research.

Next, in chapter 6, we investigate the problem of k-nearest neighbor searches through this acoustic space. We have examined a variety of existing techniques for k-nearest neighbor searches and have made a comparison between them.

In chapter 7, experiments on template based speech recognition and on pattern classification are discussed. These experiments combine all the methods presented in the previous chapters. Finally, a conclusion summing up the most important results of this research is stated in chapter 8.

Chapter 2

Template based speech recognition

In Template Based Speech Recognition (TBSR), the acoustic models are absent. Acoustic observations are not used to train probabilistic acoustic models. Instead, every acoustic unit is stored separately in a training database. Speech recognition occurs by selecting a template string matching with the input.

This chapter is organized as follows: in § 2.1, we define the concept of templates. Section 2.2 presents a statistical view of TBSR, showing some analogy with HMM based speech recognition. Section 2.3 describes the basic algorithm for non parametric speech recognition: dynamic time warping. Finally, the architecture of the basic template based speech recognition software is described in § 2.4.

2.1 The basic unit: templates

De Wachter [20] defines a template as follows:

Definition 1 *A template is the representation of an actual segment of speech. It consists of*

- *a sequence of consecutive acoustic feature vectors (or frames),*

- *a transcription of the sounds or words it represents (typically one or more phonetic symbols),*
- *knowledge of neighboring templates (consisting of a template number if templates do not overlap), and*
- *a tag with meta-information.*

Meta-information is defined as follows:

Definition 2 *Meta-information or non-verbal information is any information we have apart from acoustic realization and transcription of a segment and its context.*

These definitions leave us with a lot of choice in defining the real templates. Some straightforward choices are the following basic units:

- **Words**: the use of a word as template seems a logical choice. Recognition occurs by matching example words with the input speech. The disadvantage however is the available amount of template data. In large vocabulary speech recognition, a lot of words do not occur frequently in the training data, thus limiting the number of templates for that word severely. Moreover, a template based speech recognizer using word templates will never be able to recognize out-of-vocabulary words, since no data is available for these words.
- **Context-dependent phonemes**: context-dependent (CD) phonemes are phonemes where the phoneme label is not only dependent on the phoneme itself, but also on the context: the preceding and succeeding phoneme. Using CD phonemes as templates solves the problem of out-of-vocabulary words since these can be reconstructed by concatenating the right phonemes. However, this does not solve the sparsity of the available training data. Also in this case, some templates will be few in number.
- **Context-independent phonemes**: the use of context-independent phonemes solves both problems. Each phoneme is available in high numbers, and out-of-vocabulary words can be reconstructed.

Thus, in choosing the template unit size, a trade-off between accuracy and available training data has to be made. Moreover, the smaller the template unit size, the larger the search space will be. Unless otherwise mentioned, we will assume that templates are context-independent phonemes in the remainder of this dissertation.

2.2 TBSR: a statistical approach

In § 1.2.2, we presented the basic speech recognition equation:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} p(\mathbf{x}|\mathbf{w})P(\mathbf{w}). \tag{2.1}$$

However, in template based speech recognition, there is a third stream of information: the template sequence. A word sequence is no longer uniquely defined by a sequence of states as in HMM based speech recognition, where there is only one single model for each (context-dependent) phoneme. In TBSR however, a word sequence can be defined by many different template sequences, because there are many templates in the database that correspond to the same phoneme. Therefore, equation (2.1) has to be adapted:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} \sum_{\mathbf{t}} p(\mathbf{x}|\mathbf{w},\mathbf{t})P(\mathbf{t}|\mathbf{w})P(\mathbf{w}), \tag{2.2}$$

where $p(\mathbf{x}|\mathbf{w},\mathbf{t})$ represents the acoustic likelihood of the input given template string $\mathbf{t}$ and word string $\mathbf{w}$, which is equal to $p(\mathbf{x}|\mathbf{t})$ since $\mathbf{w}$ is conditionally independent of $\mathbf{x}$ given $\mathbf{t}$ (pronunciation variants of words not taken into account). This factor $p(\mathbf{x}|\mathbf{t})$ is an acoustic score (likelihood) for the input $\mathbf{x}$. Since $\mathbf{x}$ and $\mathbf{t}$ are both represented by sequences of speech frames a way to compare two speech fragments is necessary (analogous to comparing a speech segment with an HMM model in HMM-based SR). The most obvious way to do this is by using the dynamic time warping algorithm, presented in § 2.3.

$P(\mathbf{t}|\mathbf{w})$ is a factor describing the prior probability of the template string $\mathbf{t}$ given a word transcription $\mathbf{w}$. This term guarantees that sentences containing phonemes for which only few examples are present in

the database are not less likely than sentences with phonemes that occur very often in the template database. This is caused by taking the sum over all possible template sequences $\mathbf{t}$.

Analogous to the approach in HMM-based speech recognition, we can take the Viterbi approach. I.e. the sum over all template sequences is replaced by a maximum operator as follows:

$$\hat{w} = \arg \max_{\mathbf{w}} \max_{\mathbf{t}} p(\mathbf{x}|\mathbf{w}, \mathbf{t}) P(\mathbf{t}|\mathbf{w}) P(\mathbf{w}). \tag{2.3}$$

With this Viterbi approximation, the normalization factor $P(\mathbf{t}|\mathbf{w})$ is no longer necessary and can be dropped.

The factor $P(\mathbf{w})$ is a language model score, analogous to the language model scores used in HMM-based speech recognition, briefly described in § 1.5. We only need to define how to compute the "likelihood" of an input sequence $\mathbf{x}$ given a template and word sequence $\mathbf{t}$ and $\mathbf{w}$. In order to do this, we need a method to compare two sequences of speech frames. This comparison can be performed by using the dynamic time warping algorithm, which is discussed in the next section.

2.3 Dynamic Time Warping

As explained in chapter 1, HMM-based speech recognition uses the Viterbi-algorithm to compare new speech with acoustic models. In template based speech recognition these acoustic models are replaced by templates. Therefore, an algorithm to compare two segments of speech is required. This algorithm has to cope with the difficulty of pronunciation variations in the segment. These variants usually do not have the same length, because the length of an utterance (phoneme, word, sentence) is different for each utterance and depends on the context, the speaker, the mood of the speaker etc...

To cope with this problem, a dynamic programming approach was originally proposed by Sakoe and Chiba in 1971 [84]. Many variations have been described in subsequent research [66, 38]. The basics of this dynamic programming approach or "dynamic time warping" algorithm are described below. For more information, the reader is referred to Rabiner et al. [80].

2.3.1 Distance between input and template

Define an input speech segment $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_{T_1})$ of length T_1 and a template $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_{T_2})$ of length T_2. Both $\mathbf{x}$ and $\mathbf{y}$ are segments of speech consisting of a series of observations (feature vectors) $\mathbf{x}_i, i = 1, \ldots, T_1$ and $\mathbf{y}_j, y = 1, \ldots, T_2$. For simplicity of notation, we will also denote a distance $d(\mathbf{x}_i, \mathbf{y}_j)$ as $d(i, j)$ during this section. Usually, a squared distance is used for computing distances between frames in DTW, therefore we will explicitly denote the square in the formulas.

The distance between these segments can be computed by searching for the lowest accumulated distance on a path through the local distance matrix as illustrated in fig. 2.1. The elements of the matrix represented in this figure contain the distances between individual feature vectors. The distance between $\mathbf{x}$ and $\mathbf{y}$ is the sum of the distances encountered on a path which starts in $(\mathbf{x}_i, \mathbf{y}_1)$ and ends in $(\mathbf{x}_{T_1}, \mathbf{y}_{T_2})$. This path is represented by the arrows that are drawn between the matrix elements.

Formally, this accumulated distance concept can be formulated as follows:

$$D_\phi(\mathbf{x}; \mathbf{y}) = \sum_{k=1}^{T} d^2(\phi_x(k), \phi_y(k)) m(k) / M_\phi, \tag{2.4}$$

where $\phi_x(k)$ and $\phi_y(k)$ are warping functions resembling the position in segments $\mathbf{x}$ and $\mathbf{y}$ respectively at step k in the distance calculation, $m(k)$ is a weighting coefficient and M_ϕ is a normalization factor and T is the total number of steps in the alignment. The most obvious choice of warping functions ϕ_x and ϕ_y is the one that leads to the lowest accumulated distance. Therefore the distance between $\mathbf{x}$ and $\mathbf{y}$ is defined as follows:

$$D(\mathbf{x}; \mathbf{y}) = \min_{\phi} D_\phi(\mathbf{x}, \mathbf{y}). \tag{2.5}$$

The distance defined in equation (2.5) is also called the dynamic time warping distance or DTW score.

Several local path constraints apply to this equation in order to be a valid distance measure:

- **Endpoint constraints**: the path through the distance matrix should start at the first frame and end at the last frame of both input and

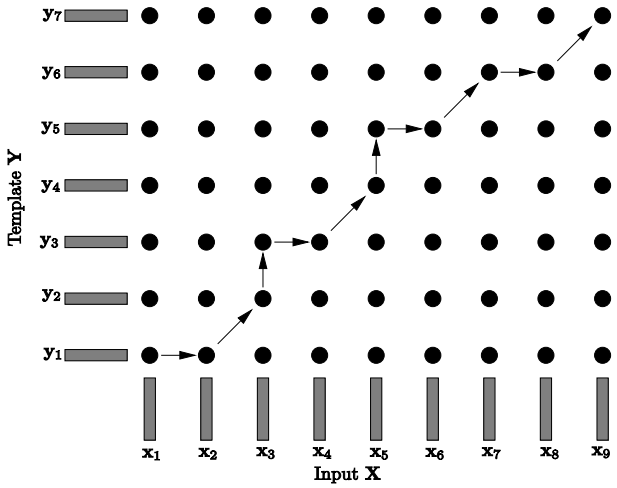


Figure 2.1: Template alignment: time warping

template:

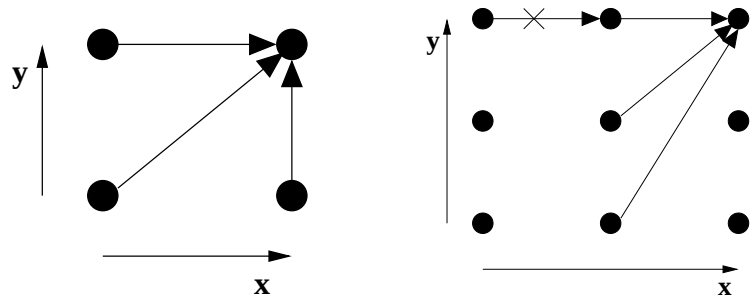
$$\phi_x(1) = 1, \phi_x(T) = T_1, \tag{2.6}$$
$$\phi_y(1) = 1, \phi_y(T) = T_2. \tag{2.7}$$

- **Monotonicity**: the path should only move forward through the local distance matrix:

$$\phi_x(k+1) \geq \phi_x(k), \tag{2.8}$$
$$\phi_y(k+1) \geq \phi_y(k). \tag{2.9}$$

- **Local continuity**: important information in template and input should not be skipped. Therefore it should be impossible to skip several frames in a row. Several variants have been proposed for these constraints, among others, Sakoe and Chiba [84] proposed:

$$\phi_x(k+1) - \phi_x(k) \leq 1, \tag{2.10}$$
$$\phi_y(k+1) - \phi_y(k) \leq 1. \tag{2.11}$$



(a) Sakoe (b) Itakura

Figure 2.2: DTW local path constraints: Sakoe and Itakura

Itakura proposed a different variant [38]:

$$\phi_x(k) - \phi_x(k-1) = 1, \tag{2.12}$$

$$0 \leq \phi_y(k) - \phi_y(k-1) \leq 2, \tag{2.13}$$

$$\phi_y(k-2) < \phi_y(k). \tag{2.14}$$

An overview of the differences between the Sakoe and Itakura local constraints is shown in fig. 2.2. The cross shown in the Itakura case indicates an impossible path, imposed by equation (2.14).

Together, these constraints lead to global path constraints. I.e. the path through the distance matrix has to lie entirely withing a specific region. In the case of Itakura-constraints, this region is shaped as a parallelogram, as illustrated in fig. 2.3.

2.3.2 Distance calculation: Dynamic Time Warping

The most harsh way to compute the DTW score is to compute the DTW distance for every possible warping function ϕ and taking the minimum. However, there exist more efficient methods to compute the distance using a dynamic programming approach, as proposed by Sakoe et al. [84]. In dynamic programming, properties of overlapping subproblems are exploited. A problem has overlapping subproblems if the problem can be divided into smaller subproblems that are reused several times. In the case of distance

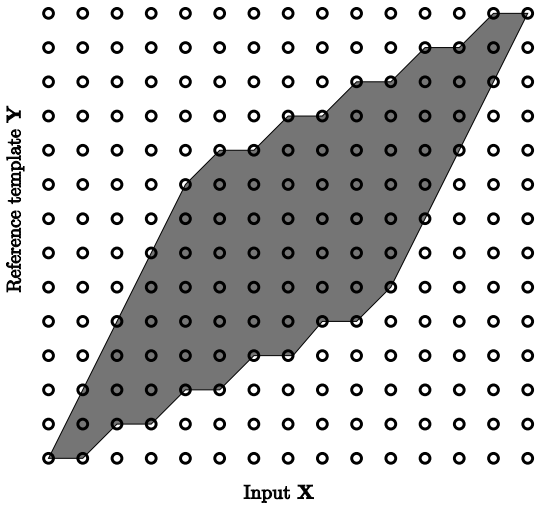


Figure 2.3: DTW: global path constraints in the Itakura case

calculation between speech segments, these overlapping subproblems are obvious: if $P_{i,j}$ denotes the partial optimal path to position (i, j) in the local distance matrix, and (i, j) is in the global optimal path, denoted P, then $P_{i,j}$ is also a part of P. If this observation is taken into account, the distance problem can easily be written in a recursive formulation.

For the Sakoe and Chiba constraints, this recursion is defined as follows:

$$D(\mathbf{x}_1^{\phi_x(t)}; \mathbf{y}_1^{\phi_y(t)}) = \min \left\{ \begin{array}{l} D(\mathbf{x}_1^{\phi_x(t)-1}; \mathbf{y}_1^{\phi_y(t)}) + d^2(\mathbf{x}_{\phi_x(t)}; \mathbf{y}_{\phi_x(t)})\gamma_0 \\ D(\mathbf{x}_1^{\phi_x(t)-1}; \mathbf{y}_1^{\phi_y(t)-1}) + d^2(\mathbf{x}_{\phi_x(t)}; \mathbf{y}_{\phi_y(t)})\gamma_1 \\ D(\mathbf{x}_1^{\phi_x(t)}; \mathbf{y}_1^{p(t)-1}) + d^2(\mathbf{x}_t; \mathbf{y}_{p(t)})\gamma_2 \end{array} \right\}, \tag{2.15}$$

with initial condition:

$$D(\mathbf{x}_1^1; \mathbf{y}_1^1) = d^2(\mathbf{x}_1, \mathbf{y}_1). \tag{2.16}$$

The weighting factors γ_0, γ_1 and γ_2 allow to weight the path options differently.

For the Itakura constraints, the recursion can be written as follows:

$$D(\mathbf{x}_1^{\phi_x(t)};\mathbf{y}_1^{\phi_y(t)}) = \min \left\{ \begin{array}{l} D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_0 \\ D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)-1}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_1 \\ D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)-2}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_2 \end{array} \right\}, \tag{2.17}$$

with starting conditions:

$$D(\mathbf{x}_1^1;\mathbf{y}_1^1) = d^2(\mathbf{x}_1,\mathbf{y}_1) \tag{2.18}$$

$$D(\mathbf{x}_1^1;\mathbf{y}_1^2) = d^2(\mathbf{x}_1,\mathbf{y}_2)\gamma_2 \tag{2.19}$$

$$D(\mathbf{x}_1^1;\mathbf{y}_1^i) = +\infty, i > 2. \tag{2.20}$$

This dynamic programming approach has been called Dynamic Time Warping.

Using DTW, we can compute the distance between an input sequence and a template. This assumes that the input sequence matches the template completely. In continuous speech recognition however, the templates are usually words or sub-word units (phonemes) and the input is an entire sentence. Therefore, the input should not be aligned to one single template but to all possible template sequences $\mathbf{t}$. This approach is further explained in § 2.4.3.

2.3.3 DTW and templates

In this framework, we can represent a template as a sequence of feature vectors. Depending on the specific local path constraints of the chosen DTW variant, the possible connections between these feature vectors are a stall, a forward step, a skip and possibly others. In the case of DTW with Itakura constraints, each feature vector of the input is visited only once (every step makes a forward move). There is a striking similarity between templates and the DTW distance and HMM's with Viterbi alignment. Both are based on searching an alignment path between an input and an HMM model, template respectively. Therefore, we can also see a template as a graph-like structure as illustrated in fig. 2.4. The different arcs denote the different options defined in DTW with Itakura constraints of equation

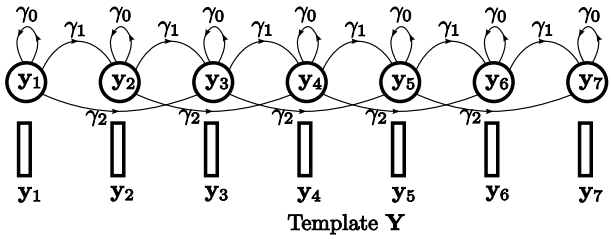


Figure 2.4: A template "model"

(2.17) and corresponding weighting factors. The "nodes" in the figure correspond to a single feature vector.

2.4 A template based speech recognition system

In this and the following sections, we will discuss the architecture of a template based speech recognition system for continuous speech recognition (TBSR). As explained before, in order to perform continuous speech recognition, the input should be aligned to all possible template sequences in order to choose the best sequence as the recognized sentence. Of course the number of template sequences is huge. Hence it is not feasible to simply list all of these possible sequences. Therefore, a TBSR system has been developed by De Wachter et al. [20, 24] that is able to cope with this huge search space.

The basic structure of this TBSR system is shown in fig. 2.5. Apart from the preprocessing unit, which is equal to an HMM preprocessing stage. The preprocessing has already been described in § 1.3. The system has two main components. In the template selection step, database templates that match with the input at a particular time are selected. This already limits the number of template sequences. For this purpose, a time filter algorithm has been constructed by De Wachter et al. [20], which selects templates from the database and inserts them in a phoneme graph. This component is discussed in § 2.4.2.

The second component is the main search algorithm. The search is performed by a token passing beam search on the activation graph con-

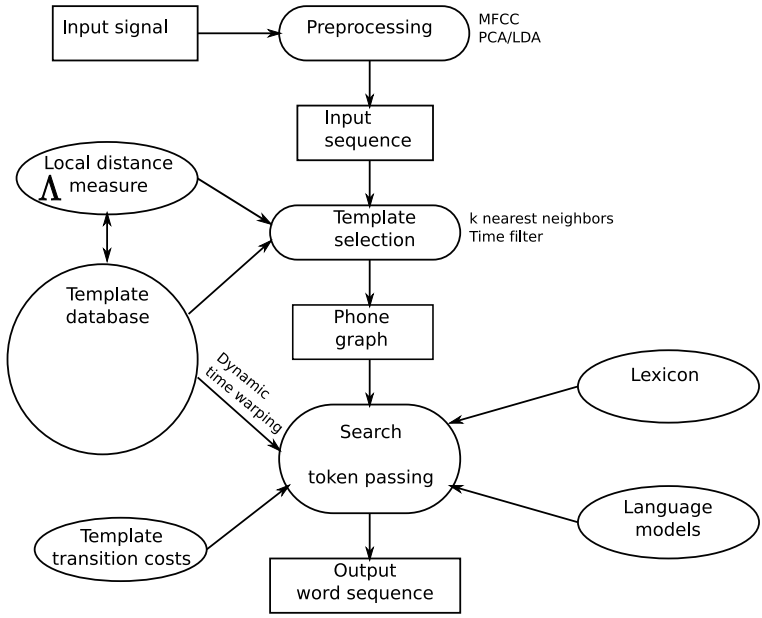


Figure 2.5: Template based speech recognition: system lay-out

structed by the template selection. This component is explained in § 2.4.3.

2.4.1 Top-down or bottom-up?

As already mentioned in previous sections, the quest for searching the most likely sentence means finding the most probable sequence of templates. In smaller tasks such as digit recognition, the most obvious choice is to take each utterance of a digit as a template. The search space then consists of a graph where each template is represented as an arc as shown in fig. 2.6. A top-down calculation of the most likely hypothesis is a search through a graph with a branching factor equal to the number of templates in the database. It is easy to see that even for a "simple" problem as digit string recognition, this search space increases by a factor 1000. The number of combinations of digits into digit strings of two digits is even higher: 11000^2 possibilities in TBSR as opposed to 11^2 possibilities in HMM based ASR.

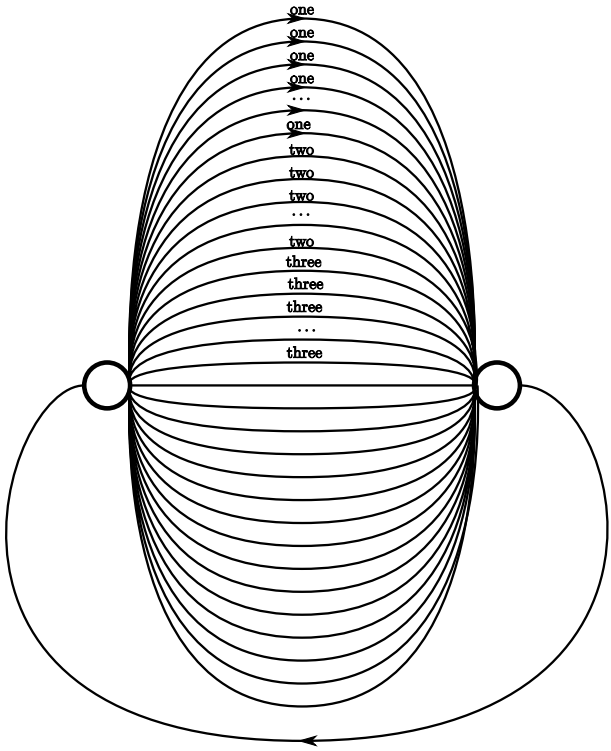


Figure 2.6: The search space for connected digit recognition

Things become even more complicated when tackling the task of large vocabulary speech recognition (LVSR). In LVSR, the most obvious choice for a template unit seems the phoneme. In that case however, the search space consists of every possible sentence allowed by the grammar, where every word has different pronunciation variants as defined in the lexicon and each phoneme has lots of different phoneme templates that match with it.

The previous paragraphs describe a top-down view of the search space. This top-down view creates a search space that is way too large to handle efficiently, even on modern computers. This motivates the need for

a bottom-up procedure to reduce the size of the search space. It is easy to imagine that a lot of the different possible alternatives in the search space are very unlikely in practice. For instance, if in reality the input starts with a 't' phoneme, it is very unlikely that this will match with a template containing an 'a'. These phones are simply too different. This motivates the search for a bottom-up procedure that selects a set of "possible" templates prior to performing a search through the search space. This procedure limits the number of possibilities for a portion of the input to the templates that are selected based on a matching score for that part of the input.

The size of the search space can now easily be kept under control by limiting the number of possible templates that this selection strategy can produce.

To perform this bottum-up speech recognition, De Wachter et al. created a bottom-up template based speech recognition that consists of two main components [24, 20]. The first component is a template selection strategy, called "time filter". The second component is a speech decoder based on the "token passing" paradigm. This decoder only takes into account templates that were selected with the template selection algorithm.

2.4.2 Template selection: the time filter algorithm

The first stage of the TBSR system is a template selection step. This step is intended to reduce the search space size. Template selection occurs by means of the "time filter" algorithm, designed by De Wachter et al. [20, 24]. The algorithm will be described briefly in this section. For more information, we refer to [20].

The time filter algorithm is based on the idea that, if a template matches with a part of the input, there should be quasi-diagonal patterns with small distances in the distance matrix of that particular template and the matching input segment. I.e. if a segment of the input speech contains an "a" sound, then this will match well with examples (templates) of that "a" sound in the database. This match appears as diagonal patterns in the distance matrix between that part of the input speech and the template.

The time filter algorithm will look for these diagonal patterns. To

reduce the number of local distance calculations, the algorithm first computes the k-nearest neighbors (k-NN) for each input frame. It only looks for diagonal patterns in these selected nearest neighbors (NN) and not in the full distance matrix. The efficient computation of these k-nearest neighbors has been studied for this research. This topic is discussed in chapter 6.

The main algorithm is shown in fig. 2.7. The ultimate goal of the algorithm is to "activate" templates that are close to the input. Activation of the template means that the template will be taken into account for the main search algorithm. The algorithm starts by activating templates for which the template head (the first frame(s)) has a matching nearest neighbor with the input, as shown in lines 3 and 4 of the algorithm. The activation contains a region where further nearest neighbors must lie within in order to continue the activation. This activation region is created in line 21. If new nearest neighbors for further input frames are found within that activation region, the region is updated (enlarged), as shown in lines 5, 8 and 9. Finally, if the activation region reaches the end of the template, a diagonal pattern is found and the entire template is activated for use in the search.

An example of the application of the time filter algorithm to a segment of input speech is shown in fig. 2.8. On the horizontal axis is the sequence of input speech frames and on the vertical axis, some templates are shown with their respective sequence of frames. The nearest neighbors for each input frame are shown with a cross. The activated templates, showing a diagonal pattern in the nearest neighbor selections, are highlighted with a dashed polygon.

Storing the activated templates: the activation graph

After the time filter algorithm has activated a template, this template has to be inserted into a data structure usable for the search. The chosen data structure is called the *activation graph*. The activation is a graph data structure that contains one node for each frame in the input. Each activated template is represented by an arc in this graph. The start and end nodes of the arc are the first and last frame of the input with which

```
1: function filter()
2: KNN = compute_kNN % compute the k
3: for i = 1 to size(kNN) do
4:    if template_head(KNN_i) then
5:       trigger(KNN_i)
6:    else if active(KNN_i) then
7:       if activation_too_old(KNN_i) then
8:          cleanup_activation_window(KNN_i)
9:       else
10:         trigger(KNN_i)
11:      end if
12:   end if
13: end for
14:
15: function trigger(NN)
16: if in_tail_region(NN) then
17:    if template_end_score(NN)>threshold then
18:       activate_template(NN)
19:    end if
20: end if
21: activate_successors(NN)
```

Figure 2.7: The time filter algorithm [20]

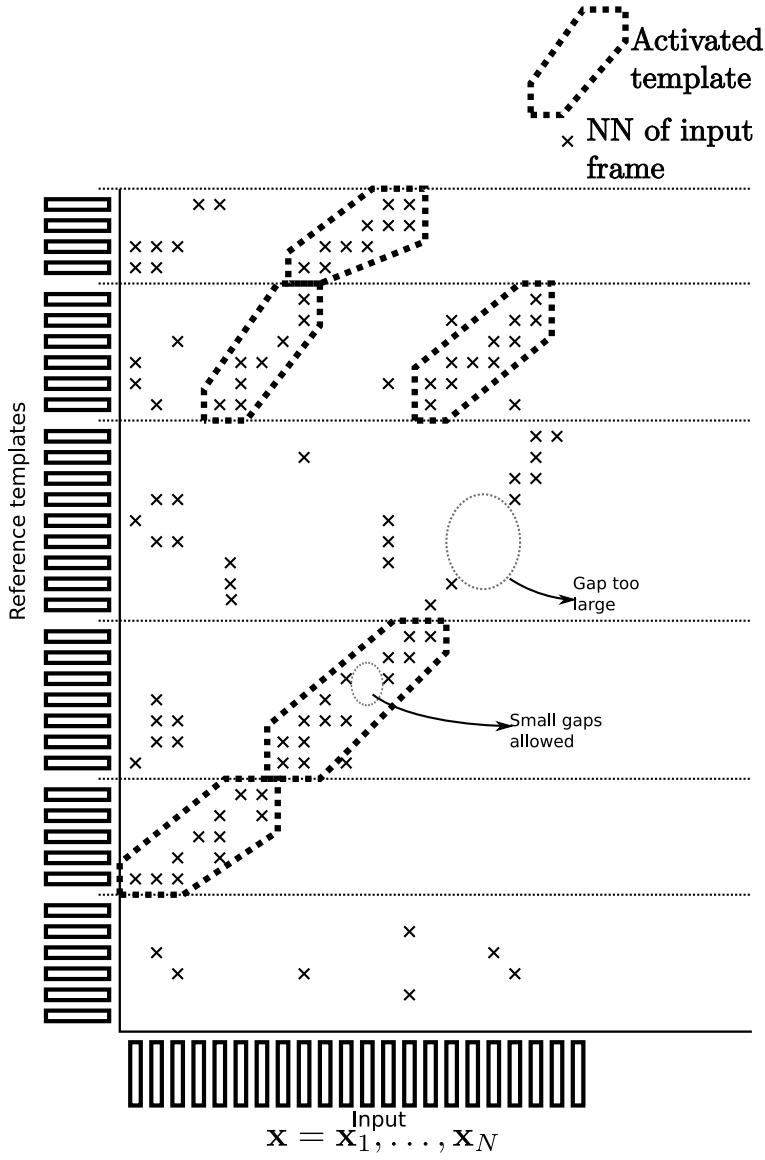


Figure 2.8: Example run of the time filter algorithm

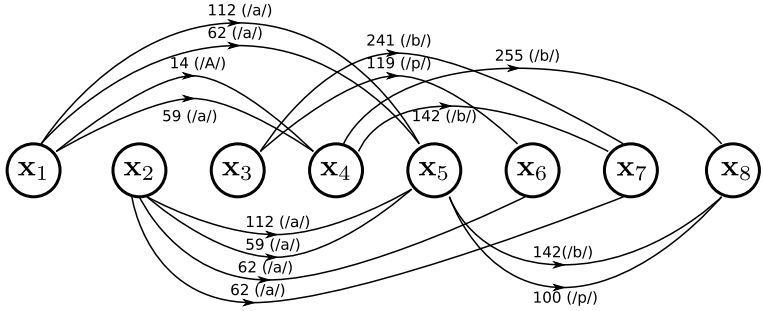


Figure 2.9: Example activation graph

the template matches. These arcs are tagged with the template number, template identity and DTW score between the template and the part of input speech it corresponds with. An example of such an activation graph is shown in figure 2.9. The DTW score of input and template is omitted in this figure.

On the number of nearest neighbors

One important parameter in the time filter algorithm is the number of nearest neighbors that has to be selected for each input frame. If too many nearest neighbors are selected, too many templates will be activated and the size of the search space will be too large to perform speech recognition efficiently. On the other hand, if the number of nearest neighbors is chosen too low, the search space will be reduced significantly and it may well be that the probability of finding the correct solution is very small.

One important criterion in this regard is the graph error rate. The graph error rate represents the most optimistic scenario. It indicates if the correct solution is possible with the graph at hand, or, if not, it states how different the correct solution is from the best possible solution with the graph. It is expressed in the minimal number of substitutions, insertions and deletions of phonemes needed to insert the correct solution in the graph.

Naturally, the graph error rate is influenced by the number of selected

nearest neighbors, if all nearest neighbors are selected, the error rate will be 0% and if no neighbors are selected, error rate will be 100%. The optimal number of nearest neighbors to select is a trade off between the average fan out of the activation graph (and the associated computational cost) and the graph error rate of the activation graph.

At this moment, there is no automatic procedure for selecting the number of nearest neighbors to take into account, but in general, it seems that taking 1% of the total number of frames in the database seems a good trade off between the graph error rate and the fan-out of the activation graph.

2.4.3 Template matching: token passing search algorithm

In this template based form of speech recognition, the decoding occurs by concatenating selected phonemes into words and sentences to form hypotheses. In continuous speech recognition, each hypothesis corresponds to a possible sentence. In order to make a selection of the most likely hypothesis, a method is needed to evaluate these hypothesis and tag them with a score. The output of a speech recognition system must be one single phrase.

One very suitable method for generating and scoring these hypotheses is the token passing paradigm, proposed by Young et al. [93]. The token passing paradigm is a time synchronous paradigm with a time step for each frame in the input speech. Each token is allocated to a time τ representing the input frame number it currently corresponds to. The token itself represents a partial solution from the first frame of the input up to the time τ. Within a template, a token is propagated through the template using the constraints (e.g. Itakura) of the DTW algorithm. When a token reaches the end of a template, it is propagated to the first (or second) frame of the following templates that are possible taking into account the lexicon and language models. When two tokens arrive in the same state (template frame) at the same time τ, they are merged, i.e. only the best token is retained for further search.

```
1: function tokenpassing()
2: for t = 1 to T do
3:    for i = 1 to S do
4:       Pass a copy of the token in state i to all connecting states j
         incrementing its s value by p_ij + d_j(t)
5:    end for
6:    Discard original tokens
7: end for
8: for i = 1 to S do
9:    find the token in state i with the smallest s value and discard the
      rest
10: end for
```

Figure 2.10: Token passing algorithm in one single model [93]

Pruning

Another very important aspect of token passing is pruning. Pruning is the automatic removal of active tokens from the list of tokens. It is necessary because the number of active tokens in the search graph tends to become very large over time. This has two main reasons. First, using DTW, there are many possibilities to match part of the input with a template. The boundaries between phonemes in the input are unknown. This is due to the time warping possibilities of the DTW algorithm. If a segment of the input of length t is close to a template according to the DTW distance, then the same segment with one feature vector more (length $t + 1$) will probably also match well with that template. Information about the start and end nodes is available in the activation graph. It seems very useful to use this information to keep the within-template distribution of tokens under control. The second propagation of tokens occurs at template boundaries. At these boundaries, tokens need to be propagated to the first frame(s) of all possible next templates. It has been determined experimentally that this average fan-out lies somewhere between 50 and 500 if 1% nearest neighbors are used in the time filter [20].

Combining information

It is clear that the search in speech recognition has to combine different information streams derived from different knowledge sources. First, there is the activation graph generated by the time filter algorithm, containing all selected templates between every pair of input frames. The second source of information is the lexicon. The lexicon contains the pronunciation of every word the recognition system knows, possibly with different pronunciation variants. The lexicon is usually formulated as a list of words and their pronunciation, but this information can easily be represented by a graph, where each arc represents a phoneme. The output of the graph is a word. The third level of information comes from the language model. These language models are word pair grammars or word N-grams. The language model score is generated when a new word hypothesis is received. In this case, the language model score is updated with this new context.

The tokens used for token passing are entities that contain a reference to a node in each of these graphs, the current position in the input, represented by the frame number in the activation graph, the current position in the activation graph and the current language model history represented by the language model graph. The tokens also contain a score generated by a combination of the information in the graphs. When tokens have to be propagated to the next node, information is combined: when a token reaches the end of a template in the activation graph, it is propagated to all following arcs, given that there is a matching outgoing arc in the lexicon graph. It is clear that the number of tokens after propagation of that single token is at most equal to the product of the fan-out of activation and lexicon graphs, but in practice, this is not the case as it requires all symbols (phonemes) to be the same.

2.4.4 Concatenating templates: transition costs

When performing a token passing search, tokens are propagated from one template into other possible successor templates. The possible successors are defined by the activation graph generated by the time filter algorithm. It is very well possible that these successive templates are very different in nature. They can be produced by a different speaker, or a speaker with

a different gender. There can be a context mismatch, i.e. the phoneme that preceded the template in its original context is different from the one that precedes it in the input speech. Other possibilities are differences in dialect regions etc...

For this purpose, a model for inserting transition costs between templates has been introduced by De Wachter et al. [20]. This model for template transition costs can be combined easily with the token passing decoder. All that has to be done is updating the score of tokens with the necessary cost when propagating them between templates.

A more in-depth discussion about the implementation of this token based paradigm in our template based speech recognition system is presented by De Wachter et al. [20].

Chapter 3

Distance measures

The next three chapters describe one of the main research topics of the research described in this dissertation. The quest for finding appropriate distance measures for template based speech recognition.

This chapter is concerned with defining the distance measures and introducing weighted, class dependent distance measures. The partitioning of the data into classes is described in chapter 4. The training of these distance measures is the subject of chapter 5.

With a distance measure, we mean a measure for the distance between two feature vectors $\mathbf{x}$ and $\mathbf{y}$. We call it a measure because, as we will see, the distance measures we define are not always metrics. The chapter is initiated with the formal definition of a metric and the Euclidean distance measure. We motivate the need for scaled distance measures and introduce a class-dependent scaled distance measure in § 3.3.2. In § 3.4 we describe several options for weighting the distance measure and introduce the novel concept of block diagonal weight matrices.

3.1 Definition

In mathematics, a distance between elements is usually represented by a metric.

Definition 3 *A* metric *is a real-valued function* $d(x, y)$ *that satisfies the*

following properties:

1. $d(x, y) \geq 0$*:* Non-negativity
2. $d(x, y) = 0 \Leftrightarrow x = y$
3. $d(x, y) = d(y, x)$*:* Symmetry
4. $d(x, y) + d(y, z) \geq d(x, z)$*: The* triangle inequality

This definition defines in a mathematically rigorous way what is usually considered as a distance between objects. A derived concept is a *quasi-metric*, where the symmetry property no longer holds. A *semimetric* on the other hand does not satisfy the triangle inequality.

3.2 The Euclidean distance measure

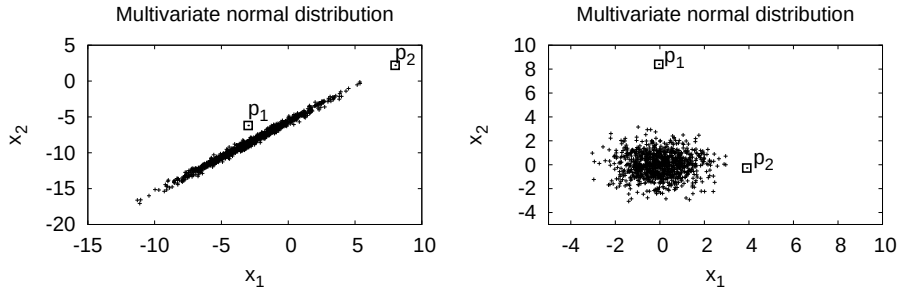
The acoustic space contains a large set of feature vectors derived from speech frames. Earlier, we explained a method to recognize speech by comparing it with example speech using the DTW algorithms. One of the main operations in this DTW algorithm is the calculation of distances between frames. The most straightforward choice would be to take the Euclidean distance metric to compare an input frame $\mathbf{x}$ with a reference frame $\mathbf{y}$. The Euclidean distance is defined as follows:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T (\mathbf{x} - \mathbf{y})} = \sqrt{\sum_{i=1}^{D} (x_i - y_i)^2}, \tag{3.1}$$

with D the dimension of the frames.

This Euclidean distance however has a serious drawback when we want to use it for speech and pattern recognition.

Data distribution The Euclidean distance only works optimally if the data points in the space are uniformly distributed across the space. For most real-life problems however, this usually is not the case. An illustration of how it goes wrong with the Euclidean distance is shown in fig. 3.1.



(a) Normal

(b) Standard Normal

Figure 3.1: Gaussian data distribution and its normalized form

In fig. 3.1a, a data set with a Gaussian distribution is shown, together with two points p_1 and p_2. It is clear that p_1 is a lot closer to the point cloud than p_2 is. However, when the whole point cloud is transformed to a standard normal distribution, as shown in fig. 3.1b, the picture is completely different, and now p_2 lies closer to the point cloud. Naturally, the second way of looking at the data makes much more sense, since the distance of a point to the cloud also reflects the probability of the point belonging to the class. It seems a good property if this is also reflected in the distance measure. The Euclidean distance however is unable to cope with this without transformations of the data space.

The Minkowski distance measure A generalization of the Euclidean distance is the Minkowski distance. It is defined as follows:

$$d(\mathbf{x}, \mathbf{y}) = \left(\sum_{i=1}^{D} |x_i - y_i|^p \right)^{1/p}. \tag{3.2}$$

However, this distance measure is not suited for our purposes. These distance measures only takes into account the components of $\mathbf{x}$ and $\mathbf{y}$. I.e. they have the same drawbacks as the Euclidean distance. Preliminary experiments with these distances have shown bad results. Other ways of

fine-tuning the Euclidean distance will have to be found. One of these methods is obtained by scaling the Euclidean distance, as explained in the next section.

3.3 Scaling the distance measure

To overcome the disadvantages of using Euclidean distance, we need to look for methods to improve upon it. One of these methods is scaling the Euclidean distance. Many variants of scaled distance measures have been proposed in different research fields [71, 72, 83, 47, 34, 90, 23]. We have adopted the idea of scaled distance measures [57]. In the next few paragraphs, some alternatives for scaling the distance measure are presented.

3.3.1 Global scaling of the distance

A first possibility is to scale the distance measure with a global weight matrix Λ. In this case, the distance measure is defined as follows:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Lambda (\mathbf{x} - \mathbf{y})}. \tag{3.3}$$

Note that this form of distance measure is still a metric if the weight matrix is positive definite, i.e.

$$\mathbf{z}^T \Lambda \mathbf{z} > 0, \text{ for all } \mathbf{z} \in \mathbb{R}^n \setminus \{\mathbf{0}\}. \tag{3.4}$$

This guarantees that the first property of a metric is fulfilled. The proof for the other properties is trivial.

Mahalanobis distance

A special case of the globally scaled distance measure is the Mahalanobis distance, where the weight matrix Λ is the inverse covariance matrix of the data Σ^{-1}, i.e.

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Sigma^{-1} (\mathbf{x} - \mathbf{y})}. \tag{3.5}$$

The Mahalanobis distance is able to solve scaling difficulties in the data, especially if the data has a Gaussian distribution. I.e. it makes the transformation shown in fig. 3.1 implicitly.

Disadvantages

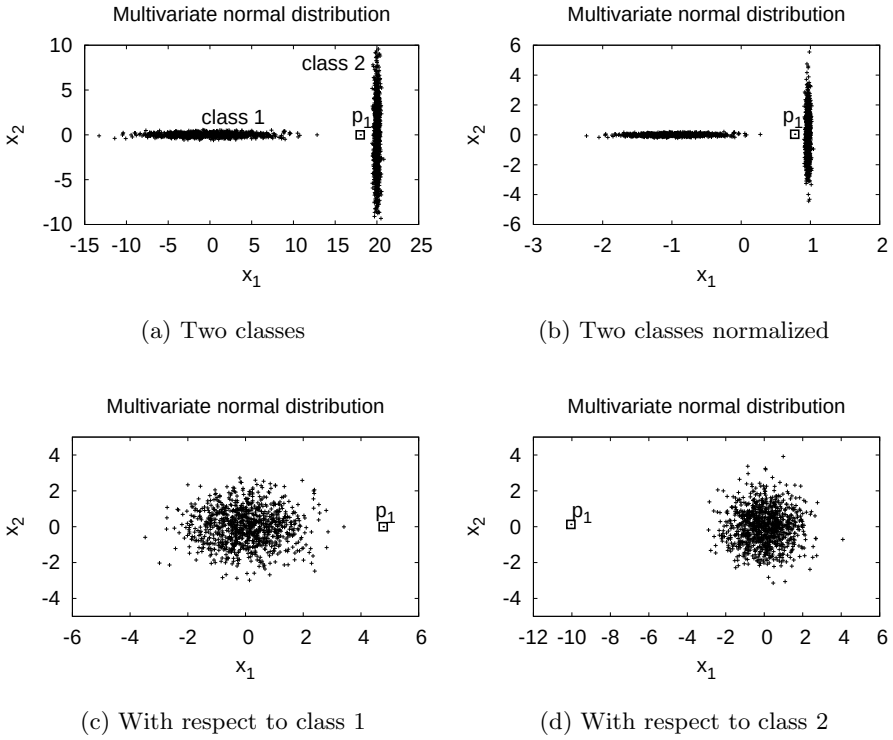
These globally scaled distance measures still suffer from some of the disadvantages of the Euclidean distance. They are able to solve most problems with global data distribution, however, when there are several classes of data, each with their own data distribution, these globally scaled distances will still perform badly on these data.

A simple example of this problem is shown in fig. 3.2. In fig. 3.2a, two classes of data points are shown. Deciding to which class point p_1 belongs using a Euclidean distance is not an easy task. If we would decide this by using a Euclidean distance measure, it would be classified as belonging to class 2, since it is a lot closer to class 2 than to class 1 in fig. 3.2a. However, in figs. 3.2c and 3.2d, the point p_1 is plotted with respect to class 1 and 2 separately, after normalization of the data to a standard normal distribution. It is obvious that in this perspective, p_1 is a lot closer to class 1 than it is to class 2. In this case, globally scaled distance measure will also perform badly. This is illustrated in fig. 3.2b, where the whole set of points is normalized, as would be the case when using the Mahalanobis distance.

From fig. 3.2, we can see that using a global distance measure, it is very difficult to decide to which class a prototype belongs, or very easy to misclassify it. However, we can also observe that if we look at the distance to both classes separately after a suitable transformation, the picture looks a lot more clear. This motivates the need for class dependent distance measures, in which the weights are also dependent on the data class which we are comparing to. This concept is explained in the next section.

3.3.2 Class-dependent distance measures

Due to the lack of expressive power of global weight matrices, researchers presented several alternative strategies to increase the number of parame-



(a) Two classes

(b) Two classes normalized

(c) With respect to class 1

(d) With respect to class 2

Figure 3.2: Prototype classification, two classes with Gaussian data distribution

ters. One such alternative is the introduction of class-dependent distance measures [71]. This assumes the data can be partitioned into classes, but this is not a problem for speech and pattern recognition. For the pattern classification problems discussed in this dissertation, the partitioning of the data into classes is inherent. For speech recognition however, this poses a more difficult problem with a lot of possible alternatives. The partitioning of the data into classes is discussed in chapter 4. From now on, we will assume that each element in the set of prototypes has a class label C_p for $p = 1, \ldots, N_K$, with N_K the total number of classes.

The class-dependent distance measure is defined as follows:

$$d(\mathbf{x};\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T \Lambda_p (\mathbf{x}-\mathbf{y})}, \quad \Lambda_p \in \mathbb{R}^{D\times D}, \tag{3.6}$$

with C_p the class to which the prototype $\mathbf{y}$ belongs. We also define the set of parameters Λ as the set of all individual weight matrices $\Lambda_i, \quad i = 1,\ldots,N_K$.

Properties of class-dependent distance measures

Note that some of the properties of a real metric are lost in equation (3.6). With this definition, the distance measure is no longer symmetric if $\mathbf{x}$ and $\mathbf{y}$ belong to a different class. Also the triangle inequality has to be relaxed: it is only valid if $\mathbf{y}$ and $\mathbf{z}$ belong to the same class. Within each class however, the distance measure still has all properties of a metric if all weight matrices Λ_i are positive definite.

One important choice to make is selecting the number of weights in each class. Several alternatives are discussed in the next sections.

3.4 Selecting the weight matrices

3.4.1 Diagonal weight matrices

The use of class-dependent distance measures with a diagonal weight matrix has already been proposed by different researchers, especially in pattern classification [71, 72]. We have investigated the applicability to template based speech recognition [22, 57, 58, 59]

When using a diagonal weight matrix, the distance measure of equation (3.6) looks as follows:

$$d(\mathbf{x},\mathbf{y}) = \sqrt{\sum_{i=1}^{D} \lambda_{p,i}(x_i - y_i)^2}. \tag{3.7}$$

With a diagonal weight matrix, each class only had D parameters to train.

The main advantage of this approach is that distance evaluations can be computed very efficiently. One distance calculation requires $\mathcal{O}(D)$ calculations. The disadvantage is that it assumes that the individual features of the examples are independent of each other. For most real-life applications however, this is not the case. We can include the dependencies of features by including off-diagonal elements in the weight matrices. In the next subsections, two approaches to increase the number of parameters in the weight matrices are presented.

3.4.2 Full weight matrices

Diagonal weight matrices are unable to model correlations present in the data, i.e. they treat each feature in the data as independent of the other features. In practice however, these dependencies do occur in the data.

In the literature, many data decorrelation techniques have been described, based on a variety of criteria, such as principal component analysis, linear discriminant analysis, etc. However, these techniques only provide a global decorrelation of the data. In other words, the global data cloud no longer shows any correlation, but this does not make sure that these correlations are also absent within the different classes present in the data.

The most straightforward way to model these dependencies is to use full weight matrices instead of diagonal weight matrices. This approach however has two main disadvantages. First, the number of parameters that need to be trained simultaneously increases drastically with the number of dimensions and classes. This number is equal to $N^2 N_K$. For example, for the speech recognition experiments, we have 25 dimensions and about 2000 classes of feature vectors, resulting in a parameter space of dimension 1250000. In pattern classification, these feature vectors are also called prototypes.

A second disadvantage is that the calculation of a distance between two elements is $\mathcal{O}(N^2)$, which results in a high computational cost if many distance evaluations are necessary. The problem at hand is the k-nearest neighbor classification problem. If the dimension of the problem is high however, the k-nearest neighbors problem is a hard problem as we will see

in chapter 6. This is due to the "curse of dimensionality", which occurs in almost every k-nearest neighbor selection algorithm.

Last but not least, using full weight matrices means that more parameters need to be estimated or trained. In order to train more parameters, more data is needed to reliably estimate these parameters. In practice however, it is not always the case that enough data is available to estimate them in a reliable way.

3.4.3 Block diagonal weight matrices

To continue the previous discussion: diagonal weight matrices assume statistical independence between features. However, for most pattern classification problems, this assumption is false. Full weight matrices on the other hand solve this problem, but introduce performance and weight estimation issues.

On the other hand, it can be observed that full weight matrices cover all possible correlations present in the data. For most pattern recognition problems however, it is not necessary. Clearly, the correlations within the data are limited to correlations between some features.

Therefore, we want to introduce a weight matrix structure that is able to model the present dependencies but introduces only the parameters necessary to model these. For this purpose, we introduce the concept of block diagonal weight matrices. For this, we have designed a method to create these block diagonal, sparse matrices. The idea of using block-diagonal matrices has already been proposed for estimating covariance matrices by the HMM speech community. Wang et al. created a block-diagonal covariance matrix structure for fast likelihood computations [91]. Koshiba et al. used covariance matrices with a block-diagonal structure for HMM-based speech recognition [48]. Matton et al. created a method for constructing weight matrices inspired on these approaches [59, 60].

The main idea of this method is that an element in the weight matrix from a particular class is only included if there is a statistical correlation between the two features for that particular class. The algorithm for constructing these matrices is shown in fig. 3.3. The idea is simple: insert a weight $\lambda_{p,i,j}$ only if the correlation between dimension i and dimension

Require: threshold T, reference database $\mathbf{Y}$

```
for all classes C do
  for all dimension i do
    for all dimension j > i do
      c ← compute corr(Y_{C,i}, Y_{C,j})
      if c > T then
        insert λ_{C,i,j} into training parameters
        insert λ_{C,j,i} into training parameters
      end if
    end for
  end for
  expand_matrix(Λ_C)
end for
```

Figure 3.3: Sparse weight matrices: weight selection

j in class C_p is sufficiently high. In this way, a real symmetric matrix is obtained. The function expand_matrix creates the block structure: if a correllation is found between features i and j and between j and k, the parameter resembling i and k is also inserted in the weight matrix. If more than three features are correlated only the ones with highest correlation are retained. The function expand_matrix selects the feature pairs with highest correlation first. The main idea behind this procedure is that features that are clustered are represented by a full matrix for this subset of features. In order to limit the number of parameters, we have restricted the clusters to a size of three. In this way, and after a suitable permutation of the features, a block-diagonal matrix structure is obtained with block of size 1×1, 2×2 and 3×3. An example of such a weight matrix is shown in fig. 3.4. Obviously, the block-diagonal structure is only obtained after a suitable permutation of the individual features. Normally these "blocks" are scattered through the matrix.

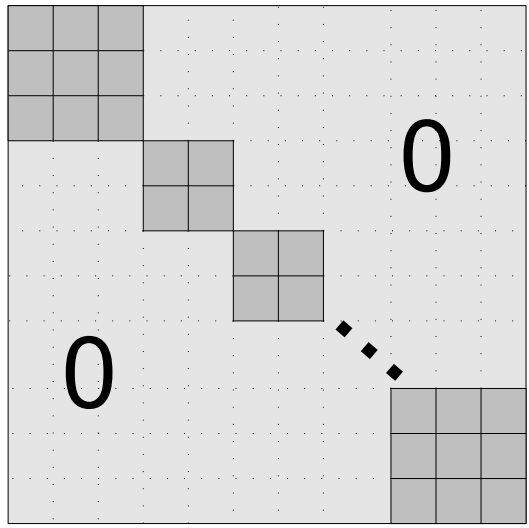


Figure 3.4: A block-diagonal weight matrix

3.4.4 Implementation

When implementing the techniques and weight matrices of the previous sections, some important issues need to be pointed out. As has been mentioned earlier, the weight matrices must be positive definite. One of the necessary conditions for a weight matrix to be positive definite is that all diagonal elements are positive. To ensure this, one possible method is to square all diagonal elements. Note the similarity with a covariance matrix, where the diagonal elements contain the squared standard deviations. This approach has been used throughout the experiments presented in this dissertation.

Furthermore, we have to ensure that calculations with the sparse block-diagonal matrices can be performed fast. For this purpose, several numerical software libraries exist. For this work we used the Sparse BLAS library [29]. A note on the sparse BLAS library is presented in appendix A.

3.5 Global data decorrelation

The most obvious way to keep the number of parameters in the block diagonal weight matrix as small as possible is by performing a global data decorrelation. We hope that this will also reduce the correlations within each class. In the literature, there exist several techniques for performing a data correlation. In the next sections we present a brief overview of these techniques.

3.5.1 Principal component analysis

In general, principal component analysis (PCA) is a technique that transforms data with a number of possibly correlated features into a smaller-dimensional set with uncorrelated features. It was first introduced in 1901 by Pearson et al. [74]. We will explain the basic principles of PCA briefly.

Formally, suppose X is a set of D-dimensional data vectors $\mathbf{x}_1, \ldots, \mathbf{x}_N$, where each $\mathbf{x}_i$ can be written in terms of a D-dimensional basis $\mathbf{v}_1, \ldots, \mathbf{v}_D$:

$$\mathbf{x}_i = \sum_{j=1}^{D} x_{i,j}\mathbf{v}_j. \tag{3.8}$$

PCA tries to find a new representation in a lower B-dimensional space with basis $\mathbf{u}_1, \ldots, \mathbf{u}_B$.

$$\hat{\mathbf{x}}_i = \sum_{j=1}^{B} \hat{x}_{i,j}\mathbf{u}_j, \tag{3.9}$$

where the lower-dimensional vector $(\hat{x}_{i,1}, \ldots, \hat{x}_{i,B})$ is the vector resulting from the PCA process. PCA does this by minimizing the squared error (measure for information loss) between $\mathbf{x}_i$ and $\hat{\mathbf{x}}_i$, defined as:

$$||\mathbf{x}_i - \hat{\mathbf{x}}_i|| = \sqrt{(\mathbf{x}_i - \hat{\mathbf{x}}_i)^T(\mathbf{x}_i - \hat{\mathbf{x}}_i)}. \tag{3.10}$$

It has been proven that this PCA transformation can be computed by writing $\mathbf{x}_i$ in terms of the eigenvectors of the sample covariance matrix, corresponding to the highest eigenvalues. Note that this is a linear transformation of the original data vectors.

PCA has some important limitations. It only decorrelates the whole data set without taking into account available class labels. In our class-dependent distance measure context however, we not only want to decorrelate the data set itself, we also want to decorrelate the classes as much as possible, because we want to reduce the number of necessary parameters for the block diagonal distance measure as much as possible. PCA is unable to do this. One way to solve this is by performing a discriminant analysis technique, which does take the class labels into account.

3.5.2 Linear discriminant analysis

Fisher linear discriminant analysis (LDA) [28] tries to find the direction (linear discriminant) that best separates two classes in a data set. As opposed to PCA, LDA explicitly models the classes present in the data and separates them as good as possible. The result from LDA is the projection of the data vectors $\mathbf{x}_i$ into a one-dimensional space.

LDA tries to transform the data set X by searching for the direction $\mathbf{w}$ that best discriminates the data. The component y_i of $\mathbf{x}_i$ along that direction $\mathbf{w}$ can then easily be computed by taking the dot product between $\mathbf{x}$ and $\mathbf{w}$:

$$y_i = \mathbf{w}^T \mathbf{x}. \tag{3.11}$$

It can be proven that this $\mathbf{w}$ can be found by maximizing the following criterion:

$$J(\mathbf{w}) = \frac{\mathbf{w}^T S_B \mathbf{w}}{\mathbf{w}^T S_W \mathbf{w}}, \tag{3.12}$$

where S_B is the between-classes scatter matrix

$$S_B = \sum_{k=1}^{K} N_k (\mu_k - \overline{\mathbf{x}})(\mu_k - \overline{\mathbf{x}})^T, \tag{3.13}$$

with N_k the number of available feature vectors in class C_k. and S_W is the within-class scatter matrix

$$S_W = \sum_{k=1}^{N_K} \sum_{i=1}^{N_k} (\mathbf{x}_{k,i} - \mu_k)(\mathbf{x}_{k,i} - \mu_k)^T, \tag{3.14}$$

where

$$\boldsymbol{\mu}_k = \frac{1}{N_k} \sum_{i=i}^{N_k} \mathbf{x}_{k,i}, \tag{3.15}$$

$$\overline{\mathbf{x}} = \frac{1}{N} \sum_{k=1}^{N_K} N_k \boldsymbol{\mu}_k, \tag{3.16}$$

and with N the total number of feature vectors.

A natural generalization of the Fisher LDA is multiple discriminant analysis, where multiple classes are defined. In this case the original D-dimensional data is projected into a set of a $N_K - 1$ dimensional data by finding $N_K - 1$ discriminant directions that separate the data as good as possible.

However, as LDA also performs a global decorrelation of the data, it still does not guarantee that the data within the classes is decorrelated as much as possible. It does not take into account the scatter matrices of the individual classes.

3.5.3 Mutual information discriminant analysis

In an attempt to overcome the disadvantages of both PCA and LDA, Demuynck et al. introduced a new type of discriminant analysis based on the mutual information criterion: mutual information linear discriminant analysis (MIDA) [25]. The MIDA transform tries to keep the Kullback Leibler distance between the original class distributions and the transformed ones as low as possible. If X denotes the data before discriminant analysis and Y denotes the data after dimension reduction, the MIDA criterion tries to minimize the Kullback Leibler distance between the density functions $p(C|X)$ and $p(C|Y)$ with C the class label. This distance is defined as follows:

$$D(p(C|X)||p(X|Y)) = \int_{\mathcal{X}} \sum_{k=1}^{N_K} p(c_k, x) \log \frac{p(C_k|x)}{p(C_k|y)} dx. \tag{3.17}$$

The minimization is equal to the minimization of the conditional class entropy $H(C|Y)$ [17]. For a definition of conditional entropy, see § 5.4.3.

This is also equal to the minimization of the conditional mutual information $I(C; X|Y)$. The method is explained in more detail by Demuynck et al. [25].

The MIDA transform is mentioned because it is present in the ESAT Spraak recognition system [26] and has been used for all our experiments with template based speech recognition. The MIDA transform occurs as a final step in the speech preprocessing. It is applied to the feature vectors containing the MFCC's and their first and second derivatives.

Chapter 4

Defining the classes

In § 3.3.2, a class-dependent distance measure was defined. However, the way in which prototypes are allocated to a specific class still has to be specified. Our ultimate goal is to obtain a partition of the training data. I.e. we want to tag each prototype (feature vector, example) in the database with one single class label.

Definition 4 *A* partition *of a set X is a set of subsets $P_i \subset X$, for which holds:*

1. $\cup_i P_i = X$
2. $P_i \cap P_j = \emptyset \quad \forall i \neq j.$

In general, there are two alternatives for defining the classes used in the class-dependent distance measure. The first option is to use a priori knowledge about the data, knowledge that is available from meta-information in the database. Clustering using a priori information is called supervised clustering. It is discussed in § 4.1. The other option is to use unsupervised clustering methods on the data without using any form of a priori knowledge of the data. In this case, the clustering is only dependent on the spatial distribution of the data itself. This form of clustering is discussed in § 4.2.

The contribution of this chapter is the combination of supervised and unsupervised clustering methods to partition the data. The techniques we investigated are described in § 4.3.

4.1 Supervised clustering

Meta-information is available in most databases for speech and pattern recognition. In pattern classification, such information comes with the training data. Each prototype has a target class. These target classes can be readily used for allocating distance measure classes to these data points. In speech recognition, the situation is a lot more complex. In the next paragraph we will present an overview of methods to include meta-information of the speech data into the class definitions for the distance measure.

4.1.1 Context-independent phoneme-based classes

Reference data for speaker independent speech recognition comes from a database with hours of speech data. This data is labeled with the correct classification, namely the sentence that is uttered in each entry. In speech recognition, correct classification means recognizing the sentence (or sequence of words) correctly. Together with this data comes a phonetic dictionary, which lists all words known to the automatic speech recognition system together with their phonetic transcription into phonemes.

These individual phoneme labels could be used as class labels for the class-dependent distance measure, i.e. each frame belonging to a specific phoneme obtains a class label based on that phoneme. This phonetic segmentation of the data is however not readily available. There are two main method for obtaining them: manual labeling and forced alignment.

Manual labeling

In manual labeling, the labeling of the data is done by hand. A listener listens to the speech data using specialized software and puts markers between the individual phonemes. It is needless to say that this method

of labeling is very time consuming and prone to error. This has been done for several speech corpora, but in general it is unavailable.

Forced alignment

An automatic way of defining the boundaries between phonemes is by using forced alignment. In forced alignment, these boundaries are defined. by an existing speech recognition system. The training data speech is passed through the existing speech recognition system and the boundaries found by the recognizer are written on the output. In HMM-based speech recognition, this technique is extensively used for training acoustic models. Phonemes are aligned using the speech recognizer and this alignment is used for improving the acoustic models themselves. However, forced alignment also poses a chicken-and-egg problem. In order to do forced alignment, one needs acoustic models, and in order to train acoustic models, a phoneme segmentation is required. This however can be solved by using acoustic models trained on a manual labeling as a bootstrap for this procedure. Usually, the training of HMM models takes several iterations of alignment and training.

Note that forced alignment (and also manual labeling) can introduce errors. If words are pronounced in an incorrect way, forced alignment will take incorrect acoustic models to perform the segmentation into phonemes. If such an inconsistency is detected, the uttered word could be removed from the training material, or, if an inconsistency is happening several times with the same phoneme, the pronunciation variant could be added to the dictionary. However, no research was performed on this topic.

Context independent classes however still have limited capabilities in the context of defining classes for the distance measure as we will see in the next section.

4.1.2 Context-dependent phoneme-based classes

Context-independent phonemes do not provide a lot of fine grained information. It is well studied that individual phonemes are influenced by their neighboring phonemes, sometimes in a very strong way. This is caused by the fact that repositioning the articulators in the mouth is a continuous

movement. In this way, pronounced sounds are influenced by the previous and forthcoming sounds, this is also called coarticulation.

It is obvious that acoustic properties of two identical phonemes, but with a different context can be different. This has already been observed by the HMM-based speech community. The use of context-dependency in HMM models has been well studied. Initially, an HMM was created for each different phoneme regardless of its context (its neighboring phonemes), resulting in context-independent acoustic models. The introduction of context-dependent models, where a different HMM is created for every phoneme with its left and right context, has given a serious boost to recognition accuracy. Such a label of a phoneme together with the label of the left and right neighbors of the phoneme (the context) is also called a triphone.

In this evolution, we can note a similarity between HMM-based speech recognition and the use of local distance measures. In HMM-based speech recognition, the database is split into context dependent triphones, where each context dependent triphone has a three-state HMM-model with a Gaussian (mixture) model as observation probabilities. In template based speech recognition, the database can also be split in context-dependent triphones where speech frames belonging to different context dependent triphones are given a different class label, and thus a different weight matrix for the local distance measure.

However, there are some trainability issues we want to apply this approach in a crude manner. The first problem arises when looking at tables with triphone counts for the databases. Duchateau et al. performed this exercise for the SI-284 WSJ1 training database[1] and noted that 22674 triphones occur in this database of which only 7724 appear more than 50 times [27]. In HMM-based speech recognition, this poses a problem for estimating the covariance matrices of the Gaussians accurately. In template based speech recognition, this will pose a problem to train the weight matrices of the distance measure accurately.

This problem of untrainability has been solved by introducing state tying in HMM models. In state tying, several context dependent tri-

[1] For an overview of speech databases, please refer to appendix B.

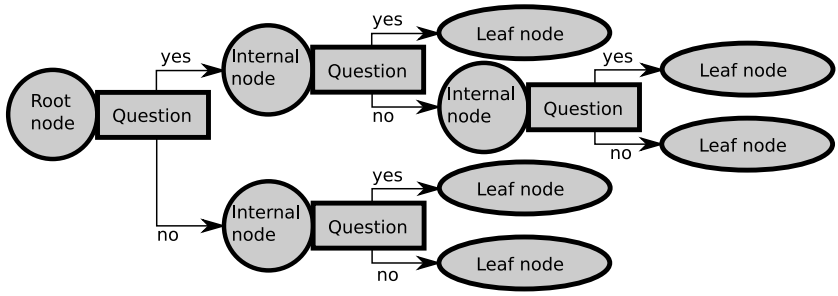


Figure 4.1: A binary decision tree

phones share some of their HMM states. Several methods have been introduced for state tying based on context clustering [30, 92], phonetic decision trees [10, 27] etc. Because the phonetic decision tree offers a nice, data driven approach to define classes, and because software for clustering based on phonetic decision trees is available in the ESAT Spraak speech recognition system [26], this seems a useful approach for defining classes in a supervised way. The concept of phonetic decision trees is introduced briefly in the next paragraphs.

Phonetic decision trees

The use of phonetic decision trees was first introduced by Bahl et al. [10] as a method to perform state tying in HMM-based speech recognition. The method is based on the theory of decision trees. A decision tree is a tree-like graph structure where each node contains a question. A path through the tree starts at the root node and goes downward through the tree based on the answer to the questions of all encountered nodes. These questions can be simple yes/no questions resulting in a binary decision tree, or questions with more than two possible answers. A simple example of a binary decision tree is shown in fig. 4.1.

In a phonetic decision tree, the leafs represent classified acoustic data. Usually, one global tree is generated for all data. An alternative is to build a tree for each context independent phoneme. All data belonging to the tree enters the trees at the root node, and based on the answer to the

in-between questions it is allocated to one particular leaf node. For this research, only global decision trees were considered. The nodes of the tree contain questions about the phoneme and its context. Example questions from the ESAT Spraak system are:

- Does the phoneme has stress on it?
- Is the phoneme a fricative?
- Is the phoneme preceded by a plosive?

For an overview of questions present in the ESAT Spraak system, we refer to Duchateau et al. [27].

Tree construction

The tree construction itself has not been considered as a part of this research. However, for completeness, a brief overview of the construction process is described here. The construction of a phonetic decision tree normally occurs in a top-down manner. The initial tree contains only a root node, which in itself is also a leaf. This root node is subdivided into two new nodes by choosing a set of all possible questions. This subdivision process is iteratively performed until a stopping criterion is activated, after which the leaf is no longer subdivided. In order to choose a question for the node, a criterion is needed. In the literature, several criteria have been presented, mostly based on a statistical distribution of the data underlying that particular node such as Gaussian, Gaussian Mixture or Poisson distributions [70]. The question which increases the likelihood of the data most after subdivision is selected.

Application to class-dependent distance measures

These context-dependent phonetic decision trees have been constructed for tying states in HMM-based speech recognition. However, they also subdivide the training data into classes based on the context of the phoneme. Moreover, the stopping criterion of the tree guarantees that enough data per class is available to train the models. As a consequence, by choosing

the right parameters, it can be guaranteed that data is split into classes for which the weight matrices for the distance measure can be trained in a reliable way. These context dependent decision trees provide a nice way to include meta-information from the training database into the definition of the classes for the distance measure.

4.2 Unsupervised clustering methods

Using meta-information for defining classes certainly seems a logical approach to take. However, there are some pitfalls. It is not at all certain that the defined classes are fine-grained enough. It is not always possible to capture information that is present in the data using the available meta-information. Moreover, there could be a mismatch between the models used for splitting the decision tree and the real distribution of the underlying data.

An alternative for using meta-information to define classes of data is the use of unsupervised clustering techniques. Clustering is a form of unsupervised classification, i.e. the data is not labeled with its identity. The task is to group them into *clusters* of similar looking data. Clustering methods can be subdivided in several groups. Jain and Dubes presented a taxonomy to classify clustering methods [39], subdividing clustering techniques into hierarchical and partitional clustering methods. Next to this taxonomy, Jain et al. also presented other classification schemes for clustering methods such as agglomerative vs. divisive, deterministic vs. stochastic, etc. For a summary of these different classification schemes, we refer to the overview presented by Jain et al. [40].

One of the disadvantages of using clustering techniques is that a dissimilarity measure between data is needed. Since the ultimate goal of using these clustering techniques is to create a distance measure, this off course poses a problem. One of the ways to solve this problem and to bootstrap the distance measure training is by using the Euclidean distance. Therefore, if a distance computation is mentioned in the next paragraphs, Euclidean distance is meant unless otherwise stated.

For this work, several existing clustering methods have been studied. In the next sections, we present a brief overview of the considered

```
1: k_means(k)
2: c := initiate_cluster_centers(k)
3: repeat
4:    for all Data points x ∈ X do
5:       setcluster(x,compute_nearest_center(x,c))
6:    end for
7:    c := update_cluster_centers(c,X)
8: until converged
```

Figure 4.2: The k-means clustering algorithm

clustering methods: k-means clustering, single-link clustering and nearest-neighbor clustering.

4.2.1 k-Means Clustering

The k-means clustering method belongs to the class of partitional clustering techniques. I.e. after application of a partitional clustering algorithm, a single partition of the data is obtained. The k-means clustering algorithm works by randomly choosing k cluster centers and then iteratively assigning each prototype to a cluster and recomputing the cluster centers until convergence. Convergence of the k-means clustering algorithm occurs when the cluster centers no longer change and the partition of the data becomes fixed. The main structure of the k-means clustering algorithm is shown in fig. 4.2.

The k-means clustering algorithm provides a simple strategy to partition the data into clusters. One of the disadvantages however is that the number of clusters k needs to be defined beforehand. To overcome this, several variants of the basic algorithm have been presented. One of the best known variants is the ISODATA algorithm presented by Ball et al. [11]. The ISODATA algorithm is able to split and converge clusters when necessary, such that it is not necessary to define k beforehand. The general structure of ISODATA is shown in fig. 4.3.

Another disadvantage of standard k-means clustering is that it has a tendency to split elongated clusters into separate clusters if a Euclidean

1: ISODATA(k, N_{min})
2: Initialize k cluster centers $\mathbf{c}_1, \ldots, \mathbf{c}_k$
3: Put each data point $\mathbf{x}$ in the cluster C_j with the closest center:
$\mathbf{x} \in C_j$ if $||\mathbf{x} - \mathbf{c}_j|| < ||\mathbf{x} - \mathbf{c}_j|| \forall i \neq j$
4: Remove clusters with fewer than N_{min} points in it
Adjust k, Rename clusters $C_1, \ldots, C_k$
5: Update cluster centers:
$\mathbf{c}_j = \frac{1}{N_j} \sum_{\mathbf{x} \in C_j} \mathbf{x}$
If a cluster was deleted: goto 3
6: Compute Δ_j and Δ:
$\Delta_j = \frac{1}{N_j} \sum_{\mathbf{x} \in C_j} ||\mathbf{x} - \mathbf{c}_j||$
$\Delta = \frac{1}{N} \sum_{j=1}^{k} N_j \Delta_j$
7: if $(I > I_{max})$ $L_{min} \leftarrow 0$; goto 9
8: Compute standard deviation vectors $\mathbf{v}_j$:
$v_{ji} = \sqrt{\frac{1}{N_j} \sum_{\mathbf{x} \in C_j} (x_i - z_{ji})^2}, j = 1, \ldots, k; i = 1, \ldots, D$
$v_{j,max} = \max_i v_{ji}$
9: Split clusters if necessary:
if $v_{j,max} > \sigma_{max}$
and $((\Delta_j > \Delta)$ and $(N_j > 2(N_{min} + 1))$ or $k \leq \frac{k_{init}}{2}$:
Split C_j centered around $\mathbf{c}_j$ in the direction of $v_{j,max}$
Goto 3
10: Compute intercluster distances:
$d_{ij} = ||\mathbf{c}_i - \mathbf{c}_j||$
11: Sort d_{ij} in ascending order and
select at most P_{max} pairs with $d_{ij} < L_{min}$
For each pair i, j if S_i nor S_j have been involved in merging this iteration:
merge S_i and S_j and update k
12: $I++$
if $(I < I_{max})$ goto 3

Figure 4.3: The ISODATA clustering algorithm

distance measure is used. An interesting solution for this problem was presented by Mao et al. [54] where the standard algorithm was extended with a regular Mahalanobis distance measure between the data. In this way, hyper-ellipsoidal clusters are created. The regular Mahalanobis distance is defined as follows:

$$d_{RM}(\mathbf{x};\mathbf{y}) = \sqrt{(\mathbf{x}-\mathbf{y})^T[(1-\gamma)(\Sigma_k+\epsilon I)^{-1}+\gamma I](\mathbf{x}-\mathbf{y})}, \qquad (4.1)$$

where Σ_k is the covariance matrix of the cluster to which $\mathbf{y}$ belongs, λ is a scaling factor for choosing between Euclidean and Mahalanobis distance, and ϵ makes sure that the matrix is nonsingular. The parameter λ has been introduced because using a normal Mahalanobis distance for clustering also entails some dangers. This distance sometimes makes big clusters "absorb" smaller clusters in the vicinity [54].

This approach seems particularly useful in our class-dependent distance measure context, because it also contains a scaled distance measure (local Mahalanobis) for each class in the cluster.

4.2.2 Hierarchical clustering

Hierarchical clustering methods are different from partitional algorithms in the sense that they do not provide one single partition of the data into clusters. Instead, hierarchical clustering builds a dendrogram with nested grouping of the data. A dendrogram is a tree diagram that illustrates the partitioning of the data into different clusters. It is hierarchical in the sense that a parent node in tree contains the data that is present in every subtree of the parent node. Most hierarchical methods start from a situation where each data pattern is in its own cluster. The most used methods are the single-link [39] and complete-link [46] clustering algorithms. Both algorithms share the same principle: at each iteration, the *closest* two clusters are merged. The way "closest" is defined however differs. In single-link clustering, the distance between clusters is defined as the minimum distance between every pairwise of data points from both clusters. With the complete-link method, the distance between clusters is defined as the maximum pairwise distance.

Both clustering methods generate very different clusters. Whereas the complete-link algorithm tends to produce tight and compact clusters, the single-link algorithm produces elongated clusters [40].

4.2.3 Other clustering methods

In the literature, lots of clustering methods exist based on a diversity of principles. Research on these clustering methods however is not a part of the research performed for this dissertation. For a nice overview of clustering methods in pattern recognition, we refer to Jain et al. [40, 39].

4.3 Combining supervised and unsupervised clustering methods

So far, we have discussed two different clustering strategies: supervised clustering based on forced alignment and phonetic decision trees, and unsupervised clustering based on partitional and hierarchical clustering algorithms. The first type of clustering methods has the big advantage of incorporating meta-information such as the phoneme identity into the clusters, resulting in linguistically meaningful clusters. Using phonetic decision trees, we are able to build fine-grained clusters of data retaining enough data points per cluster to reliably estimate the weights of the distance measure [2]. On the other hand, it is not entirely guaranteed that these clusters are not overlapping. Moreover, we are not at all certain that clusters are fine-grained enough to be able to model them with a weighted distance measure. For instance, if a cluster generated by a supervised clustering method consists of two clouds of points that are separated, it would be wiser to split that cluster and assign a class-dependent weight matrix on each of these two clouds (clusters).

Unsupervised clustering methods have the advantage to make a partition of the data, i.e. clusters generated by most clustering algorithms are non-overlapping. Moreover, some clustering algorithms like the k-means algorithm with the regular Mahalanobis distance tend to produce

[2] Weight estimation of the distance measures is subject of chapter 5.

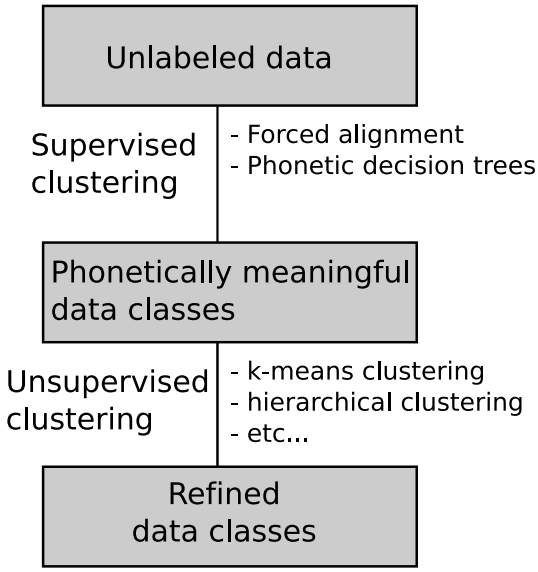


Figure 4.4: Combined clustering

hyper-ellipsoidal clusters. This type of clusters is ideal for the weighted distance measures presented in section 3.3.2. The disadvantage of this type of clustering on the other hand is that these clusters usually do not have any linguistical meaning.

Because both approaches have some very positive aspects for the problem at hand, it seems natural to try to combine both to obtain a clustering method that incorporates advantages of both methods. Naturally, we want to retain some phonetic information in the clusters because this is important information for speech recognition, and as we will see in chapter 5, it will also be helpful in training the class-dependent distance measures. Fig. 4.4 presents a summary of possible alternatives to combine supervised and unsupervised clustering techniques. The different options are discussed in the next paragraphs.

4.3.1 Combination on the context-independent phoneme-level

The first alternative is to perform the supervised clustering based on context-independent phoneme information. In this case, a class is generated for each available context independent phoneme (typically 40–45 classes). The segmentation of the training data is obtained after a forced alignment of the data based on available context independent acoustic models.

In the context of template based speech recognition, there is an intuitive reasoning behind this choice. The template selection algorithm can select phonemes regardless of their context. If such a phoneme with a wrong context is selected and encountered in the recognized phoneme sequence, it nevertheless is correct.

After assigning the context-independent phoneme-based labels to the frames, an unsupervised clustering algorithm is used on the data of each individual class. We have used the ISODATA clustering algorithm to avoid the need for choosing the "correct" number of clusters beforehand.

4.3.2 Combination on the context-dependent phoneme-level

An alternative approach is combining context-dependent phoneme clustering with unsupervised clustering methods. In this case, the first step consists of context-dependent phoneme clustering with phonetic decision trees. The decision trees are built after a forced alignment of the example data with available context-independent acoustic models. After the creation of the decision trees, a forced alignment of the training data based on the acoustic HMM models described with the phonetic decision tree is performed.

Next, the same approach as for the context-independent based partitioning is used. An unsupervised clustering algorithm is used on every class individually. Of course, the classes generated by the phonetic decision tree are already much more fine-grained (and thus smaller). Therefore, a further clustering of the data is in many cases not possible because not enough data is available for some particular classes. In general, we placed a threshold on the minimal size of a class. We experimentally fixed

this threshold to a minimum of 1000 frames per class.

4.3.3 Experimental results

We have performed experiments with both context-independent and context-dependent phoneme based classes. These experiments have shown that classes based on context-dependent phonemes produce superior results [57]. In this research, we observed about 10% relative improvement on word error rate on the Resource Management database.

Chapter 5

Training Distance measures

5.1 Introduction: pattern classification

5.1.1 Description

The automatic classification of patterns is a broad area of research in the machine learning area. The aim of pattern classification is the allocation of a certain input to a specific class in a predefined set of classes. Examples of pattern classification tasks are automatic identification of diseases based on a set of symptoms, optical character recognition, automatic document classification, speech recognition, where an input utterance of speech is mapped onto a sequence of words, ...

Formally, a classifier Ξ is a mapping from a feature space $\mathcal{X}$ into a discrete set of classes $C = \{C_1, C_2, \ldots, C_{N_K}\}$. The feature space usually is a set of features derived from the input pattern. This input pattern can e.g. be a picture, a set of sensor data, a set of symptoms from a patient, an audio signal, a movie, ... In order to process this input pattern into the feature space, a preprocessing (feature extraction) unit is required. The overall classification process is illustrated in fig. 5.1.

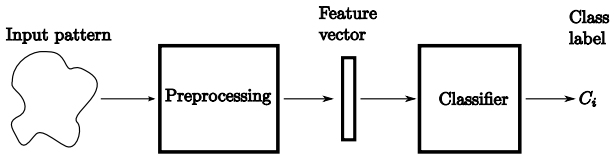


Figure 5.1: A typical classification system

5.1.2 Training a classifier

Training (or learning) the classifier is performed by defining a decision boundary and computing the parameters of this boundary such that it separates the data into categories. Usually, this is performed on a given collection of labeled data, also called the training data. There are several types of learning:

a) **Supervised learning.** In supervised learning, the set of labels (classes) is defined beforehand, and the class of each training example is available in the database.

b) **Unsupervised learning.** In unsupervised learning, there are no predefined labels available. It is the task of the system to discover the structure in the training data by searching for clusters in the input patterns. An example of unsupervised learning are the unsupervised clustering methods presented in § 4.2.

c) **Reinforcement learning.** In reinforcement learning, the correct class label is not given to the training procedure. Instead, the learner only gets feedback for right or wrong classification. It does not say how wrong the decision is.

However, it is not always possible to train the classifier such that it perfectly recognizes each category of patterns. Depending on the type of decision boundaries used, these may not be sufficient to model the data accurately. Also, in many problems, overlap of the classes is unavoidable.

5.1.3 Classification in speech recognition

Speech recognition is also a form of classification. In this case, the input comes from an audio signal that is preprocessed to obtain a sequence of measurements. The output of a recognition system is a phonetic transcription of the uttered speech. The "categories" could be phonemes, words or sentences depending on the task.

In this context, optimizing the classification means minimizing the errors made by the recognition system. In continuous speech recognition, this usually means minimizing the word error rate: the number of wrong (inserted, deleted or substituted) words in the recognizer output.

5.2 Decision theory

5.2.1 The Bayes classifier

Bayesian classification theory builds a statistical method for the problem of pattern classification [28]. The basis of this method is the modelling of the costs that are caused by classification of patterns. Because of the general importance of this theory, we will give a brief overview of Bayesian classification theory in this section.

First, some notations are introduced. We have a classification problem at hand with different classes. We will label these $\omega_1, \ldots, \omega_K$. Note the difference in notation compared to the classes for class-dependent distance measures called $C_1, \ldots, C_{N_K}$. This is introduced because, as we will see in the forthcoming sections, the classes of the class-dependent distance measure are not always the same type of classes as the classes for the classification problem. Using these classes, we suppose a prior probability $P(\omega_k)$ that models the likelihood of encountering a prototype of class ω_k.

Next, we need measurements on the object that has to be classified. We will use $\mathbf{x} = x_1, \ldots, x_D$ to denote these measurements. The probability of the object $\mathbf{x}$ belonging to class ω_k is denoted by $P(\omega_k|\mathbf{x})$. Using Bayes' rule, we can model this probability as follows:

$$P(\omega_k|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_k)P(\omega_k)}{p(\mathbf{x})},$$

or, in words:

$$\text{posterior} = \frac{\text{likelihood} \times \text{prior}}{\text{evidence}}.$$

Based on this formula, a classification decision can easily be defined. According to Bayes decision theory, the optimal classifier is defined by

$$\hat{\omega} = \arg\max_{\omega_k} P(\omega_k|\mathbf{x}). \tag{5.1}$$

5.2.2 Bayesian decision theory

Suppose we have a classification problem with classes $\omega_1, \ldots, \omega_K$. We can define a cost for misclassification. We will denote the cost for misclassifying a prototype of class ω_i as ω_j as $\ell(\omega_j|\omega_i)$. Suppose we classify $\mathbf{x}$ as belonging to ω_k, then the total loss incurred by this decision can be computed as follows:

$$R(\omega_k|\mathbf{x}) = \sum_{j=1}^{K} \ell(\omega_k|\omega_j) P(\omega_j|\mathbf{x}), \tag{5.2}$$

One possible decision rule based on this could then be to classify an observation by minimizing the risk:

$$\hat{\omega} = \arg\min_{k} R(\omega_k|\mathbf{x}). \tag{5.3}$$

If we perform this action for every $\mathbf{x}$, the overall risk will be minimized. The overall risk is defined as follows:

$$R = \int_{\mathcal{X}} R(\Omega(\mathbf{x})|\mathbf{x}) p(\mathbf{x}) d\mathbf{x}. \tag{5.4}$$

$\Omega(\mathbf{x})$ stands for the decision that is made for $\mathbf{x}$ and $\mathcal{X}$ denotes the entire feature space. This overall risk can be rewritten as follows [61]:

$$R = \sum_{k=1}^{K} \int_{X_k} \ell(\Omega(\mathbf{x})|\omega_k) p(\mathbf{x}, \omega_k) 1(\mathbf{x} \in \omega_k) d\mathbf{x}, \tag{5.5}$$

where X_k stands for the part of the observation space that causes misclassification for class ω_k and 1(.) is a function with value 1 if the condition is true, otherwise the value is 0. In this context, the risk of classifying $\mathbf{x}$ as belonging to class ω_j is:

$$R(\omega_j|\mathbf{x}) = 1 - P(\omega_j|\mathbf{x}). \tag{5.6}$$

5.2.3 Discriminant functions

Another, very common method to regard a classification problem is in terms of a set of discriminant functions. In this case, a set of discriminant functions $g_k(\mathbf{x}), i = 1, \ldots, K$ is defined, one for every class ω_i.

The decision rule can be written in terms of these discriminant functions: classify an observation $\mathbf{x}$ as belonging to class ω_{k^*} if its corresponding discriminant $g_k(\mathbf{x})$ has the highest value:

$$k^* = \arg\max_k g_k(\mathbf{x}). \tag{5.7}$$

It is very easy to represent a Bayes classifier in this way. The standard Bayes classifier is obtained if the posterior probability is taken as discriminant:

$$g_k(\mathbf{x}) = P(\omega_k|\mathbf{x}) = \frac{p(\mathbf{x}|\omega_k)P(\omega_k)}{p(\mathbf{x})}. \tag{5.8}$$

It is also possible to use one of the following discriminants to obtain the same result:

$$g_k(\mathbf{x}) = p(\mathbf{x}|\omega_k)P(\omega_k). \tag{5.9}$$

$$g_k(\mathbf{x}) = \log p(\mathbf{x}|\omega_k) + \log P(\omega_k). \tag{5.10}$$

The different discriminants of equations (5.8), (5.9) and (5.10) all result in the same classification result, however, some might be easier to compute than others [37].

It is also possible to express the risk-based Bayes classifier in terms of discriminant functions. The risk-based Bayes classifier is obtained in the following way:

$$g_k(\mathbf{x}) = -R(\omega_k|\mathbf{x}) = -\sum_{j=1}^{K} \ell(\omega_k|\omega_j)P(\omega_j|\mathbf{x}). \tag{5.11}$$

5.2.4 The ideal classifier

The Bayes classifier is often referred to as the ideal Bayes classifier, because it describes the optimal partitioning of space among the different classes. This is the ultimate goal of any classifier.

However, implementing this rule is not as easy as it seems. It supposes that the posterior probabilities of the categories are known. However, in most pattern classification problems, these probabilities are unknown. Many classification methods present strategies to estimate these probabilities, after which Bayes rule can be applied. There exist a lot of different strategies for classification design. There are two big choices to be made when constructing a classifier. The first choice is which likelihood function to use. This is discussed in § 5.3. The second choice is which strategy to use to estimate the parameters of the likelihood function. An overview of parameter estimation strategies is presented in § 5.4.

5.3 Parametric vs. non parametric

One important choice to make in classifier design is the choice of the likelihood functions for the conditional densities $p(\mathbf{x}|\omega_k)$ and the prior probabilities $P(\omega_k)$.

5.3.1 Parametric likelihood functions

Parametric distribution functions are the most common approach to model the conditional likelihood functions in Bayesian classifier design. In parametric methods, a parametric density distribution has to be chosen. The best choice of distribution is a distribution that fits the data perfectly (i.e. the real probability distribution). However, this probability distribution is in general unknown. Therefore, a probability distribution function has to be chosen by the classifier designer. By far the most popular choice is to use the normal (or Gaussian) distribution.

Normal distribution

The multivariate normal density is defined as follows:

$$p(\mathbf{x}) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu})^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu})\right), \tag{5.12}$$

where $\boldsymbol{\mu}$ is the mean vector:

$$\boldsymbol{\mu} = E[\mathbf{x}] = \int_{\mathcal{X}} \mathbf{x}p(\mathbf{x})d\mathbf{x}, \tag{5.13}$$

and Σ is the covariance matrix:

$$\Sigma = E\left[(\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T\right] = \int_{\mathcal{X}} (\mathbf{x}-\boldsymbol{\mu})(\mathbf{x}-\boldsymbol{\mu})^T p(\mathbf{x})d\mathbf{x}. \tag{5.14}$$

The reason why this distribution is popular is because of the central limit theorem. This theorem states that the sum of a sufficiently large number of independent random variables will be approximately normally distributed.

Discriminants for normal distribution

In practice, usually the log-likelihood of normal distributions is used for likelihood computations, as it is mathematically equivalent and easier to compute. In the case of Bayesian classification, the discriminant using the form of equation (5.10)looks as follows:

$$g_k(\mathbf{x}) = \log p(\omega_k|\mathbf{x}) = \log p(\mathbf{x}|\omega_k) + \log P(\omega_k). \tag{5.15}$$

If the density $p(\mathbf{x}|\omega_k)$ has a normal distribution, combining equations (5.12) and (5.15) leads to the following discriminant:

$$g_k(\mathbf{x}) = -\frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\Sigma_k^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) - \frac{D}{2}\ln 2\pi - \frac{1}{2}\ln|\Sigma_k| + \ln P(\omega_k), \tag{5.16}$$

where $\boldsymbol{\mu}_k$ is the mean of the observations in class ω_k. Note that the term in D can be dropped, since it is independent of i and thus plays no role in taking decisions using this discriminant.

5.3.2 Non-parametric likelihood functions

It is not always easy to choose a parametric distribution function. Therefore, non-parametric density estimation techniques have been proposed. In these cases, the density is implicitly based on the data itself, rather than on a probability distribution.

One example of the application of non-parametric techniques is in nearest neighbor classification. In this form of classification, the decision on the class of a new observation is based on the labels of the neighboring prototypes in feature space. The "local" density is thus implicitly constructed by looking at the neighborhood of the observation. In this section, we will present an overview of some important aspects of non-parametric density estimation. Unless otherwise referenced, most of the theory presented here is based on Duda et al. [28].

Non-parametric density estimation

Suppose we have a vector $\mathbf{x}$ and we have N samples $X = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N\}$ drawn independently and identically distributed according to an unknown probability law $p(\mathbf{x})$. Non-parametric density estimation works by building a series of regions $\mathcal{R}_1, \mathcal{R}_2, \ldots$ containing this example $\mathbf{x}$, and created using more and more samples ($\mathcal{R}_1$ with one sample of X , $\mathcal{R}_2$ with two samples of X etc. . .). If we define V_N as the volume of $\mathcal{R}_N$ and $k_N(\mathbf{x})$ the number of samples in $\mathcal{R}_N$, the density estimate $p_N(\mathbf{x})$ is given by:

$$p_N(\mathbf{x}) = \frac{k_N(\mathbf{x})/N}{V_N}. \tag{5.17}$$

The remaining problem is how to choose the region $R_i, i = 1, \ldots, N$. Several options are discussed in the next sections.

Parzen windows

In Parzen window density estimation (also called kernel density estimation), the region in which the samples fall $\mathcal{R}_N$ is assumed to be a D-dimensional hypercube with edge length h_N. The Parzen theory also

defines a window function ϕ:

$$\phi(\mathbf{u}) = \begin{cases} 1 & |u_j| \leq 1/2; j = 1, \ldots, D \\ 0 & \text{otherwise.} \end{cases} \tag{5.18}$$

In fact $\phi(\mathbf{u})$ is a unit hypercube centered at the origin. This window function is used to estimate the density around $\mathbf{x}$, it is translated to be centered around $\mathbf{x}$ and transformed to have edge length h_N. Using this window, the number k_N of samples $\mathbf{x}_i$ that fall within R_N around $\mathbf{x}$ can be defined as:

$$k_N(\mathbf{x}) = \sum_{i=1}^{N} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_N}\right). \tag{5.19}$$

If equations (5.17) and (5.19) are combined, the Parzen density estimator is obtained:

$$p_N(\mathbf{x}) = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{V_N} \phi\left(\frac{\mathbf{x} - \mathbf{x}_i}{h_N}\right). \tag{5.20}$$

A natural extension to this approach is replacing the hypercube window with any window $K(\mathbf{x})$. Any window that ensures $p_N(\mathbf{x})$ is a density is a valid window. Such windows can be constructed when keeping $V_N = h_N^D$ (i.e. keeping the volume of the window identical to that of the hypercube) and ensuring that K satisfies the following criteria:

$$K(\mathbf{u}) \geq 0 \tag{5.21}$$

and

$$\int_{\mathcal{X}} K(\mathbf{u}) d\mathbf{x} = 1. \tag{5.22}$$

A window satisfying these criteria is also called a "kernel".

It is well known that it is difficult to choose the sequence of regions $\mathcal{R}_1, \mathcal{R}_2, \ldots$ with corresponding volumes $V_1, V_2, \ldots$ [28]. The choice of the kernel is a difficult problem in kernel density estimation. Several window functions have been proposed. A detailed discussion of these different types of windows is beyond the scope of this dissertation. For more information on this topic, the reader is referred to [88, 28].

k-nearest neighbor density estimation

It is not always possible to define a window function that works best for a given set of data. In these cases, it could be better to let the window be dependent on the data itself rather than on an arbitrary function. One method to do this is by constructing a window around $\mathbf{x}$ and letting it grow until a predefined number of samples k_N are included in this window. Naturally, these k_N samples are also the k_N-nearest neighbors (k_N-NN) of $\mathbf{x}$. Here we can take the following distribution:

$$p_N(\mathbf{x}) = \frac{k_N/N}{V_N}, \tag{5.23}$$

with V_n the volume of the window. It can be proven that if $\lim_{N\to\infty} k_N = \infty$ and $\lim_{N\to\infty} k_N/N = 0$, then $\lim_{N\to\infty} p_N(\mathbf{x}) = p(\mathbf{x})$ [28]. The size V_N is equal to the volume of the unit sphere centered around $\mathbf{x}$ and with radius the distance to the k_N-th nearest neighbor of $\mathbf{x}$.

It is possible to link this with the theory of kernel density estimation by defining a generalization of the nearest neighbor density of equation (5.23). This generalization is defined as follows:

$$p_N(\mathbf{x}) = \frac{1}{N}\frac{1}{r_{k_N}(\mathbf{x})^D}\sum_{i=1}^{N} K\left(\frac{\mathbf{x}-\mathbf{x}_i}{r_{k_N}(\mathbf{x})}\right), \tag{5.24}$$

where $r_{k_N}(\mathbf{x})$ is the distance to the k_N-th nearest neighbor of $\mathbf{x}$. With a window corresponding to the D-dimensional unit hypersphere, the simple distribution of equation (5.23) is obtained. I.e.

$$K(\mathbf{x}) = \begin{cases} c_D^{-1} & \text{if } |\mathbf{x}| \leq 1 \\ 0 & \text{otherwise,} \end{cases} \tag{5.25}$$

with c_D being the volume of the D-dimensional hypersphere:

$$c_d = \begin{cases} \frac{\pi^k}{k!} & \text{for } D = 2k \text{ even} \\ \frac{2^k \pi^{k-1}}{D!!} & \text{for } D = 2k-1 \text{ odd.} \end{cases} \tag{5.26}$$

However, in practice, this nearest neighbor density estimator is not very useful. It is very sensitive to noise in the data and emphasizes the

"tails" of the distribution too much. Solutions to this problem have been proposed by using adaptive kernels.

The reason for explaining this k-nearest neighbors density estimation is because it can be used for pattern classification using a k-NN rule. In order not to overload this chapter, we present the theory of k-NN computation and classification in chapter 6.

One important note to point out is that a nearest neighbor rule leaves much room for interpretation. It is for instance undefined which distance measure to use for computing the nearest neighbors. Traditional nearest neighbor estimators almost always use the Euclidean distance. However, as we have explained in chapter 3, a Euclidean distance measure is in most cases suboptimal. We have introduced the local class-dependent distance measures based on a set of weight matrices Λ. Until now however, we have not yet defined how to estimate these weights. We will discuss some different strategies to find these weights in the next section, which is concerned with parameter optimization.

Other kernel functions

There are many possible functions that are useful as kernel for Parzen density estimation. A popular choice is the Gaussian kernel function:

$$K(\mathbf{x}, \Sigma) = \mathcal{N}(\mathbf{x}, \mathbf{0}, \Sigma) = \frac{1}{\sqrt{(2\pi)^D |\Sigma|}} \exp(-\frac{1}{2}\mathbf{x}'\Sigma^{-1}\mathbf{x}). \tag{5.27}$$

5.4 Parameter estimation

In general, parameter estimation is concerned with finding the optimal parameters for the likelihood function. When parametric distributions are used, the parameters are based on the parameters of the distribution itself. For instance: a normal distribution is completely defined if the mean $\boldsymbol{\mu}$ and covariance matrix Σ are known. These parameters can be estimated by taking the sample mean $\overline{\mathbf{x}}$ and sample covariance matrix S.

With non-parametric methods however, the meaning of parameters is different. One possible set of parameters comes from the choice of the kernel function. In our case, we have a second set of parameters that need

to be defined: the weights of the local distance measure Λ. It is logical to insert also these parameters in the set of optimization parameters for the classifier.

In general, there are two very different approaches to parameter estimation. The first approach is by using only data of the class itself to compute the likelihood parameters and to optimize the likelihood of the data. This is called **maximum likelihood parameter estimation**. It is discussed in § 5.4.1. The other alternative is to discriminate the data in the different classes as much as possible. These methods are called **discriminative parameter estimation** and they are discussed in § 5.4.2.

5.4.1 Maximum likelihood parameter estimation

Suppose a likelihood function has been chosen for the data and this likelihood function has a parameter set, which we will call θ. We have a likelihood function $p(\mathbf{x}|\omega_k)$ which is distributed according to this likelihood. The parameters of these distribution functions are different for each class ω_k. We will denote the parameters belonging to the distribution of class k by θ_k.

Furthermore, maximum likelihood (ML) estimation assumes that samples of class ω_i contain no information about θ_j if $i \neq j$. That means it assumes that the parameters of the different classes are independent of each other.

Suppose that we have a set of training samples Y containing N samples $\mathbf{x}_1, \ldots, \mathbf{x}_N$. We have

$$P(Y|\theta) = \prod_{i=1}^{N} p(\mathbf{x}_i|\theta). \tag{5.28}$$

This likelihood $P(Y|\theta)$ is the probability that the training vectors Y where drawn from a probability distribution based on θ. As usual, the log-likelihood is taken instead of the real likelihood. The maximum likelihood estimate $\hat{\theta}$ is then the parameter set that maximizes this log-likelihood function:

$$\hat{\theta} = \arg\max_{\theta} \log P(Y|\theta). \tag{5.29}$$

This log-likelihood can be maximized by setting the gradient to zero:

$$\nabla_\theta \log P(Y|\theta) = \sum_{i=1}^{N} \nabla_\theta \log p(\mathbf{x}_i|\theta) = 0. \tag{5.30}$$

Of course, these zero-gradient points can be maxima, minima or inflection points.

An extension of ML estimation: the maximum a posteriori (MAP) estimator also takes into account the prior probabilities, i.e. $\log(P(Y|\theta)P(\theta))$. If $P(\theta)$ has a uniform distribution, MAP estimation is equal to ML estimation.

ML estimation for a Gaussian distribution

If ML estimation is applied to a Gaussian likelihood, the sample mean and variances are found as global optimum of the likelihood. I.e.

$$\hat{\boldsymbol{\mu}} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i, \tag{5.31}$$

and

$$\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{x}_i - \hat{\boldsymbol{\mu}})(\mathbf{x}_i - \hat{\boldsymbol{\mu}})^T. \tag{5.32}$$

ML in HMM based speech recognition

We have introduced HMM-based speech recognition in chapter 1. For some details about ML-training of hidden Markov models, the reader is referred to § 1.4.

Disadvantages of ML parameter estimation

In the optimal case, the likelihood functions correspond to the exact data distribution and enough training data is available. Only in this case, ML parameter estimation will lead to the optimal classifier. However, in practice, both of these assumptions are not fulfilled [67].

Optimality of the solution. Maximum likelihood estimation (MLE) finds the optimal parameters based on the underlying model. Naturally, this optimality is only valid if the underlying model is correct. As we have already argued in § 1.6.1, the HMM model is a simplification of the true nature of speech and thus these models are not correct (or at least not complete).

Availability of training data. The second condition for the optimality of MLE is that an unlimited amount of training data is available. In practice, a sufficient amount of training data has to be available. In speech recognition however, some classes do not have the necessary amount of training data available to reliably estimate these parameters.

Corrective training

In an attempt to improve upon standard ML estimation with an incorrect model, corrective training has been proposed. In corrective training, a classification of the training data is made with the parameters estimated by ML training. After this classification, the observations that are classified in a wrong class are "corrected", i.e. the model parameters are adjusted to improve their likelihood in the correct class. Corrective training can also be seen as a special case of discriminative training. It is explained in § 5.4.4.

5.4.2 Discriminative training

Discriminative training is an attempt to overcome the limitations of standard MLE. Instead of only looking at the parameters of the correct class for each training example, it also takes into account the other competing classes. Discriminative training tries to make the discrimination of a pattern between its correct class and the competing classes as high as possible.

It is easy to see that in order to apply Bayes decision rule, the perfect knowledge of $P(\omega_k|\mathbf{x})$ is sufficient, it is however not a necessary condition. All we need to know is for which class ω_k the likelihood $P(\omega_k|\mathbf{x})$ is the highest. Therefore, posterior, likelihood and prior need not be modelled

explicitly. Discriminative training aims to do this by finding the optimal decision boundaries.

5.4.3 Maximum Mutual Information

One alternative strategy for MLE is the maximum mutual information (MMI) theory [9, 69, 68]. MMI has been specifically designed to train the model parameters of problems that are modelled by a possibly incorrect model. The basic idea is to find the set of parameters Λ that minimize the conditional entropy $H_\Lambda(\Omega|X)$. Another viewpoint is that MMI finds a parameter set Λ that provides as much information as possible about Ω given the input X. MMI was successfully applied in HMM-based speech recognition [69].

The MMI parameter estimation strategy finds its source in information theory [17]. We will introduce this section by defining some important concepts from information theory.

Information theory

In information theory, the **entropy** of a discrete random variable Ω is defined as:

$$H(\Omega) = -\sum_{k=1}^{K} P(\Omega = \omega_k) \log P(\Omega = \omega_k), \tag{5.33}$$

where $P(\Omega = \omega_k)$ denotes the probability that the class is ω_k. A possible interpretation of the entropy of a random variable is the amount of uncertainty that is present in this random variable. If the log is a 2-log, the entropy is measured in bits. Another definition of entropy is given by the expected value:

$$H(\Omega) = E\left[\log \frac{1}{P(\Omega)}\right]. \tag{5.34}$$

The **joint entropy** $H(\Omega, X)$ is defined as follows:

$$\begin{aligned} H(\Omega, X) &= -E \log P(\Omega, X) \\ &= -\sum_{k=1}^{K}\sum_{i=1}^{N} P(\Omega = \omega_k, X = \mathbf{x}_i) \log P(\Omega = \omega_k, X = \mathbf{x}_i). \end{aligned} \tag{5.35}$$

The **conditional entropy** $H(\Omega|X)$ is defined as follows:

$$H(\Omega|X) = -\sum_{k=1}^{K}\sum_{i=1}^{N} P(\Omega = \omega_k, X = \mathbf{x}_i) \log P(\Omega = \omega_k | X = \mathbf{x}_i). \quad (5.36)$$

The final information theoretic concept we have to introduce is the **mutual information** $I(\Omega; X)$, which is defined as follows:

$$I(\Omega; X) = H(\Omega) - H(\Omega|X). \quad (5.37)$$

Using equations (5.33) and (5.36), this can be worked out as follows:

$$\begin{aligned} I(\Omega; X) &= H(\Omega) - H(\Omega|X) \\ &= \sum_{k,i} p(\omega_k, \mathbf{x}_i) \log P(\omega_k|\mathbf{x}_i) - \sum_{k} P(\omega_k) \log P(\omega_k) \\ &= \sum_{k,i} p(\omega_k, \mathbf{x}_i) \log P(\omega_k|\mathbf{x}_i) - \sum_{k,i} P(\omega_k|\mathbf{x}_i) p(\mathbf{x}_i) \log P(\omega_k) \\ &= \sum_{k,i} P(\omega_k, \mathbf{x}_i) \log \frac{P(\omega_k|\mathbf{x}_i)}{P(\omega_k) p(\mathbf{x}_i)}. \end{aligned} \quad (5.38)$$

Model-based entropy

As usual, in many problems the real conditional likelihood of the posterior $P(\omega_k|\mathbf{x}_i)$ and prior $P(\omega_k)$ is unknown. They are replaced by a model with parameter set Λ. If we replace the likelihood in the entropy with the model-based likelihood, we obtain the **model-based entropy**. The conditional entropy now becomes:

$$H_\Lambda(\Omega|X) = -\sum_{k=1}^{K}\sum_{i=1}^{N} p(\omega_k, \mathbf{x}_i) \log P_\Lambda(\omega_k|\mathbf{x}_i), \quad (5.39)$$

where P_Λ is the model-based likelihood function.

Brown et al. proved that the real entropy is a lower bound for any model-based entropy [15], i.e.

$$H_\Lambda(\Omega|X) \geq H(\Omega|X). \quad (5.40)$$

Using the model-based conditional entropy, the model-based mutual information now becomes:

$$I_\Lambda(\Omega; X) = \sum_{k,i} p(\omega_k, \mathbf{x}_i) \log \frac{P_\Lambda(\omega_k|\mathbf{x}_i)}{P_\Lambda(\omega_k) p_\Lambda(\mathbf{x}_i)}. \tag{5.41}$$

The criterion

The goal in MMI training is to minimize the conditional entropy. According to equation (5.37), this can be written as follows:

$$H(\Omega|X) = H(\Omega) - I(\Omega; X). \tag{5.42}$$

Since the real joint likelihood $p(\omega_k, \mathbf{x}_i)$ is unknown, we have to assume the sample $(\omega, \mathbf{x})$ of labeled data we have is representative. We will denote the estimation of this joint likelihood based on the parameter set Λ as $p_\Lambda(\omega_k, \mathbf{x}_i)$. We now have to choose the parameter set Λ such that it maximizes the following criterion:

$$\begin{aligned} J_{\text{MMIE}}(\Lambda) &= \sum_{k,i} \log \left(\frac{p_\Lambda(\omega_k, \mathbf{x}_i)}{P_\Lambda(\omega_k) p_\Lambda(\mathbf{x}_i)} \right) \\ &= \sum_{k,i} \left(\log p_\Lambda(\mathbf{x}_i|\omega_k) - \log p_\Lambda(\mathbf{x_i}) \right). \end{aligned} \tag{5.43}$$

Note that if only the first term of this criterion is taken into account, the ML estimate is obtained. MLE and MMI differ in the second term which is absent in MLE. In order to fully understand the difference between MLE and MMI estimation, we have to look at the derivatives of both criteria. Using some straightforward computations, the derivative of J_{MMIE} looks as follows:

$$\frac{\partial J_{\text{MMIE}}(\Lambda)}{\partial \lambda_i} = Q_\Lambda(\mathbf{x}|\omega) \left[\frac{1}{p_\Lambda(\mathbf{x}|\omega)} - \frac{P_\Lambda(\omega)}{p_\Lambda(\mathbf{x})} \right] - \sum_{\hat{\omega} \neq \omega} \frac{Q_\Lambda(\mathbf{x}|\hat{\omega}) P_\Lambda(\omega)}{p_\Lambda(\mathbf{x})}. \tag{5.44}$$

A closer look at this formula leads to the finding that the first term is in the direction of $Q_\Lambda(\mathbf{x}|\omega)$, which is exactly the same direction as the MLE

derivative. The difference is the second term which is subtracted: it has a term in the direction of $Q_\Lambda(\mathbf{x}|\hat{\omega})$ for every wrong class $\hat{\omega}$. In other words, MMIE tries to increase the likelihood of the correct class and decreasing the probability of the wrong classes simultaneously. The weight by which the increase or decrease is processed is given by the prior probabilities $P_\Lambda(\omega)$.

MMIE and HMM-based speech recognition

Traditionally, MMIE uses gradient descent to optimize the parameters Λ. However, many, more efficient, re-estimation techniques have been created. Most of these procedures are based on re-estimation formulas derived from gradient descent on the HMM parameters [69].

5.4.4 Minimum Classification Error

The theory of Minimum Classification Error (MCE) is another discriminative approach to the training of classifiers. It has already been applied to many different types of classifiers, such as multi-layer perceptrons and hidden Markov models [44, 43, 16, 61].

The goal in minimum classification error is to directly minimize the number of errors made by the classifier. However, the criterion of the classification error is discontinuous by nature: a classification decision is either correct or wrong, there is no in-between. This causes discontinuities in the target function. To circumvent this problem, MCE makes use of a continuous loss function. In the following sections, we describe MCE, which is based on discriminant functions to obtain a misclassification measure.

Discriminant function

In § 5.2.3, we have defined a classifier in terms of a discriminant function. The general principle of minimum classification error relies on this theory of discriminants. The basic principle of MCE is the definition of a discriminant function $g_k(\mathbf{x}, \Lambda)$ which is a measure for how much the observation $\mathbf{x}$ belongs to the class ω_k. As explained in § 5.2.3, the classifier will choose

the class ω_{k^*} with the best discriminant:

$$k^* = \arg\max_k g_k(\mathbf{x}, \Lambda). \tag{5.45}$$

Depending on the classifier at hand, this discriminant has different forms. In a multi-layer perceptron for instance, the discriminant could be the output of the MLP for the given input. In vector quantization the discriminant could be modelled with the distance to a set of reference prototypes. In nearest neighbor classification, the discriminant might be the distance to the k-th nearest neighbor.

The misclassification measure

The goal of MCE is the minimization of the number of classification errors made by the classifier. The decision to which class an observation $\mathbf{x}$ belongs is made by the decision rule of equation (5.45) and is dependent on the input pattern $\mathbf{x}$ itself and the parameter set Λ.

Based on these discriminants, it is possible to define a measure for misclassification. Amari et al. [1] suggested the following measure:

$$d_k(\mathbf{x}, \Lambda) = \frac{1}{K} \sum_{j \in \mathcal{N}_k} \left(g_j(\mathbf{x}, \Lambda) - g_k(\mathbf{x}, \Lambda) \right), \tag{5.46}$$

with $\mathcal{N}_k$ the set of classes for which the discriminant is better than the one of the correct class. The main disadvantage of this measure is that it does not measure the correctness of the decision. If classification is correct, the value of the measure is 0. There is no gradation in the correctness of classification.

To overcome this, McDermott et al. have proposed a new measure [61]. It is defined as follows:

$$d_k(\mathbf{x}, \Lambda) = -g_k(\mathbf{x}, \Lambda) + \left[\frac{1}{K-1} \sum_{j \neq k} g_j(\mathbf{x}, \Lambda)^\eta \right]^{\frac{1}{\eta}}, \tag{5.47}$$

with η a constant which determines how much the competing classes $\omega_j, j \neq k$ are taken into account. If $\eta = 1$, every competing class has

the same weight. If $\eta \to \infty$ however, this equation degrades to the difference in score between the correct category and the best matching incorrect category:

$$d_k(\mathbf{x}, \Lambda) = -g_k(\mathbf{x}, \Lambda) + g_j(\mathbf{x}, \Lambda), \tag{5.48}$$

with ω_j the most competing class (the class with the highest discriminant). This approach has a specific name: **corrective training**. The correctness of classification is visible in the sign of the measure. If it is lower than zero, classification is correct. If it is above zero, classification is incorrect. This measure also measures the correctness (or incorrectness) of classification, which is reflected in the difference between zero and the value of the measure. This misclassification measure will be used in the further discussion.

The loss function

Next in the theory of MCE is the definition of a loss function. The loss function should reflect the goal of the training procedure. In the case of pattern classification, the loss function should measure the classification error. The ideal loss function would be the classification error itself:

$$\ell_k(\mathbf{x}, \Lambda) = \begin{cases} 0 & \text{if } d_k(\mathbf{x}, \Lambda) < 0 \\ 1 & \text{if } d_k(\mathbf{x}, \Lambda) \geq 0. \end{cases} \tag{5.49}$$

This form of loss function however is not very useful for parameter optimization. Most parameter optimization processes use gradient information and the gradient of this step-function is either zero or undefined, which makes it not useful for gradient-based optimization.

Therefore, other loss functions have been proposed in the literature. Some of the most popular loss functions are the **sigmoid** loss function with slope ξ:

$$\ell_k(\mathbf{x}, \Lambda) = \frac{1}{1 + e^{-\xi d_k(\mathbf{x}, \Lambda)}}, \tag{5.50}$$

and the **piecewise linear** loss function:

$$\ell_k(\mathbf{x}, \Lambda) = \begin{cases} 0 & \text{if } d_k(\mathbf{x}, \Lambda) < d_1 \\ (d - d_1)/(d_2 - d_1) & \text{if } d_1 \leq d_k(\mathbf{x}, \Lambda) < d_2 \\ 1 & \text{if } d_k(\mathbf{x}, \Lambda) \geq d_2. \end{cases} \tag{5.51}$$

These loss functions are illustrated in fig. 5.2.

Using these measures and functions, the criterion for MCE can be constructed. In MCE, the total classification error should be minimized, i.e. the classification error for all available training observations. Since the classification error for one single observation is measured by the local loss function, the sum of this local loss over all training examples measures the total classification error. This is called the **total loss** function $L(\Lambda)$, and is at the same time the MCE criterion:

$$J_{\text{MCE}}(\Lambda) = L(\Lambda) = \sum_{k=1}^{K} \sum_{i=1}^{N_k} \ell_k(\mathbf{x}_{k,i}, \Lambda). \tag{5.52}$$

MCE and Bayes decision theory

McDermott et al. presented the link between MCE and Bayes decision theory [61]. The most important aspects of this analysis are presented in this section.

In Bayes decision theory, the probability of making an error can be written as follows:

$$P(\text{error}) = \sum_{k=1}^{K} \int_{\mathcal{X}} p(\mathbf{x}, \omega_k) 1(\mathbf{x} \in \omega_k) 1(P(\omega_k|\mathbf{x}) \neq \max_i P(\omega_i|\mathbf{x})) d\mathbf{x}, \tag{5.53}$$

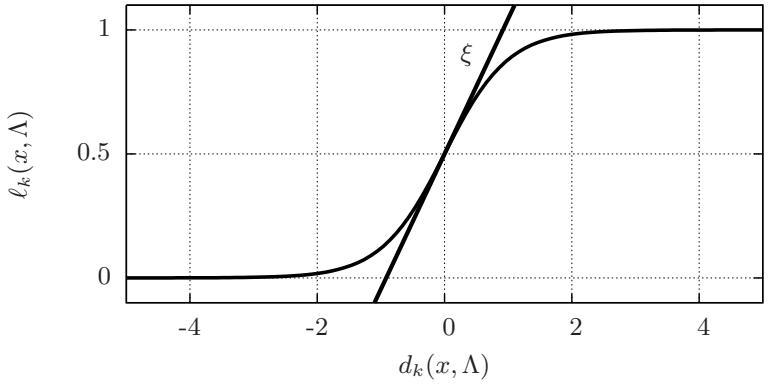
where $1(.)$ is a function with value 1 if the condition is true, otherwise the value is 0.

The MCE classification error, defined in terms of the discriminant can be expressed as follows:

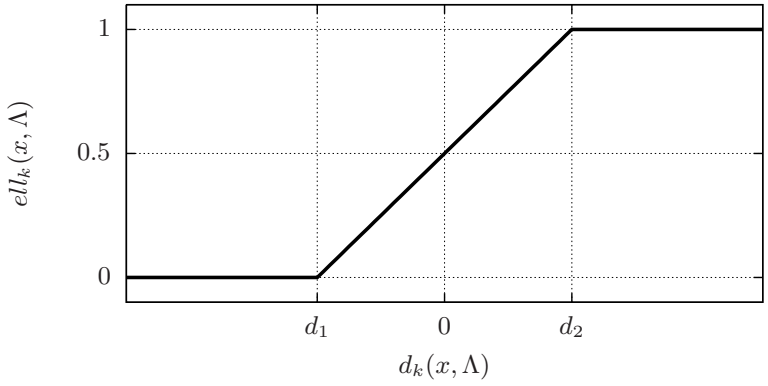
$$P_\Lambda(\text{error}) = \sum_{k=1}^{K} \int_{\mathcal{X}_k(\Lambda)} p(\mathbf{x}, \omega_k) 1(\mathbf{x} \in \omega_k) d\mathbf{x}, \tag{5.54}$$

where

$$\mathcal{X}_k(\Lambda) = \left\{ \mathbf{x} \in \mathcal{X} | g_k(\mathbf{x}, \Lambda) \neq \max_i g_i(\mathbf{x}, \Lambda) \right\}. \tag{5.55}$$



(a) Sigmoid



(b) Piecewise linear

Figure 5.2: Some loss functions

This can be transformed in the following way:

$$\begin{aligned} P_\Lambda(\text{error}) &= \sum_{k=1}^{K} \int_{\mathcal{X}} p(\mathbf{x}, \omega_k) 1(\mathbf{x} \in \omega_k) 1(g_k(\mathbf{x}, \Lambda) \neq \max_i g_i(\mathbf{x}, \Lambda)) d\mathbf{x} \\ &\approx \sum_{k=1}^{K} \int_{\mathcal{X}} p(\mathbf{x}, \omega_k) 1(\mathbf{x} \in \omega_k) \ell_k(\mathbf{x}, \Lambda) d\mathbf{x}. \end{aligned} \tag{5.56}$$

The second expression in equation (5.56) is only an approximation because the MCE loss usually is a continuous 0-1 function, which is only an approximation of the real classification error.

Comparing equations (5.53) and (5.56), we can see that the Bayes decision and the MCE decision are identical if

$$\forall k, \mathbf{x}; 1(g_k(\mathbf{x}, \Lambda) \neq \max_i g_i(\mathbf{x}, \Lambda)) = 1(P(\omega_k|\mathbf{x}) \neq \max_i p(\omega_i, \mathbf{x})). \tag{5.57}$$

5.4.5 Margin-based estimation strategies

Recently tendency towards margin-based discriminative training strategies is seen in the context of HMM-based speech recognition. The idea of a margin is borrowed from the field of support vector machines, where a correct decision is still possible if the sample falls within a margin around the decision boundary.

Inspired by this idea, large margin estimation has been proposed [42]. In large margin estimation the parameters of the model are only updated if a sample is accurately classified. Misclassified samples are ignored.

More recently, soft margin estimation (SME) has been introduced as a discriminative technique for training HMM model parameters [52, 51]. It has been observed that also misclassified samples are very useful for training the classifier. The main idea behind SME is the introduction of a soft margin. When a sample falls within this margin, misclassification is tolerated.

In fact, SME is an extension of the other discriminative criteria. It

can be formulated as follows:

$$J_{\text{SME}}(\Lambda) = \frac{\lambda}{\rho} + \frac{1}{N}\sum_{k=1}^{K}\sum_{i=1}^{N_k}\ell_k(\mathbf{x}_{k,i},\Lambda), \tag{5.58}$$

where ρ is the soft margin and λ is a parameter to adapt the influence the penalty for the empirical risk. The traditional loss function is replaced by a function that incorporates the margin ρ as follows:

$$\ell_k(\mathbf{x},\Lambda) = \begin{cases} \rho - d_k((x),\Lambda), & \text{if } \rho - d_k((x),\Lambda) > 0 \\ \alpha & \text{otherwise,} \end{cases} \tag{5.59}$$

where $d_k((x),\Lambda)$ is a misclassification measure analogous to the ones introduced for MCE. The main idea is that also samples that fall within the margin are considered for updating the parameters.

SME training has however not been considered for our research. As mentioned in the conclusions in chapter 8, it would be a valid extension of our work.

5.4.6 Minimum Phone Error

The minimum phone error (MPE) is a discriminative training criterion that was specifically designed in the context of speech recognition [77, 78].

The minimum phone error criterion is usually defined in terms of sentences s. The reference sentence corresponding to observation $\mathbf{x}_i$ is denoted by s_i. The criterion is defined as follows:

$$J_{\text{MPE}}(\Lambda) = \sum_{i=1}^{N}\sum_{s} P_\Lambda^\alpha(s|\mathbf{x}_i)A(s,s_i), \tag{5.60}$$

where $A(s,s_i)$ is the phone accuracy of sentence s compared to the reference sentence $\mathbf{s}_i$ and $P_\Lambda^\alpha(s|\mathbf{x}_i)$ is a scaled posterior probability of the sentence s. The scaling factor is usually chosen in function of the language model scale [78]. Expanding equation (5.60) using Bayes rule leads to the following MPE criterion:

$$J_{\text{MPE}}(\Lambda) = \sum_{i=1}^{N}\frac{\sum_s p_\Lambda^\alpha(\mathbf{x}_i|s)P^\alpha(s)A(s,s_i)}{\sum_u p_\Lambda^\alpha(\mathbf{x}_i|u)P^\alpha(u)}. \tag{5.61}$$

5.4.7 Unified view

Schlüter et al. created a unifying view on MMI, MCE and some other discriminative training criteria [87]. This unifying view shows the link between these discriminative criteria. Despite the different approach, they only differ on a small number of aspects.

The unified criterion for discriminative training is defined as follows:

$$J_{\mathrm{UN}}(\Lambda) = \sum_{\mathbf{x}} f\left(\log \frac{P^{\alpha}(\omega_{\mathbf{x}}) p_{\Lambda}^{\alpha}(\mathbf{x}|\omega_{\mathbf{x}})}{\sum_{k \in \mathcal{M}_{\mathbf{x}}} P^{\alpha}(\omega_k) p_{\Lambda}^{\alpha}(\mathbf{x}|\omega_k)}\right), \tag{5.62}$$

where $\mathcal{M}_{\mathbf{x}}$ is the set of alternative classes that is considered for observation $\mathbf{x}$.

Using this unifying view, the MMI criterion is obtained for $\alpha = 1$ and $f(z) = z$. The MCE criterion with sigmoid loss can be obtained by setting $f(z) = -1/(1+\exp(\xi z))$. It is even possible to represent the ML criterion in this way by setting $\mathcal{M}_{\mathbf{x}} = \emptyset$ and $f(z) = z$. A more extensive overview of discriminative training criteria that can be represented in this way is presented in Macherey et al. [53].

5.4.8 Discriminative training and speech recognition

Most of the discriminative training criteria were developed in the context of HMM-based speech recognition. MCE for instance was first defined for isolated word recognition [16] and was later expanded for continuous speech recognition. In continuous speech recognition, classification is usually performed on sentences. The standard discriminant for MCE training is the log posterior probability. The MCE criterion in the context of continuous speech recognition can be written as follows:

$$J_{\mathrm{MCE}} = \sum_{r=1}^{R} \ell_r \left(-\log p_{\Lambda}(\mathbf{x}_r|s_r) + \log \left[\frac{1}{K-1} \sum_{s \neq s_r} p_{\Lambda}(\mathbf{x}|s)^{\eta} \right]^{1/\eta} \right), \tag{5.63}$$

where s is a model for a word sequence and s_r is the model for the word sequence corresponding to the observation $\mathbf{x}_r$. In this case, the classification error it measures is the sentence error and the classes ω_k correspond

to word sequences s. The misclassification measure however uses all possible incorrect classes: incorrect word sequences. It is obvious however that the number of incorrect word sequences is huge. Therefore only a set of most probable wrong word sequences is generated. In the literature, these incorrect word sequences are usually generated with an N-best list [16] or with lattices [87].

In MCE training using equation (5.63), there is a mismatch of the criterion with the usual word error rate measure, which is the common measure for speech recognition accuracy. Also the phone error rate is a common accuracy measure. It seems a more logical approach to try to minimize the word error rate or phone error rate rather than minimizing the sentence error rate. For this purpose, the phone error rate criterion was created. It is presented in § 5.4.6.

Results in HMM-based recognition

In general, discriminative training leads to better recognition results as compared to maximum likelihood training. However, the question which criterion is the best one is more difficult and this question is not frequently addressed in the literature. Most researchers still compare the performance of their discriminative strategy to ML training. Some small comparisons have been made. Reichl et al. for instance found that MCE training leads to better accuracy than MMI [82]. Schluter found that the difference between MMI and MCE reduces when the model complexity (represented by the number of Gaussians in the mixture) increases [86].

5.4.9 Other discriminative criteria

nearest neighbor rules

Paredes et al. have proposed a discriminative training procedure for training weights of a class dependent distance measure with diagonal weight matrices in the context of k-nearest neighbor classification [71]. It is defined as follows:

$$J_{\text{par}}(\Lambda) = \frac{\sum_{\mathbf{x}} d(\mathbf{x}, \mathbf{x}_{\text{nn}}^{=})}{\sum_{\mathbf{x}} d(\mathbf{x}, \mathbf{x}_{\text{nn}}^{\neq})}, \tag{5.64}$$

where $\mathbf{x}_{\text{nn}}^{=}$ is the nearest neighbor of $\mathbf{x}$ in the same class and $\mathbf{x}_{\text{nn}}^{\neq}$ is the nearest neighbor of $\mathbf{x}$ in a different class. This criterion was inspired on the intuitive observation that within-class distances should be small and inter-class distances should be large.

In more recent work, Paredes et al. have improved upon this distance measure by measuring the real NN classification error [72]. In this case, the criterion becomes:

$$J_{\text{par}}(\Lambda) = \frac{1}{N} \sum_{\mathbf{x}} \ell \left(\frac{d(\mathbf{x}, \mathbf{x}_{\text{nn}}^{=})}{d(\mathbf{x}, \mathbf{x}_{\text{nn}}^{\neq})} \right), \tag{5.65}$$

where $\ell(f)$ is a loss function, typically a sigmoid.

5.5 Optimization of criteria

5.5.1 Gradient descent

The gradient descent algorithm is a simple algorithm for optimization. It is an iterative procedure that optimizes the function $F(\Lambda)$ by constructing a set of parameters $\Lambda_1, \Lambda_2, \ldots$ such that $F(\Lambda_1) \geq F(\Lambda_2) \geq \ldots$. The next parameter set is found by moving the previous one in the direction of the gradient, i.e.

$$\Lambda_{i+1} = \Lambda_i - \gamma_i \nabla F(\Lambda_i). \tag{5.66}$$

Gradient descent has some important drawbacks. It is quite slow and quite sensitive to the learning rate γ_i. If the learning rate is fixed, the procedure can lead to bad optimization. On the other hand, it is very difficult to find good learning rates as a function of the iteration.

Gradient descent also suffers from computational issues. If we look at the MCE criterion for instance, this is a sum of the local loss over all training examples in the database, and usually, computing these local losses itself is a computationally expensive task, since it involves computing discriminants for all competing classes. Luckily variations on the gradient descent were defined which are able to optimize the parameter set more efficiently. The generalized probabilistic descent is such an algorithm.

5.5.2 Generalized probabilistic descent

The probabilistic descent method [1] and forthcoming generalized probabilistic descent (GPD) method for minimum error classification [44] were created in the context of the MCE framework for optimizing the parameters of parametric probability distributions. The high cost of computing the overall loss function was the main motivation for constructing this method. Probabilistic descent is usually defined in terms of a loss function. Suppose a loss function $\ell_k(\mathbf{x}, \Lambda)$ is defined. The total loss can be obtained by taking the expected value of this local loss.

$$L(\Lambda) = E[\ell(\mathbf{x}, \Lambda)] = \sum_{k=1}^{K} P(\omega_k) \int_{\mathcal{X}} \ell_k(\mathbf{x}, \Lambda) p(\mathbf{x}|\omega_k) d\mathbf{x}. \tag{5.67}$$

This loss cannot be computed in a straightforward way since the true probability distribution of $p(\mathbf{x}|\omega_k)$ and $P(\omega_k)$ are unknown. It is mentioned in [45, 44, 61] that for an infinite number of observations $\mathbf{x}_i$, it is also possible to find a (local) minimum of $L(\Lambda)$ with the following adaptation process:

$$\Lambda_{i+1} = \Lambda_i - \gamma_i U \nabla \ell_k(\mathbf{x}_i, \Lambda_i), \tag{5.68}$$

where γ_i is a step size that satisfies the following conditions:

$$\sum_{i=1}^{\infty} \gamma_i \to \infty \tag{5.69}$$

and

$$\sum_{t=1}^{\infty} \gamma_i^2 < \infty. \tag{5.70}$$

The matrix U is a scaling matrix allowing different features having a different weight. In the most basic case, U is the identity matrix.

The key aspect to observe in this method is that the total loss $L(\Lambda)$ does not have to be calculated. Instead, with each training observation, only the local loss for that particular observation has to be calculated.

The probabilistic descent function was first defined for the expected value of the local loss. It also needs an infinite amount of training observations. In practice however, this is very unlikely. GPD can also be applied

on problems with only a finite number of examples available. In that case, instead of the expected value, the empirical average loss is taken:

$$L_1(\Lambda) = \frac{1}{N} \sum_{k=1}^{K} \sum_{i=1}^{N_k} \ell_k(\mathbf{x}_{k,i}, \Lambda). \tag{5.71}$$

Proving that GPD also works for the empirical average instead of the expected value is straightforward [61]. The limitation of needing an infinite amount of training data is circumvented by reusing the same training data again after every available observation has been processed. In this case, it is also possible to batch process the whole training data before updating the parameters, i.e.

$$\Lambda_{i+1} = \Lambda_i - \gamma_i \nabla L_1(\Lambda_i). \tag{5.72}$$

5.6 Frame-based training of distance measures

In this section, we will explore frame-based training methods for distance measures. In this case we have a set X of feature vectors $\mathbf{x}_i, i = 1, \ldots, N$, where each label has a class label $\omega_k, k = 1, \ldots, K$. We will use the non-parametric densities defined in § 5.3.2, since the problems we want to tackle in this research are nearest neighbor classification and template based speech recognition, which are by nature non-parametric.

In order to compare feature vectors, we need a distance measure. Throughout this section, we use the class-dependent distance measure defined in § 3.3.2, i.e.

$$d(\mathbf{x}; \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Lambda_p (\mathbf{x} - \mathbf{y})}. \tag{5.73}$$

We will perform classification experiments with a k-nearest neighbor classifier. Therefore, the nearest neighbor likelihood function will be used.

5.6.1 Maximum Likelihood based distance measures

Suppose we have an input observation $\mathbf{x}$ and a reference frame $\mathbf{y}$ belonging to class C_k. If we assume the class C_k has a Gaussian distribution, we can

compute the log-likelihood of $\mathbf{x}$:

$$-\log(f(\mathbf{x}|C_k)) = -\log\left((2\pi)^{D/2}\right) + \log\left(|\Sigma|^{-1}\right) + \frac{1}{2}(\mathbf{x}-\boldsymbol{\mu}_k)^T\Sigma^{-1}(\mathbf{x}-\boldsymbol{\mu}_k) \tag{5.74}$$

Inspired by this likelihood, De Wachter et al. have proposed a ML-based distance measure, called the **local Mahalanobis distance** [23]. It is defined as follows:

$$d^2(\mathbf{x};\mathbf{y}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^T\Sigma_k^{-1}(\mathbf{x}-\mathbf{y}) + \log|\Sigma_k|^{1/2}, \tag{5.75}$$

where Σ_c is the covariance matrix of the class C_k, the class to which $\mathbf{y}$ belongs. Of course, the real covariance matrix of the class is unknown. Instead, it is estimated by the sample covariance matrix $\hat{\Sigma}_k$.

We recall that the classes C_k for the distance measure are not necessarily the same as the classes ω_k for classification. However, with frame-based distance measures, these classes are identical.

5.6.2 Discriminative training of frame-based distance measures

Maximum mutual information

In this section, we want to train the weights of the local distance measure using the maximum mutual information criterion. In this dissertation, we consider two problems: k-nearest neighbor pattern classification and template based speech recognition. The classification of speech is by nature not frame based, but phoneme-based, word-based or sentence-based. Therefore, frame-based training of distance measures is not optimal for TBSR. However, the first component of the TBSR system after preprocessing is a k-nearest neighbor selection. It seems not illogical that improving the nearest neighbor selection process will also improve the performance of the entire speech recognition system. It has been shown however that the graph generation is less crucial than the recognition itself. However, for completeness, we will also consider frame-based distance measures for the speech recognition problem.

In mutual information processing, we have to choose a likelihood function: $p_\Lambda(\mathbf{x}|\omega_k)$. In the context of k-nearest neighbor classification, we will use the nearest neighbor likelihood function for this, i.e.

$$p_\Lambda(\mathbf{x}|\omega_k) = \frac{1}{N_k} \frac{1}{c_D d_k(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D}, \tag{5.76}$$

where $\mathbf{x}_{\text{nn}}^{=}$ is a notation for the nearest neighbor of $\mathbf{x}$ in the same class.

The conditional entropy was defined in terms of the expected value of this likelihood. However, in our case, only a discrete number of samples are available. Instead of the expected value, we will use the sample mean, i.e.

$$\hat{H}_\Lambda(\Omega|Y) = -\frac{1}{N} \log \prod_{i=1}^{N} P_\Lambda(\omega_{\mathbf{y}_i}|\mathbf{y}_i), \tag{5.77}$$

where $\omega_{\mathbf{y}_i}$ is the class of $\mathbf{y}_i$.

The posterior likelihood can easily be worked out as follows:

$$P_\Lambda(\omega_{\mathbf{y}}|\mathbf{y}) = \frac{p_\Lambda(\mathbf{y}|\omega_{\mathbf{y}})P(\omega_{\mathbf{y}})}{\sum_k p_\Lambda(\mathbf{y}|\omega_k)P(\omega_k)}, \tag{5.78}$$

and the prior probabilities are estimated using frequency counts, i.e.

$$P(\omega_k) = \frac{N_k}{N}. \tag{5.79}$$

Combining the equations, together with the observation that the log operator does not influence the minimization process, the criterion to optimize becomes:

$$\begin{aligned} J_{\text{MMI}}(\Lambda) &= \log \prod_{i=1}^{N} \frac{p_\Lambda(\mathbf{y}_i|\omega_{\mathbf{y}_i})P(\omega_{\mathbf{y}_i})}{\sum_k p_\Lambda(\mathbf{y}_i|\omega_k)P(\omega_k)} \\ &= \log \prod_{i=1}^{N} \left(\frac{P(\omega_{\mathbf{y}_i})}{N_{\omega_i} c_D d(\mathbf{y}_i; \mathbf{y}_i^{=})^D} \Big/ \sum_k \frac{P(\omega_k)}{N_k c_D d(\mathbf{y}_i; \mathbf{y}_i^{(k)})^D} \right) \\ &= \log \prod_{i=1}^{N} \left(\frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{=})^D} \Big/ \sum_k \frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{(k)})^D} \right) \\ &= \sum_{i=1}^{N} \left(\log \frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{=})^D} - \log \sum_k \frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{(k)})^D} \right). \end{aligned} \tag{5.80}$$

This criterion can be optimized using gradient descent. We have to compute the derivative of $J_{\mathrm{MMI}}(\Lambda)$ with respect to every $\lambda_{k,i,j}$.

The derivative of a local distance is easily derived as follows, for $i \neq j$:

$$\begin{aligned} \frac{\partial}{\partial \lambda_{k,i,j}} d(\mathbf{x}, \mathbf{y}) &= \frac{\partial}{\partial \lambda_{k,i,j}} (\mathbf{x} - \mathbf{y})^T \Lambda_k (\mathbf{x} - \mathbf{y}) \\ &= \lambda_{k,i,j} (x_i - y_i)(x_j - y_j), \end{aligned} \tag{5.81}$$

if k is the class of $\mathbf{y}$, otherwise, the derivative is zero. Here x_i denotes the i-th component of $\mathbf{x}$. A special case occurs if $i = j$ because the diagonal elements are squares. In this case the derivative is:

$$\frac{\partial}{\partial \lambda_{k,i,i}} d(\mathbf{x}, \mathbf{y}) = 2\lambda_{k,i,j} (x_i - y_i)^2. \tag{5.82}$$

Minimum classification error

For minimum classification error training, a discriminant function has to be defined. In § 5.2.3, we have seen that choosing the posterior probability as discriminant leads to the Bayes classifier. Because of the optimality of the Bayes classifier, is seems natural to also take the (log) posterior as discriminant:

$$g_j(\mathbf{x}, \mathbf{\Lambda}) = \log P(\omega_j | \mathbf{x}) = \log \frac{p(\mathbf{x}|\omega_j) P(\omega_j)}{\sum_{k=1}^{K} p(\mathbf{x}|\omega_k P(\omega_k)}. \tag{5.83}$$

We now combine this discriminant with the misclassification measure of equation (5.47). Because of computational constraints, we will only look at the case $\eta \to \infty$. In this case the misclassification measure can be

derived:

$$\begin{aligned} d_j(\mathbf{x}, \Lambda) &= -\log \frac{p(\mathbf{x}|\omega_j)P(\omega_j)}{\sum_{k=1}^{K} p(\mathbf{x}|\omega_k)P(\omega_k)} + \log \frac{p(\mathbf{x}|\omega_i)P(\omega_i)}{\sum_{k=1}^{K} p(\mathbf{x}|\omega_k)P(\omega_k)} \\ &= -\log \frac{1}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D} + \log \sum_k \frac{1}{d(\mathbf{x}; \mathbf{x}^{(k)})^D} \\ &\quad + \log \frac{1}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{\neq})^D} - \log \sum_k \frac{1}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{(k)})^D} \\ &= -\log \frac{1}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D} + \log \frac{1}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{\neq})^D} \\ &= \log \frac{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{\neq})^D}, \end{aligned} \tag{5.84}$$

where ω_i is the most confusing class for the observation $\mathbf{x}$. I.e. the class for which the discriminant is minimal excluding the discriminant of the correct class. The symbol $\mathbf{x}_{\text{nn}}^{\neq}$ denotes the nearest neighbor of $\mathbf{x}$ in the most confusing class ω_i, and $\mathbf{x}_{\text{nn}}^{(k)}$ denotes the nearest neighbor of $\mathbf{x}$ in class ω_k.

The derived criterion is very intuitive: if the distance from $\mathbf{x}$ to its nearest neighbor in the same class is larger than the distance to its nearest neighbor in a different class, classification is incorrect. In this case the division of both will be larger than 1 and the misclassification measure will be larger than zero and vice versa. This is also what we expect from a one-nearest neighbor classifier: classification is correct if its nearest neighbor is in the correct class and wrong otherwise.

Next, we need to choose a loss function, approximating the classification error itself. We have chosen the sigmoid function for this purpose, i.e.

$$\begin{aligned} \ell_k(\mathbf{x}, \Lambda) &= 1 \Big/ \left(1 + \exp\left(-\epsilon \log \frac{d(\mathbf{x}; \mathbf{x}^{=})^D}{d(\mathbf{x}; \mathbf{x}^{=})^D}\right)\right) \\ &= 1 \Big/ \left(1 + \left(\frac{d(\mathbf{x}; \mathbf{x}^{=})}{d(\mathbf{x}; \mathbf{x}^{\neq})}\right)^{-\epsilon D}\right). \end{aligned} \tag{5.85}$$

Note that the criterion obtained in this way is highly similar to the criterion from Paredes et al. in equation (5.65).

Once again, this criterion can be optimized using gradient descent. We have to compute the derivative of $l_k(\mathbf{x}, \Lambda)$ with respect to every $\lambda_{k,i,j}$. Naturally, this derivative will be 0 if C_k is not the class of $\mathbf{x}$ or $\mathbf{x}^{\neq}$.

In the case ω_k is the class of $\mathbf{x}$ the derivation of the update equation is as follows:

$$\begin{aligned}\frac{\partial}{\partial\lambda_{k,i,j}}\ell_k(\mathbf{x},\Lambda) &= -\frac{1}{\ell_k(\mathbf{x},\Lambda)}\frac{\partial}{\partial\lambda_{k,i,j}}\left(\frac{d(\mathbf{x};\mathbf{x}^{=})}{d(\mathbf{x};\mathbf{x}^{\neq})}\right)^{-\xi D} \\ &= \frac{\xi D}{\ell_k(\mathbf{x},\Lambda)}\left(\frac{d(\mathbf{x};\mathbf{x}^{=})}{d(\mathbf{x};\mathbf{x}^{\neq})}\right)^{-\xi D-1}\frac{1}{d(\mathbf{x};\mathbf{x}^{\neq})}\frac{\partial}{\partial\lambda_{k,i,j}}d(\mathbf{x};\mathbf{x}^{=}).\end{aligned} \tag{5.86}$$

In the case C_k is the class of $\mathbf{x}^{\neq}$, the equation is as follows:

$$\begin{aligned}\frac{\partial}{\partial\lambda_{k,i,j}}\ell_k(\mathbf{x},\Lambda) &= \frac{\xi D}{\ell_k(\mathbf{x},\Lambda)}\left(\frac{d(\mathbf{x};\mathbf{x}^{=})}{d(\mathbf{x};\mathbf{x}^{\neq})}\right)^{-\xi D-1} d(\mathbf{x};\mathbf{x}^{=})\frac{\partial}{\partial\lambda_{k,i,j}}\frac{1}{d(\mathbf{x};\mathbf{x}^{\neq})} \\ &= \frac{\xi D}{\ell_k(\mathbf{x},\Lambda)}\left(\frac{d(\mathbf{x};\mathbf{x}^{=})}{d(\mathbf{x};\mathbf{x}^{\neq})}\right)^{-\xi D-1}\frac{d(\mathbf{x};\mathbf{x}^{=})}{d(\mathbf{x};\mathbf{x}^{\neq})^2}\frac{\partial}{\partial\lambda_{k,i,j}}d(\mathbf{x};\mathbf{x}^{\neq}).\end{aligned} \tag{5.87}$$

5.7 Unit-based training of distance measures

In § 5.6, we presented some strategies to train the weights of a distance measure based on a frame-based criterion. For nearest neighbor classification, this is the optimal strategy to follow. However, for speech recognition, things are a bit different. Speech recognition is concerned with the correct classification of sequences of frames and the categories are phonemes, words or even sentences. In this section, we will define some training strategies for optimizing the distance measure for speech recognition.

The main difference in speech recognition as compared to pattern classification is in the likelihood function. These likelihood functions are no longer based on one single observation, but rather on a sequence of observations. In this section, $\mathbf{x}$ and $\mathbf{y}$ denote sequences of observations.

5.7.1 Phone-based distance measures

In this section, phoneme-based training procedures will be constructed. Phones are selected based on their likelihood score. In template based speech recognition, the similarity of phonemes is computed using the dynamic time warping algorithm. In a sense, this DTW algorithm computes a distance between sequences of phonemes and the most likely phonemes are the nearest neighbors of the input to the reference phonemes in the database.

To model the likelihood of an input $\mathbf{x}$, we have to construct a likelihood function. The standard nearest neighbor likelihood function cannot be used any more. The dimension of the sphere is not fixed, and the DTW distance measure is even further away from a metric than the class-dependent distance measure.

Therefore, we have to define a new likelihood function. Inspired on the nearest neighbor density, we define the likelihood as follows:

$$P_\lambda(\mathbf{x}|\omega_k) = \frac{1}{N_k D_{\text{DTW}}(\mathbf{x}; \mathbf{x}^=)}. \tag{5.88}$$

Intuitively, this is still a distance measure: the likelihood is high if the distance to the nearest DTW neighbor is low and vice versa. Note however that this likelihood is no longer a probability distribution. Luckily, discriminative training also works for this kind of distance measures.

This DTW distance is also a function of the parameters Λ, because it is a weighted sum of local distance measures between frames. The path through the local distance matrix can easily be reconstructed if considering a slight variation of the standard DTW algorithm: storing backpointers of the chosen path.

We can now apply this likelihood to discriminative training of the distance measures. Experimental results on speech recognition databases are shown in chapter 7.

MMI training

The principle of MMI training with the phoneme-based likelihood is entirely the same as for the frame-based likelihood. We replace the likelihood

with the DTW-based likelihood from equation (5.88).

The only difference is the change of distance function. The criterion for MMI training becomes:

$$J_{\mathrm{MMI}}(\Lambda) = \sum_{i=1}^{N} \left(\log \frac{1}{D_{\mathrm{DTW}}(\mathbf{y}_i; \mathbf{y}_i^{=})} - \log \sum_k \frac{1}{D_{\mathrm{DTW}}(\mathbf{y}_i; \mathbf{y}_i^{(k)})} \right). \quad (5.89)$$

The derivation of update equations is straightforward. The only difference with frame-based MMI is that the dissimilarity measure now is a sum of local distances.

MCE training

For MCE training, we can also follow the same strategy. We still take the case $\eta \to \infty$. The first step is defining a discriminant. There are several options that seem possible. The first option is to choose the log posterior probability itself as discriminant, with the DTW-based likelihood of equation (5.88). The derivation of the loss is analogous as in the frame-based case. The loss function becomes:

$$\ell_k(\mathbf{x}, \Lambda) = 1 \Bigg/ \left(1 + \left(\frac{D_{\mathrm{DTW}}(\mathbf{x}; \mathbf{x}^{=})}{D_{\mathrm{DTW}}(\mathbf{x}; \mathbf{x}^{\neq})} \right)^{-\xi} \right). \quad (5.90)$$

Another option for the discriminant is the likelihood measure itself. In template based speech recognition, this "likelihood" is measured by the DTW distance between two sequences of frames. To be correct however, also the prior likelihood has to be taken into account. In this case, we have the following discriminant:

$$g_k(\mathbf{x}, \Lambda) = -\frac{N_k}{N} D_{DTW}(\mathbf{x}, \mathbf{x}_{\mathrm{nn}}^{(k)}). \quad (5.91)$$

5.7.2 Sentence-based distance measures

In the previous sections, we trained the distance measure weights only on frame-based criteria, with the goal to improve the nearest neighbor

selection and on template-based criteria, which was motivated by trying to improve the template selection.

In continuous speech recognition however, the task at hand is recognizing entire sentences. Therefore the final weight training strategy we present is based on improving recognition by training weights on entire sentences. All discriminative objective functions try to improve the likelihood of the correct sentence and at the same time decreasing the likelihood for a set of competing sentences. This set of competing sentences has to be constructed. In the literature, there are two large categories of methods for computing the most likely competing sentences. The first method is by constructing an N-best list, which basically is a list of most likely sentences. The other alternative is by creating a word lattice. The strategy of using word lattices was first presented by Macherey et al. [53]. Using word lattices complicates discriminative training, since the correct sentence is no longer uniquely defined in the lattice, because next to the transcription, also the time alignment is part of this lattice. Usually several equal transcriptions with slightly different time alignments are present in these graphs. Therefore, instead of only taking into account the best correct sentence, the sum of all likelihoods of correct transcriptions has to be taken.

Another complication in lattice-based training is that it is not easy to remove the correct transcriptions from these graphs, since parts of these correct transcriptions might also be present in incorrect hypothesis. Therefore the approach Macherey et al. take is to subtract the likelihood of the correct hypothesis from the probability of the spoken word sequence [53].

Chapter 6

k-nearest neighbors

Throughout this dissertation, the k-nearest neighbor problem is encountered several times. In low dimensional spaces, the k-NN problem has been solved efficiently. However, for moderate to high dimensional spaces, documentation on the problem is scarce. We investigate the applicability of several nearest neighbor methods on real speech data. In [56], we presented an overview of tree-based nearest neighbor methods and compared them. An extended version of this comparison is presented in this chapter.

6.1 Introduction

The (k-) nearest neighbor problem occurs in a wide range of areas, including pattern recognition, speech and image processing, data compression, databases, computer graphics etc. In this thesis, the k-nearest neighbor problem is encountered in two different situations. First of all, the first step of the template selection of the TBSR system is a nearest neighbor search for every input frame. The second place where the k-NN problem is encountered is in the discriminative training procedures for the class dependent distance measure.

In this chapter, we present an overview of some popular and less popular k-nearest neighbor algorithms and compare performance results. In many cases, the k-NN problem has to be tackled in high-dimensional

spaces, ranging from two to a few hundred dimensions.

6.1.1 Classification of nearest neighbor methods.

Data structures

k-nearest neighbor methods can be classified according to the data structure they build and use to perform an efficient search. Several popular methods build tree-like structures, where each node in the tree corresponds to a separation of the feature space. Examples of such structures are kd-trees and bbd-trees. Other methods use graph structures, where each node corresponds to a data point, and links between nodes correspond to points that are close together. An example is the extended roadmap algorithm.

Exact vs. approximate nearest neighbors

Another classification that can be made is between exact and approximate nearest neighbors. Exact nearest neighbors looks for the real nearest neighbors.

Several researchers have tried to reduce the complexity of the nearest neighbor search by relaxing the query. Instead of searching for the exact k-nearest neighbors, the ϵ-k-nearest neighbors suffice. A point p is an ϵ-k'th nearest neighbor of a query point q if the ratio of the distances between p and q and between the real k'th nearest neighbor and q is not larger than $1 + \epsilon$. The first researcher who used this idea was Arya [6]. Later, several other researchers proposed various algorithms for the approximate nearest neighbor problem.

6.1.2 The Curse of Dimensionality

Before explaining the different methods in detail, a remark has to be made about an inherent problem in k-nearest neighbor searches (and many other geometric problems). Many (if not all) k-nearest neighbor algorithms suffer from the curse of dimensionality [12]. The curse of dimensionality can be expressed in many forms. In its most general form, the curse of dimensionality states that the volume of a mathematical space is exponentially

proportional to the dimension. This also means that the number of points in a space has to increase in an exponential way with increasing dimension to fill the space with the same density. In practice, the number of data points is usually limited resulting in a very sparse occupation of the high dimensional space.

The curse of dimensionality has big consequences. It means that in high dimensional data spaces, many of our intuitive notions about 3D geometry are incorrect. Böhm et al. present some examples of this observation [14]. One of these examples is the following. In high dimensional spaces, a sphere intersecting with every boundary of a cube does not necessarily include the center of the cube: a 16-dimensional sphere with center-point $(0.3, 0.3, \ldots, 0.3)$ and radius 0.7 obviously intersects every boundary of the unit hypercube. However, the distance from the center-point to the center of the cube is $\sqrt{16 \times 0.2^2} = 0.8$. The center of the cube is not a part of this sphere. In fact, the volume of a unit sphere becomes insignificant compared to the volume of the unit hypercube in high dimensions.

Another peculiar observation is that the volume of the unit sphere decreases with increasing dimension. In the limit case with $D \to \infty$, the volume is 0.

In terms of the k-nearest neighbor problem, the curse means that the number of data points that have to be investigated increases exponentially with the dimension of these points. One possible explanation for this is that the variances of the distances of uniform data decrease with increasing dimension. This means that the more dimensions the problem has, the closer the nearest and the farthest neighbor lie to each other. As we will see, this has significant consequences for nearest neighbor search algorithms in moderate to high dimensional spaces.

6.2 Tree Structures

Many practical k-nearest neighbor search algorithms use tree structures to index the data. Among those structures are Samet's quad-trees [85], kd-trees originally proposed by Bentley [13] with its different variants [32, 62] and bbd-trees proposed by Arya [5]. Good overviews of different indexing

structures based on trees are written by Gaede et al. [36] and Böhm et al. [14].

There are many tree structures described in the literature. Many of them use hyperplanes to split the data. In this section, two kinds of these trees that are used in the experiments are explained, namely kd-trees and bbd-trees. A short overview is presented on how such a tree is created and the complexity of tree creation is mentioned.

6.2.1 Kd-trees

Kd-trees were invented by Bentley [13] and were proposed as a tool for fast nearest neighbor queries. At each node of the tree, the data space is split with a hyperplane along one of the basic dimensions. Originally, as proposed by Bentley, the space is split along consecutive dimensions of the search space at each level of the tree. So, in the root node, the space is split along the first dimension, on the second level, the split occurs along the second dimension etc. A few years later, Friedman et al. proposed another way to choose the splitting direction. Their method uses the direction with the maximal variance of point coordinates [32].

Tree construction

The kd-tree construction algorithm is a procedure that incrementally splits the data space by creating a $D-1$ dimensional hyperplane along a basic dimension of the feature space. The main construction algorithm is shown in fig. 6.1. An example of a point cloud with its space partitioning and the corresponding kd-tree is shown in fig. 6.2. The line in the tree nodes denotes the direction of the split.

As explained, the kd-tree method has several variants and parameters. This leads to several choices for the developer to make. One of these choices is the direction of the split in each node. Friedman et al. have shown that, in small dimensional spaces, performance improves if the direction with maximal variance is chosen [32].

Another parameter in kd-trees is the so called bucket size b. This is the maximal number of data points that can be put in a leaf node of

```
1: function kd-tree_training(𝒳, b)
2: if empty(𝒳) then
3:    return null
4: else if size(𝒳) < b then
5:    return new leaf(𝒳)
6: else
7:    node = new node()
8:    node.dir := split_direction(𝒳)
9:    node.m = median(𝒳, dir)
10:   𝒳1, 𝒳2 = split(𝒳, dir, node.m)
11:   node.left = kd-tree_training(𝒳1, b)
12:   node.right = kd-tree_training(𝒳2, b)
13:   return node
14: end if
```

Figure 6.1: The kd-tree construction algorithm

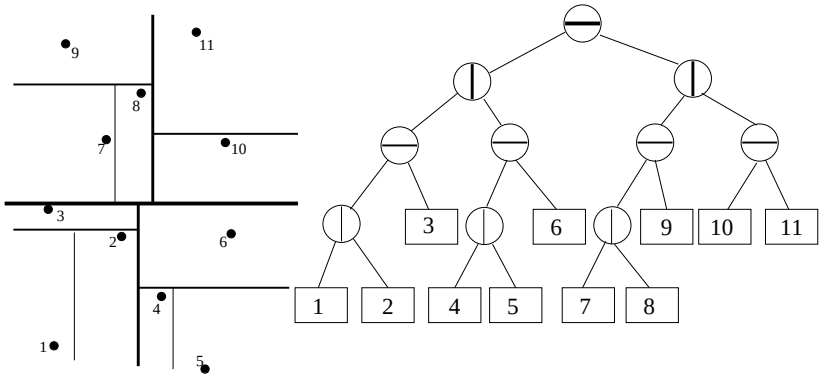


Figure 6.2: Point cloud and its corresponding kd-tree

the kd-tree. The standard tree as proposed by Bentley uses a bucket size of 1, but there are valid arguments to increase this bucket size. One of the arguments is that elements that are in the same bucket have a high probability to be close to each other, so if one of the points in the bucket is a k-nearest neighbor, some of the other points in the bucket are it probably too. In this case, the standard kd-tree algorithm with bucket size 1 will cause more overhead than the brute force computation of the distance from the query point to all the points in the bucket.

One of the disadvantages of the kd-tree is that buckets can become very narrow. The result of this is that, when searching for nearest neighbors, more nodes have to be examined, increasing the computing time. To overcome this, another criterion called "balanced aspect ratio" was introduced. This criterion ensures the ratio of the length of longest and shortest side of the bucket remains within certain bounds. This ensures that the buckets do not become "narrow". However, for certain data distributions, the kd-tree algorithm fails to meet this criterion [31]. On the other hand, the splitting in kd-trees occurs along the median of the coordinate values in the split direction. It is easy to see that in this case, the balanced aspect ratio is not always guaranteed. To overcome this, some alternative splitting rules have been proposed.

Alternatives for the splitting rule. As mentioned above, the standard kd-tree algorithm uses a median splitting rule. It is easy to see that, for certain non-uniform data distributions, this leads to splits which violate the balanced aspect ratio criterion. Most alternative splitting rules guarantee the balanced aspect ratio at the cost of losing the equal distribution of the data in the tree nodes. This leads to unbalanced trees. Some performance results on these splitting rules are shown in § 6.4.5.

Time complexity. The expected time complexity of kd-tree construction is $O(KN \log N)$, with K the dimension of the search space and N the number of data points. This time complexity is acceptable. Of far more importance however is the time complexity of the search algorithm to find the K-NN. The training of the tree has to be performed only once, the search is performed for every query point.

Kd-tree search

An algorithm for searching the k-nearest neighbors of a query point using optimized kd-trees was proposed by Friedman [32]. Summarized, this algorithm searches through the kd-tree, starting at the node in which the query point is located until the hypersphere centered at the query point and with radius the distance to the k'th closest nearest neighbor found so far, fits entirely within the kd-node currently under investigation. In this case, all remaining points in the kd-tree are further away than that k'th closest nearest neighbor so these points don't need to be investigated. The basic structure of the kd-tree search algorithm is shown in fig. 6.3. The function "c_dist" calculates the distance between a query point and a cell of a kd-tree.

Fig. 6.4 shows an example of a kd-tree search for one nearest neighbor. The square-shaped point is the query point for which the nearest neighbor has to be found. The * indicates the node under current investigation and the highlighted point is the current point marked as nearest neighbor.

Complexity. The complexity of the kd-tree search algorithm is not easily derivable. In his paper, Friedman derives that the search time is logarithmic in the number of points N [32]. He also states that the number of records that are examined for the k-nearest neighbors according to the ∞-distance in the ideal case is

$$\overline{R}_\infty(k, D) = (k^{1/D} + 1)^D, \tag{6.1}$$

which shows an exponential behavior in the dimension, the curse of dimensionality, and a sub-linear behavior in the number of nearest neighbors. However, practical experiments show that the real behavior of the kd-tree search algorithm is much worse (even super-linear). For an empirical derivation of the time complexity in the number of requested nearest neighbors, see § 6.4.6.

```
1: function kd-tree_search(node, x, var nn)
2: dim = node.dir
3: if isleaf(node) then
4:    worst = nn[0]
5:    for i=1 to node.nbelements do
6:       if worst > distance(x, node.elements[i]) then
7:          add_nn(nn, node.elements[i])
8:       end if
9:    end for
10: else
11:    if x[dim] ≤ node.m then
12:       kdtree_search(node.left, x)
13:       if c_dist(x, node.right) < nn[0].dist then
14:          kdtree_search(node.right, x)
15:       end if
16:    else
17:       kdtree_search(node.right, x)
18:       if c_dist(x, node.left) < nn[0].dist then
19:          kdtree_search(node.left, x)
20:       end if
21:    end if
22: end if
```

Figure 6.3: The kd-tree search algorithm

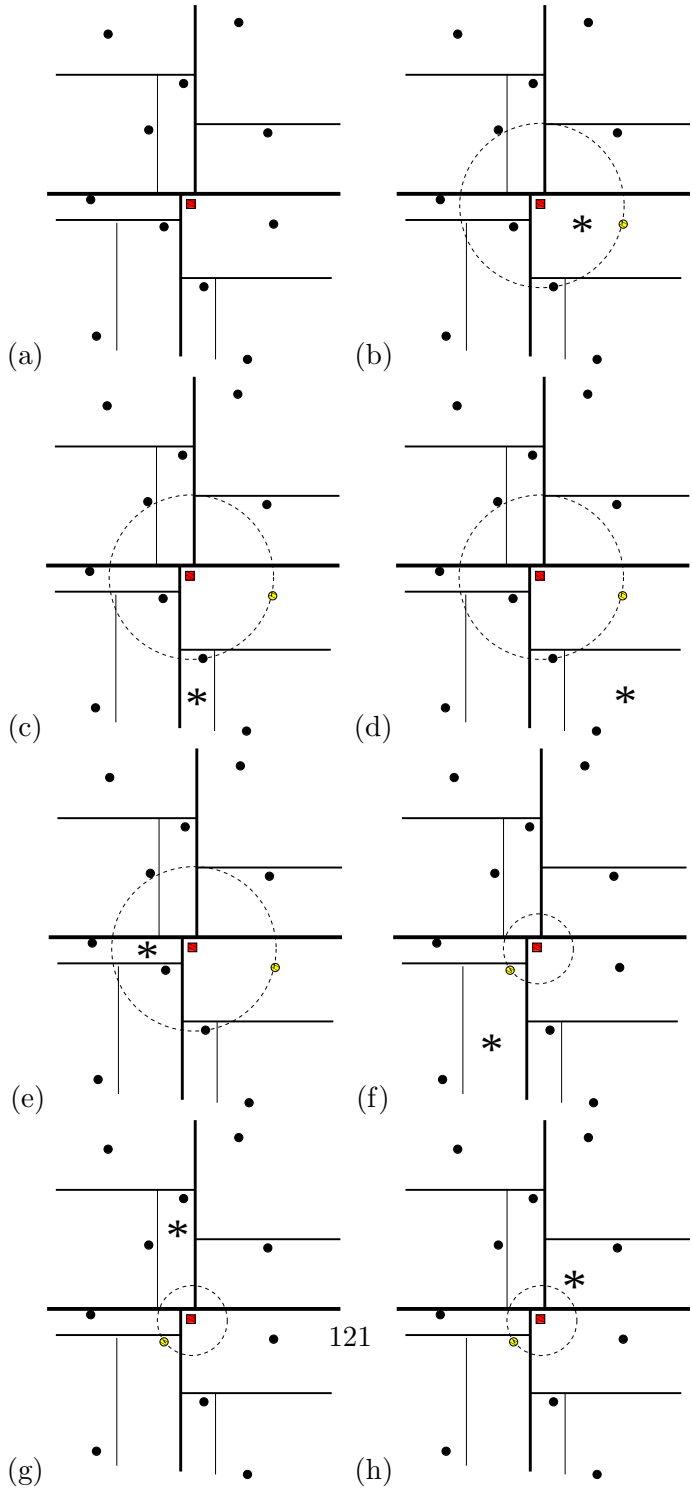


Figure 6.4: Kd-tree standard search algorithm

Priority Search

Priority search is a search technique that lists the nodes in the search tree in a somewhat more intelligent way compared to the standard kd-tree search. It visits the nodes in the tree in order of increasing distance to the query point q. This technique is described by Arya and Mount [6]. The technique can be implemented in an easy way using a priority queue. The priority queue is sorted according to the distance between q and the tree node. At each internal node in the search algorithm, the child node farthest from q is inserted into the sorted priority queue together with its distance to q. If the search arrives at a leaf node, the distances to the points in that leaf node are computed and the list of k-NN is updated. If there is a node in the priority queue with distance closer than the k'th nearest neighbor found so far, search continues in that node. The search terminates when distance of q to the closest node in the priority queue is greater than the distance to the k'th nearest neighbor found so far.

6.2.2 Bbd-trees

Bbd-trees (balanced box decomposition) were introduced by Arya [5]. This type of trees uses two splitting criteria in the tree nodes. The first one is an iso-oriented hyperplane, similar to the ones used in kd-trees. The second one is a shrink operation which zooms in on the data if the data is highly clustered. For this shrink-operation, different alternatives have been proposed by Ayra, details can be found in [5].

Another aspect of these trees is that they satisfy the "balanced aspect ratio" criterion. A consequence of this is that bbd-trees cannot guarantee that the points are evenly distributed among the nodes of the tree. I.e. one subtree of a node can have more data points allocated to it than the other subtree in that node.

Time complexity. The expected time complexity for bbd-tree time construction is equal to that of kd-tree construction, namely $O(KN \log N)$.

For the bbd-trees, Arya was not able to give bounds on the runtime other than a trivial one. His search algorithm also suffers from the curse of dimensionality: its complexity has exponential factors in the dimension of the search space too, as seems the case for every investigated space partitioning algorithm.

6.3 Graph structures

De Wachter et. al have proposed a new algorithm for nearest neighbor search: the extended roadmap algorithm [23]. It is based on another algorithm, the Roadmap algorithm of Povey et al. for efficient selection of similar Gaussians [75]. In this section we will present the basic idea of data structure creation and search.

6.3.1 The Roadmap algorithm

The original Roadmap algorithm, created by Povey et al. is an algorithm that is able to efficiently locate similar Gaussians [75, 76]. In order to define the similarity between two Gaussians g_1 and g_2, a distance measure is defined:

$$d(g_1(\mathbf{x}); g_2(\mathbf{x})) = -\sum_{i=1}^{D} \log O(g_1^{(i)}(x_i), g_2^{(i)}(x_i)), \tag{6.2}$$

where $O(g_1^{(i)}, g_2^{(i)})$ is a measure for the overlap between g_1 and g_2 along dimension i. The training algorithm then constructs a graph where each Gaussian is represented by a node in the graph. Next, for each Gaussian, a list with similar Gaussians is created using the distance measure of equation (6.2). These similar Gaussians are connected by inserting a node in the graph. The search through this graph is performed by a hill climbing strategy. For the details of the algorithm, the reader is referred to Povey et al. [75].

6.3.2 The extended roadmap algorithm

Inspired by the idea of the roadmap algorithm for finding similar Gaussians, De Wachter et al. have proposed a new method for nearest neighbor searches called the **extended roadmap** algorithm [23]. The algorithm builds a graph structure similar to the roadmap algorithm. However there are some differences. Because the feature space in our context is partitioned into several classes and the distance measures between these classes are not metrics, the algorithm constructs one graph per class. Each node in these graphs corresponds to a data point in feature space belonging to the class represented by the graph.

Extended roadmap training

The training algorithm consists of two basic operations.

- **Linking** is an operation between two nodes x and y. It works by trying to get from y as close to x as possible by following arcs in the graph in such a way that the distance to the target decreases (or only slightly increases) with each step. The same operation is performed the other way around: from x to y and an arc is added between the two obtained endpoints of the search if necessary.

- **Cleaning** is an operation to clean up arcs in the graph. If connected nodes can still be without a specific arc, the arc is removed.

Using these steps, the roadmap training algorithm can be constructed. The main structure of the algorithm is shown in fig. 6.5. The stopping conditions for the algorithm are usually based on the total number of arcs in the graph.

Extended roadmap search

The algorithm for searching through the extended roadmap contains two steps. In the first step, the nearest neighbor of the query point is searched in each of the graphs. In the second stage, the k-nearest neighbors are searched in each graph simultaneously. The nearest neighbor in a graph

```
1: function roadmap_training(𝒳,k)
2: Partition 𝒳 into graphs for each class, making sure that each graph is
   connected.
3: while continue() do
4:   for i = 1 to size(𝒳) do
5:     NN = nearest_neighbors(x_i, k, class(x_i))
6:     for j = 1 to k do
7:       link(x_i, NN_j)
8:     end for
9:     R = random_neighbors(x_i, k, class(x_i))
10:    for j = 1 to k do
11:      link(x_i, R_j)
12:    end for
13:    clean()
14:  end for
15: end while
```

Figure 6.5: The roadmap training algorithm

is found by a search similar to the search in the linking procedure: start from a random node and get as close to the input as possible. The second step is performed by iteratively expanding the active nodes in the graph with the nodes they are linked to. If no active nodes are closer to the input than the current k-th nearest neighbor, the search is terminated and the current list of k-nearest neighbors is returned.

Time complexity. De Wachter et al. created an efficient implementation of the roadmap algorithm using hash tables to avoid computing the same distance calculations twice. In the worst case scenario, input and initialization are on the outer ends on the graph and N_c distance calculations have to be performed, leading to a time complexity of $\mathcal{O}(N_c K)$. By using the hash tables, De Wachter et al. reduced the time complexity of the second step in the search to $\mathcal{O}(Dk \log k)$. This second step dominates the search when the number of requested nearest neighbors is large, which is the case in the context of the template selection in TBSR. Moreover, in the current experimental setup of the template based speech recognition system, k is dependent on N. In the current implementation, a linear relation is made between k and N, however, as De Wachter et al. have argued, the real dependence is probably sub-linear [20].

Nearest neighbors. Finally, it has to be mentioned that the extended roadmap algorithm is an approximate nearest neighbor algorithm. It depends on the parameters of the training algorithm how connected the graphs are: the more connections between nodes there are, the better the accuracy of the search will be, but at the same time, the slower search will be performed. It is very difficult however to find a concrete expression of how "approximate" the nearest neighbors are since it also depends on the distribution of the example data itself. This approximation property will be verified experimentally.

Number of points	1516015
Number of dimensions	25
Number of classes	149

Table 6.1: TiDigits data set

6.4 Experiments

In this section, the results of some experiments with the different k-nearest neighbor algorithms are presented. Experiments on kd-trees and bbd-trees were performed using the ANN toolbox of Arya et al. [4]. For experiments with the roadmap algorithm, software from De Wachter et al. was used [23]. All experiments were performed on real speech data. The data itself comes from the TiDigits database [49]. The data is preprocessed using standard MFCC features and decorrelated using MIDA, generating 25-dimensional feature vectors. The data is labeled with a class using phonetic decision trees. The distance measure used throughout the experiments is the local Mahalanobis distance. If not otherwise stated, all data from TiDigits is used. The standard parameters for TiDigits are shown in table 6.1.

The tree structures used in the experiments are created in a similar way as the roadmap: one tree per class is constructed. Search occurs by looking for the nearest neighbor of a query point in every tree simultaneously.

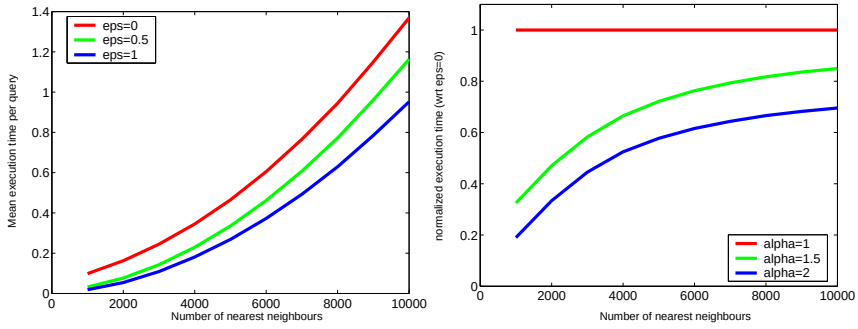
6.4.1 Exact vs Approximate k-nearest neighbors

In this experiment, the influence of the approximation factor ϵ is investigated. A first experiment investigates how many of the exact nearest neighbors are present in the list of returned nearest neighbors for varying values of ϵ. These results are computed using kd-tree software and are shown in table 6.2. Notice that the number of correct nearest neighbors decreases rapidly with increasing value of ϵ.

The introduction of approximate nearest neighbors comes at the cost of no longer obtaining the exact NN. In a second experiment, the gain

ϵ	100-NN	1000-NN	5000-NN	10000-NN
0	100%	100%	100%	100%
1	85.36%	81.07%	77.66%	75.95%
2	54.88%	51.17%	49.50%	49.12%

Table 6.2: Number of exact k-nearest neighbors computed as a function of ϵ.



(a) Execution times of ANN algorithms (b) Comparison of execution time of ANN and exact NN

Figure 6.6: Searching (approximate) NN with a kd-tree for varying ϵ.

of using approximate NN instead of exact NN is investigated. Naturally, the expectation is that the higher the approximation factor is, the faster the search will be performed. In fig. 6.6, execution time measurements are shown with approximate nearest neighbors for increasing number of requested NN. It can be seen from fig. 6.6b that the relative gain in computing time decreases with increasing number of requested nearest neighbors. As the number of requested nearest neighbors is high in our TBSR setup (0,5% to 1% of the database size), the gain from using approximate nearest neighbors instead of exact NN is very low. Moreover, the number of correct nearest neighbors decreases with increasing k. In this context, approximate nearest neighbors do not seem useful in our TBSR context.

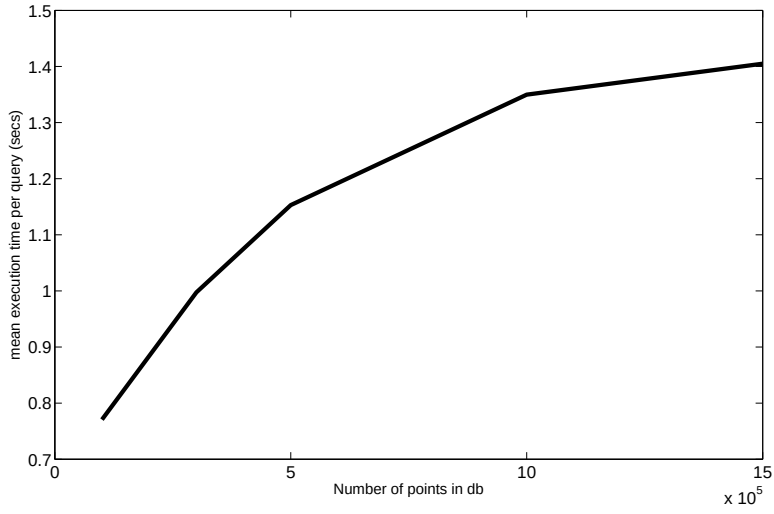


Figure 6.7: Scale-up experiment with kd-trees

6.4.2 Scale-up

This experiment investigates the effect of increasing number of elements N on the execution time. A good behavior of the algorithm in the context of TBSR is very important, because in the ideal case, an increase in amount of data does not lead to an increase in execution time. However, this seems very unlikely. Instead, having a less than linear behavior would be a nice property of the algorithm. According to Friedman, the complexity of the search should be logarithmic in number of elements N [32]. The results of this experiment are shown in fig. 6.7.

Concerning the roadmap search algorithm, De Wachter et al. state that the worst case complexity of the roadmap search is linear in the number of points per class, however, experimental results show that this worst case scenario does not occur in practice [20].

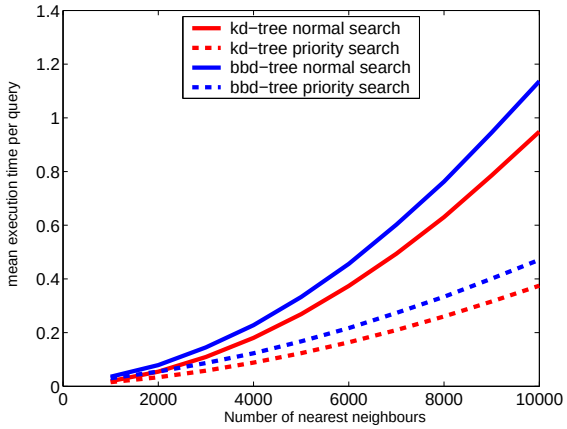


Figure 6.8: Comparison of kd-tree and bbd-tree search algorithms

6.4.3 Kd-trees vs Bbd-trees

This experiment compares kd-trees and Bbd-trees. Fig. 6.8 shows a comparison for the approximate nearest neighbors problem with $\epsilon = 1$. Strangely, the performance of the bbd-tree search is worse than the performance of the kd-tree search. A possible explanation for this is that the data is not highly clustered. This could be due to the curse of dimensionality: the distribution of data across feature space is more uniform with increasing dimension and the bbd-algorithm only produces overhead in this case because the shrink operation will seldomly be used and will only produce an extra load.

6.4.4 Standard vs Priority Search

This experiment compares the standard kd-search algorithm and the priority search algorithm of § 6.2.1. The results of this comparison are displayed in fig. 6.9. As can be seen, the introduction of the priority search technique improves the performance of the search drastically for kd-trees as well as for bbd-trees. The behavior of priority search in the number of requested nearest neighbors is almost linear for the first 10000

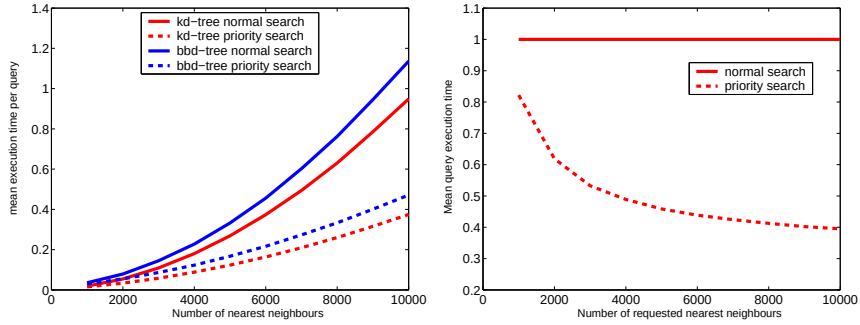


Figure 6.9: Comparison between standard kd-tree search and priority search.

nearest neighbors.

6.4.5 Splitting rules and the cell aspect ratio

A criterion to investigate the quality of a tree is the mean cell aspect ratio. This is the ratio between the lengths of the longest and the shortest boundaries of a cell. Experiments with the standard kd-tree split rule have shown that the mean aspect ratio of the cells is quite large (over 33). If this ratio is large this means that the cells are quite "skinny". One can intuitively see that this degrades performance of the search. In this experiment another splitting rule is chosen (sliding midpoint splitting) which guarantees a lower aspect ratio. Some results are shown in fig. 6.10. The restriction of balanced aspect ratio provides only minor improvements to the search performance.

6.4.6 Empirical complexity of the search

Time complexity in the number of data points

According to Friedman, the time complexity of the k-nearest neighbor search with kd-trees is $O(\log N)$ [32]. This can be verified empirically by producing a log-plot of the execution times of the scale-up experiment from § 6.4.2. If the complexity is indeed logarithmic, this plot should appear

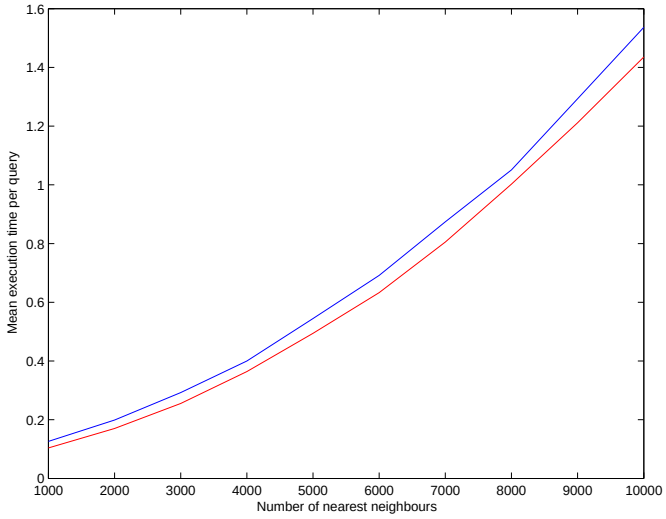


Figure 6.10: Comparison of splitting rules. top line is standard split, bottom line is sliding midpoint split

as a straight line. Such a log-plot is shown in fig. 6.11. As expected, this plot indeed is approximately straight.

Time complexity in the number of requested nearest neighbors

The derivation of the time complexity as a function of the number of requested nearest neighbors is more difficult. We were unable to find an analytical expression of the time complexity as a function of the number of requested nearest neighbors. For ideal cases and the maximum distance, the complexity could be approximated using equation (6.1) on p. 119. According to this equation, the time complexity should be sub-linear in k. In practice this is not the case as can be derived from several plots in this dissertation.

If the complexity is $O(k \log k)$ then a log-plot of the function $t(k)/k$ should be a straight line for k large enough. Such a plot is shown in fig. 6.12 on p. 134. This is not at all a straight line, contradicting the

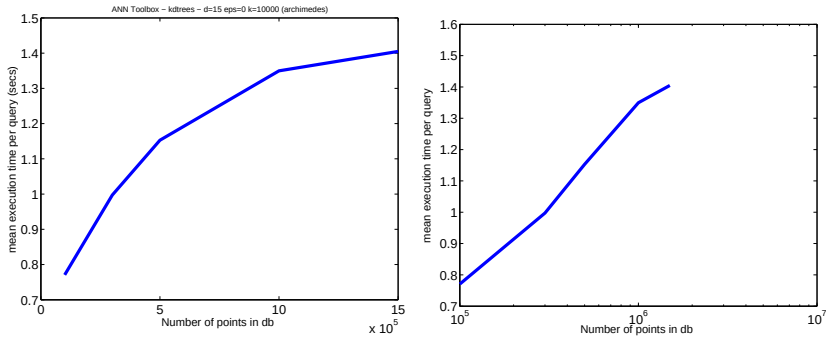


Figure 6.11: (a) Mean execution time as function of number of requested nearest neighbors. (b) Log-plot of (a)

hypothesis that the complexity is $O(k \log k)$.

Another possibility is that the complexity is polynomial in k, most probably quadratic. To check this, several polynomials of increasing order are estimated using the least squares criterion. The variance of the error is a good statistic to determine the optimal degree of the polynomial. The error variance for polynomials of increasing degree is shown in table 6.3. The polynomials in this table were made monic (neglecting the sign of the highest order coefficient). From degree 2 on, this variance is small and it is smallest for a degree of 3. However, a closer look at the coefficients of the third order polynomial shows a highest order coefficient that is negative. This is very unlikely. Therefore the second order approximation is preferred. This would mean that the time complexity in k is indeed quadratic. This constatation is confirmed when looking at the plot of $t(k)/k$ on a standard scale as shown in fig. 6.13. A straight line can be observed (for big values of k). From these constatations, we can conclude that the time complexity of the search as function of the number of requested nearest neighbors is very likely to be $O(k^2)$.

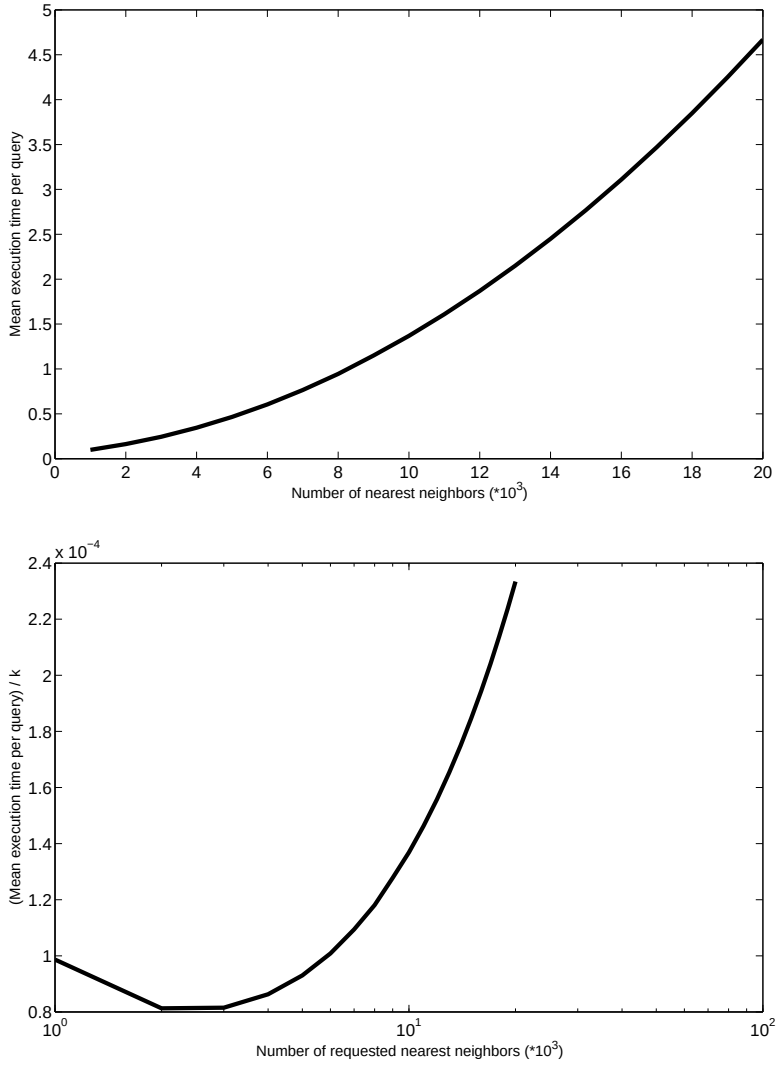


Figure 6.12: (a) Mean execution times for increasing k. (b) Log-plot showing k on the x-axis and $t(k)/k$ on the y-axis.

x	σ^2 (variance)	S (error)	polynomial
1	$2.17 \cdot 10^{-1}$	$1.74 \cdot 10^{0}$	$x - 2.92$
2	$1.09 \cdot 10^{-4}$	$7.64 \cdot 10^{-4}$	$x^2 + 3.70 \cdot x + 4.83$
3	$7.68 \cdot 10^{-5}$	$4.61 \cdot 10^{-4}$	$-x^3 + 411 \cdot x^2 + 1138 \cdot x + 2362$
4	$9.21 \cdot 10^{-5}$	$4.60 \cdot 10^{-4}$	$-x^4 - 147 \cdot x^3 + 7717 \cdot x^2$ $+217425 \cdot x + 443432$
5	$9.53 \cdot 10^{-5}$	$3.81 \cdot 10^{-4}$	$x^5 + 52.7 \cdot x^4 + 957 \cdot x^3 + 11049 \cdot x^2$ $+81020 \cdot x + 84956$
6	$1.24 \cdot 10^{-4}$	$3.72 \cdot 10^{-4}$	...

Table 6.3: Error variance for increasing polynomial degree

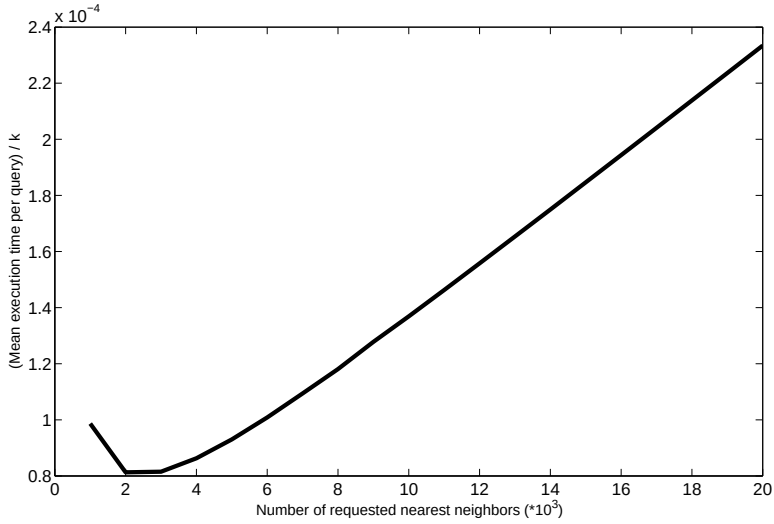


Figure 6.13: Plot showing k on the x-axis and $t(k)/k$ on the y-axis

6.5 Discussion and conclusion

The k-nearest neighbor problem still is a very time-consuming search problem. This seems an intrinsic problem of nearest neighbor searches. The main problem with k-nearest neighbors in high dimensions is the so-called "curse of dimensionality" that causes time complexity factors exponential in the dimension of the search space in the complexity of the search algorithms. Many attempts have been made to overcome this exponential dependence. However, no attempt has succeeded in eliminating the "curse". Even a relaxation of the problem: ϵ-approximate nearest neighbors did not improve performance very much (the exponential factor is still there). The introduction of priority search has a significant impact on the performance of the search, at least for the databases used in this study. The cell aspect ratio seems a less important feature when comparing performance statistics.

The extended roadmap search algorithm on the other hand has linear complexity in the number of dimensions and logarithmic behavior in database size, making it much more useful in high dimensions.

As a summary, a comparison in complexity of different search algorithms is shown in table 6.5. The roadmap search algorithm has much better properties than the kd-tree algorithms. It does not suffer from the curse of dimensionality. However, the difference with standard brute force search is small. The overhead in the roadmap algorithm lies mainly in the creation of the data structure. However, the data structure has to be reconstructed every time the distance measure changes. Because of this, brute force search has been used for most experiments described in this dissertation.

	D	N	k
Kd-tree	exponential	logarithmic	quadratic
Roadmap	cubic [23]	logarithmic	linear
Brute Force	linear	linear	linear

Table 6.4: Comparison of time complexity of different k-nearest neighbor search algorithms

Chapter 7

Experimental results

In this chapter, the results of some experiments combining the methods discussed throughout chapters 2 to 6 will be discussed. We will apply the concept of class dependent distance measures to typical problems in pattern recognition in § 7.1. A second set of experiments presents results from applying class dependent distance measures to template based speech recognition.

7.1 Pattern classification

7.1.1 Experiment setup

In the first set of experiments, the class dependent distance measures are applied on problems in pattern recognition. The databases used for these experiments are a selection of problems from the UCI repository [7] and from STATLOG [63]. A brief summary of the databases used for the experiments is shown in table 7.1. In this table, the first column contains the official name of the database, in the second column, the number of training examples N is given. The third column contains the number of classes of the database N_c and the last column shows the dimension of the data. For a more in detail description of the databases, we refer to appendix B.

Each of these databases contains a set of training examples, consisting

	N	N_c	D
diabetes	768	2	8
balance	625	3	4
dna	3186	3	180
ionosphere	200	2	34
liver	345	2	6
satimage	4435	6	36
vehicle	846	4	19

Table 7.1: UCI and STATLOG databases

of D-dimensional feature vectors. The class-dependent distance measure of equation (3.6) is used. Experiments are performed with both diagonal and block-diagonal weight matrices. If the dataset contains categorical features, features with only a discrete set of possible values (e.g. gender, blood type, ...), these categorical features are replaced by a set of binary features: one for each possible value of the categorical feature. This is done to be able to model the distance between categorical features in the same model, a normal Euclidean or class-dependent distance measure between categories does not make sense.

The weights of the distance measure are trained on a set of criteria defined in chapter 5. For the local Mahalanobis distance, sample mean and variance are used. Discriminative training of the weights is performed using the minimum classification error criterion applied on the nearest neighbor density function. Furthermore, training of the weights is done using K-fold cross validation (CV). Classification results are the mean classification error of each cross validation run. If enough data is available, 10-fold CV is used, otherwise training is performed with 5-fold CV.

7.1.2 Experimental results

In the first set of experiments, the effect of using diagonal and block diagonal weight matrices is investigated. Experiments are performed on every database described in table 7.1. Classification is performed by a simple

	Eucl	**Diag**	**Block diag**	**Rel. impr.**	
	1	2	3	$1 \leftrightarrow 3$	$2 \leftrightarrow 3$
diabetes	32.03	30.99	29.51	7.8%	4.7%
balance	25.26	18.52	16.12	36.2%	12.9%
dna	23.44	4.91	4.52	79.0%	5.8%
ionosphere	18.06	15.23	11.92	34.0%	7.9%
liver	39.13	33.33	26.09	33.2%	21.7%
satimage	10.55	12.01	11.42	−8.2%	4.9%
vehicle	37.23	30.61	28.25	24.1%	7.7%

Table 7.2: Mean classification error on test sets.

1-nearest neighbor classifier. Results of these experiments are shown in table 7.2. The second column shows baseline experiments with a Euclidean distance measure. These results are consistent with other results from the literature [71, 72]. The second and third column show the classification error with a class dependent distance measure. Weights are trained using the discriminative MCE criterion. The final two columns show the relative improvement of a block diagonal distance measures compared to a Euclidean and diagonal distance measure.

One can observe that significant relative improvements are obtained when using block diagonal weight matrices instead of diagonal ones. However, there are some pitfalls in this comparison. The results shown in table 7.2 are results obtained when using the exact data set from the database. I.e. there was no compensation for possible global correlations in the data. It is clear however that these experimental results can be improved upon by including a global decorrelation transformation of the data prior to weight training and classification experiments. Moreover, also the number of parameters with a correlation above the threshold will decrease as these global correlations will no longer be present. As a result the threshold itself (introduced to keep the number of parameters under control) can be decreased, allowing for a more accurate description of the remaining correlations in each class.

	LDA Eucl	**LDA Diag**	**Block diag**	**Rel. impr.**	
	1	2	3	$1 \leftrightarrow 3$	$2 \leftrightarrow 3$
diabetes	31.51	29.81	29.32	7.0%	1.6%
balance	20.10	18.35	16.14	19.7%	12.0%
dna	6.12	4.80	4.50	26.5%	6.3%
ionosphere	15.46	12.21	11.98	22.5%	1.9%
liver	34.12	29.71	26.40	22.6%	11.1%
satimage	11.04	12.20	11.45	−3.7%	6.1%
vehicle	32.12	30.71	28.02	12.8%	8.8%

Table 7.3: Mean classification error on test sets.

In a second set of experiments, the same experiments are performed, but this time a global decorrelation of the dataset is performed in advance. Decorrelation is performed using LDA. Results of these experiments after LDA are shown in table 7.3. The baseline experiment is again a Euclidean distance, but this time on the decorrelated data.

We can observe that results when using both Euclidean distance and distance with diagonal weight matrices improve significantly on most data sets. The results with block diagonal weight matrices however remain more or less the same. This is mainly due to the fact that most within-class correlations were already modelled by the block diagonal weight matrices before the global decorrelation.

7.2 Speech recognition

The results of the distance measures on pattern recognition databases discussed in the previous section were only preliminary experiments. They were used as a proof of concept of the potential of the developed distance measures. The ultimate goal of the creation of these distance measures however is to improve speech recognition accuracy in the context of template based speech recognition.

In this section we will use a large number of distance measures us-

ing the different modelling and training strategies discussed in previous chapters. First, some details of the experimental set-up are presented, followed by experimental results with both frame-based and unit-based distance measures.

7.2.1 Experimental set-up

For the speech recognition experiments, the distance measures are evaluated on two commonly used benchmark databases. The first is "Resource Management" which is a small vocabulary task containing spoken commands in a naval environment [79]. The second benchmark is the Wall Street Journal medium vocabulary database with the Nov 92 test set [73]. For a brief discussion of these benchmarks, we refer to appendix B.

For all recognition experiments, the template based speech recognition system discussed in chapter 2 is used. All templates are phoneme-based and are constructed using a forced alignment of the training material using HMM acoustic models as a bootstrap.

In the same way as De Wachter et al. [20], template transition costs are optimized on development sets. For details of the template transition costs the reader is referred to De Wachter et al. [20].

The recognition accuracy is measured using two kinds of experiments. Standard speech recognition accuracy is measured using the word error rate (WER). The WER is the ratio between the number of word substitutions, insertions and deletions and the total number of words. The second type of experiments are phoneme recognition experiments. In these type of experiments, the phone error rate is used as accuracy measurement.

Preprocessing of the speech is performed by the standard MFCC preprocessing described in § 1.3. The MIDA transform ensures global correlations of the data are no longer present.

Unless otherwise mentioned, the classes for the distance measure are based on the HMM state numbers after forced alignment of the training material using context-dependent phonetic decision trees as explained in § 4.1.2.

	Eucl.	LMah	MCE D	MCE BD
RM feb89	2.95%	2.34%	2.45%	2.40%
RM oct89	3.42%	3.01%	3.20%	3.12%
RM feb91	4.07%	2.61%	2.40%	2.55%
RM sep92	5.78%	4.72%	4.92%	4.39%
mean	4.06%	3.17%	4.24%	3.12%

Table 7.4: WER on RM with frame-based distance measures

7.2.2 Frame-based distance measures

In the first set of experiments, frame based distance measures are used for weight training and recognition experiments. Experiments are performed using the local Mahalanobis (LMah) distance and frame-based distance measure based on minimum classification error. Furthermore, for the discriminative trained distance measure, both diagonal and block diagonal weight matrices are used. For the block diagonal weight matrices a threshold for the correlation was experimentally chosen. The threshold was chosen in such way that the number of parameters in the distance matrices is at most $3 \times D$, with D the dimension of the feature vectors.

In table 7.4, performance of the frame-based distance measures on the RM database in terms of WER is shown. The column "MCE D" shows results with a diagonal discriminative distance measure, "MCE BD" is the block diagonal one. Note that the "MCE D" distance is not able to improve upon the LMah distance measure. However, when block diagonal weight matrices are used, results improve slightly.

In table 7.5, the same experiments are performed on the WSJ5k database with the nov92 evaluation set. Note that also in this case, there is only a slight improvement with the "MCE BD" distance measure.

The conclusion drawn from this set of experiments is that, although frame-based discriminative distance measures are able to improve nearest neighbor classification, they are not able to improve speech recognition results when compared to the maximum likelihood based local Mahalanobis distance measure.

	WER
Eucl	9.8%
Loc Mah	9.4%
MCE B	9.4%
MCE BD	9.1%

Table 7.5: WER on WSJ5k with frame-based distance measures

7.2.3 Unit-based distance measures

In the previous set of experiments, frame based distance measures were used. The criterion to train these distance measures however does not entirely reflect the way speech recognition works. In this section we will examine the unit-based distance measures developed in § 5.7. In particular, we will use phoneme based training of the distance measures with both the MCE and MMI criteria.

In this case, there is a difference between classes of the distance measure, defined in § 3.3.2 and the classes of the pattern classifier defined in chapter 5. In order to have sufficient modelling capabilities for the distance measures, the classes for these are constructed in the same way as for the frame-based distance measures. The class labels for the classification task however consist only of the context-independent phoneme labels. The motivation behind this decision is that a correct phoneme sequence with a wrong context also leads to correct recognition in template based speech recognition. This is unlike HMM-based speech recognition, where it is impossible to correctly recognize a phoneme sequence when the context is wrong. Moreover, the context mismatch model developed by De Wachter et al. [20] somehow compensates for wrong context in recognition.

First, performance on the RM task will be investigated. The results of these experiments are shown in table 7.6. For easy comparison the results of experiments with the Euclidean and Local Mahalanobis distance measures is repeated.

The same experiment is repeated for the WSJ5k database. In this case also the phone recognition accuracy is included. The phone error

	Eucl.	LMah	MCE D	MCE BD	MMI D	MMI BD
RM feb89	2.95%	2.34%	2.35%	2.25%	2.30%	2.27%
RM oct89	3.42%	3.01%	3.10%	2.85%	3.20%	3.01%
RM feb91	4.07%	2.61%	2.42%	2.29%	2.61%	2.35%
RM sep92	5.78%	4.72%	4.82%	4.30%	5.00%	4.89%
mean	4.17%	3.17%	3.17%	2.92%	3.28%	3.13%

Table 7.6: WER on RM with unit-based distance measures

	WER	**PER**
Eucl	9.8%	22.0%
Loc Mah	9.4%	19.1%
MCE B	9.5%	20.1%
MCE BD	8.9%	17.5%

Table 7.7: WER and PER on WSJ5k with unit-based distance measures

rate is more directly related to the training criterion. The results of these experiments are shown in table 7.7.

The logical next step to perform is to perform sentence-based discriminative training of the distance measures, as currently occurs in most discriminative training strategies for HMM based speech recognition. However, the large computational requirements, discussed in § 5.7.2 caused us to make too many simplifications to the model. It is nearly impossible to list all possible phone strings or word strings since the number of strings that differ only in a single template (phone) is already huge. Experiments with simplified situations, in which a lot of these alternative hypotheses were not considered did not lead to good results.

	RM avg.	WSJ nov92
TB LMah	3.07%	9.8%
TB Discr.	2.72%	9.1%
HMM Spraak ML	3.35%	5.10%
HMM HTK ML [33]		8.41%
HMM HTK Discr. [33]		8.05%

Table 7.8: Comparison between template based and HMM based speech recognition

7.2.4 Comparison with HMM based speech recognition

In this final section, we will compare the presented results to those of a similar HMM-based speech recognition. As explained in this text, also HMM's can be trained based on ML or discriminative training. For the ML experiments, the ESAT Spraak system is used [26]. The Spraak system however only contains subroutines for ML training, therefore HMM results with discriminative training will be borrowed from the literature.

The results of this study are presented in table 7.2.4. The table contains two entries for the ML baseline. The first is the performance of the ESAT Spraak system, the second is borrowed from the literature and generated with the Hidden Markov Model Toolkit (HTK). This HTK baseline was found in the literature [51, 33]. This baseline system consists of an HMM system with 2329 shared states and observation densities modeled by an 8-mixture GMM. The features consisted of 12 MFCC's together with energy and their first and second order derivatives. The used language model is a bigram. The result of discriminative training is borrowed from [33], which used MCE training on a generalized posterior upon the HTK baseline system.

De Wachter et al. have shown that combining DTW and HMM approaches may lead to an increase of recognition accuracy. For details on these experiments, we refer to [21, 20]

Chapter 8

Conclusions

This chapter concludes the work presented in this dissertation. It presents an overview of the contributions in this dissertation. Furthermore an overview of the experimental results is given and the chapter ends with some pointers to possible further research.

8.1 Main contributions

The research for this dissertation concentrated on the subject of distance measures for template based speech recognition. Traditional DTW-based speech recognition uses the (squared) Euclidean distance to compare feature vectors. We have introduced new distance measures to replace this Euclidean distance. We have tested these new distance measures on problems from pattern classification and on template based speech recognition.

A second topic of the research concerns k-nearest neighbor search algorithms. We performed a comparative study on different algorithms investigating the time complexity as a function of the number of prototypes, the number of dimensions and the number of requested nearest neighbors.

Class dependent distance measures

The creation of class dependent distance measures itself is not a new idea. However, the introduction of these class dependent distance measures in

DTW-based speech recognition is a novel idea unseen before. The research however the work presented by different other authors. Research on this topic focused mainly on the definition of the classes. Many different class definition procedures have been proposed. They were based on supervised and unsupervised clustering techniques.

The first approaches were based on supervised clustering techniques. With forced alignment, a separation of the example data into phonemes is generated. An initial attempt to define classes was based on these phoneme labels. These classes however were unable to improve recognition accuracy, most likely because they are too coarse grained. A second was approach based on forced alignment using context dependent HMM states generated by a phonetic decision tree. Performance results of speech recognition based on this type of classes showed an increase in recognition accuracy.

A second set of class division procedures were based on unsupervised clustering. In particular, the k-means, ISODATA and some hierarchical clustering algorithms have been implemented and applied to the speech recognition data. However, unlike supervised clustering, recognition accuracy did not increase with this form of clustering.

In a third set of methods, supervised and unsupervised clustering methods were combined. An initial separation of the example data was made using supervised clustering techniques. We tried to improve upon this partition by further separating the data using unsupervised clustering. Experiments with this combination however did not show an increase of performance when compared with a partitioning based only on supervised clustering.

Formulation of the distance measures

The main formulation used to define the class dependent distance measures is the one found in equation (3.6). This definition introduces a class dependent weight matrix Λ_c. Several alternative approaches to define these weight matrices have been proposed.

First experiments were based on distance measures with diagonal weight matrices. These distance measures have the advantage of fast computabil-

ity. However, they lack the ability to model correlations present in the training data. Global correlations in the training data can be removed by component analysis or discriminant analysis techniques. However, this does not ensure that within-class correlations are removed too.

In order to model these correlations, full weight matrices were introduced. These full weight matrices can model any linear correlation present in the example material. However the introduction of full matrices increased the distance calculation complexity from linear to quadratic. Since a lot of these distance calculations are performed throughout both training and recognition, this is problematic. Training of the distance measures became unmanageable.

The final contribution in this regard is the introduction of block diagonal distance measures. It was experimentally verified that correlations in many combinations of dimensions are negligible. Therefore, weight matrices that only contained a parameter in a specific position if there is a correlation between the respective dimensions were constructed. A threshold on the correlation ensures that the number of parameters is kept under control and that distance measures can be calculated by an algorithm with linear complexity. The introduction of these sparse weight matrices is a novel idea unseen before.

Training of distance measures

The third and final main contribution in the context of class dependent distance measures is the creation of training procedures to define the weights of the distance matrices. An original maximum likelihood approach was presented by De Wachter et al. [20]. This work focused on discriminative training methods to train the weights. Several discriminative training techniques created in HMM research were adapted and combined with the class dependent distance measures. In particular maximum mutual information and minimum classification error were investigated. The use of discriminative training methods is common in HMM-based speech recognition. We have proposed a novel method to use these techniques to train the weights of the class dependent distance measure. Training of the distance measures was performed with frame-based and unit-based training

procedures.

In frame-based training procedures, the discriminative criterion is applied on nearest neighbor classification of the speech frames. These frame-based distance measures caused an increase in performance compared to the Euclidean distance measures. However, they were unable to improve upon the maximum likelihood distance measure. The main explanation is the mismatch between nearest neighbor classification and speech recognition, where distance between templates is more important than distance between individual frames.

Therefore, unit-based distance training procedures were created, based on the DTW score between different units. Distance trained with these distance measures did show an increase in performance when compared with both the Euclidean and ML approaches.

8.2 Main experimental results

Experiments with these distance measures were performed on both pattern classification and template based speech recognition. In pattern classification, classification based on k nearest neighbor classification was investigated. We have compared the discriminative training procedures with a standard Euclidean distance and have shown that our class-dependent distance measures with sparse weight matrices and trained with a discriminative method almost always outperform the method with the Euclidean distance. If we compare our method with other work, the introduction of sparse weight matrices also shows a consistent improvement in classification accuracy.

In speech recognition, the problem at hand is much more complex. Techniques that seemed to work well in pattern classification did not always work in template based speech recognition. However, using a combination of techniques, baseline results based on Euclidean and maximum likelihood distance measures were improved upon. The best working combination was found in class dependent distance measures using block diagonal weight matrices and based on a unit-dependent minimum classification error training criterion.

The combination of the experiments on both pattern classification and

template based speech recognition show that our novel approach is a valid approach with a true potential to improve recognition accuracy.

Finally, results from template based speech recognition were compared with results from HMM based speech recognition. This comparison still shows a gap between the performance of typical HMM systems and our template based approach. Though somewhat disappointing, results still show a fair recognition accuracy for template based speech recognition.

In the margin of the research, we have performed a comparative study on k-nearest neighbor algorithms. We have experimentally shown that most algorithms suffer from the curse of dimensionality, causing a bad behavior of the time complexity in terms of the number of dimensions. If the number of dimensions is high, a brute force search almost always outperforms the other algorithms.

The presented work has many opportunities for improvement of recognition accuracy. A small overview is presented in the next section.

8.3 Pointers for further research

Research in this domain is not finished. The main problem remains the computational complexity of the different training and recognition algorithms. Due to this computational complexity, there still is a mismatch between the discriminative training criterion and the speech recognition performance measures. The main possible improvements upon our work presented concern the improvement of the training procedures for the weight matrices. Recent research on large margin estimation and soft margin estimation show a small improvement in HMM based speech recognition.

The true recognition accuracy is measured using word error rate on sentences. It is natural to compute the discriminative criteria based on these sentences. This already occurs in the context of discriminative training of HMM's. However, discriminative training techniques need the formulation of a lot of alternative wrong hypotheses. In HMM based discriminative training, this is usually done using N-best lists or word graphs. In template based speech recognition however, the production of these alternative hypothesis is a much more complex problem. The number of

possible alternative hypotheses is huge compared to the number of alternatives in HMM training. Moreover, the correct solution is no longer unique. When including these observations in discriminative training on a template based system, the sum over all alternatives in the discriminative training criterion becomes huge and unmanageable.

In this research, this was avoided by only using unit based training criteria and by restricting the number of alternative hypothesis using only the nearest neighbor in each class. This however is an oversimplification of the problem and does not reflect the real discriminative training criterion. Further research on this topic will almost certainly lead to an increase in recognition accuracy if a method is found to make the criterion and formulation of alternative approaches manageable.

Another interesting pointer for further research is the recent introduction of large and soft margin estimation techniques, a set of discriminative training procedures that seem to work well with HMM-based speech recognition. It would be interesting to see if these ideas can be transferred to the training of the weights and what the corresponding results are.

Finally, the results with template based speech recognition still show a performance gap with HMM-based speech recognition. However, a countless number of possible explanations can be formulated, each being a valid subject for investigation. Several possible explanations have been formulated in this chapter, other ones still need to be discovered. Research on this topic is not finished.

Appendix A

The Sparse BLAS

The Sparse Basic Linear Algebra Subprograms (BLAS) describe kernel operations for unstructured, sparse vectors and matrices. Sparse matrices are by definition matrices in which most of the matrix elements are zero, as opposed to dense matrices. The standards, as defined by the BLAS Technical Forum, have interfaces available in different programming languages, among which C, Fortran 77 and Fortran 95. These standards are described by Duff et al. [29]. The Sparse BLAS subroutines are not useful for dense vector and matrix operations. Other standards such as dense BLAS and Lapack exist for such purposes.

The Sparse BLAS supports three levels of operations. Level 1 involves operations with only vectors. An overview of these operations is found in table A.1. Level 2 involves matrix-vector operations. These are listed in table A.2. Finally, level 3 is concerned with matrix-matrix operations. These list of supported operations is shown in table A.3.

With these basic operations, it is possible to perform the distance calculations as described in chapter 3. A small example of how to use these operations for this purpose with the C interface is shown in figure A.1. The main operations are shown on lines 5, 6 and 7, respectively subtracting vector $\mathbf{y}$ from $\mathbf{x}$, multiplying it with the weight matrix of the class to which $\mathbf{y}$ belongs and multiplying it with the vector $\mathbf{x} - \mathbf{y}$ again. The *d*-prefix for the operations makes sure the operations are performed in double precision.

name	description	operation
USDOT	sparse dot product	$r \leftarrow \mathbf{x}^H\mathbf{y}$
USAXPY	sparse vector update	$\mathbf{y} \leftarrow \alpha\mathbf{x} + \mathbf{y}$
USGA	sparse gather	$\mathbf{y} \leftarrow \mathbf{y}\|_{\mathbf{x}}$
USGZ	sparse gather and zero	$\mathbf{x} \leftarrow \mathbf{y}\|_{\mathbf{x}}; \mathbf{y}\|_{\mathbf{x}} \leftarrow 0$
USSC	sparse scatter	$\mathbf{y}\|_{\mathbf{x}} \leftarrow \mathbf{x}$

Table A.1: Sparse BLAS level 1 operations

name	description	operation
USMV	sparse matrix-vector product	$\mathbf{y} \leftarrow \alpha A\mathbf{x} + \mathbf{y}$ $\mathbf{y} \leftarrow \alpha A^T\mathbf{x} + \mathbf{y}$ $\mathbf{y} \leftarrow \alpha A^H\mathbf{x} + \mathbf{y}$
USSV	sparse triangular solve	$\mathbf{x} \leftarrow \alpha A^{-1}\mathbf{x}$ $\mathbf{x} \leftarrow \alpha A^{-T}\mathbf{x}$ $\mathbf{x} \leftarrow \alpha A^{-H}\mathbf{x}$

Table A.2: Sparse BLAS level 2 operations

name	description	operation
USMM	sparse matrix-matrix multiplication	$C \leftarrow \alpha AB + C$ $C \leftarrow \alpha A^T B + C$ $C \leftarrow \alpha A^H B + C$
USSV	sparse triangular solve	$B \leftarrow \alpha T^{-1} B$ $B \leftarrow \alpha T^{-T} B$ $B \leftarrow \alpha T^{-H} B$

Table A.3: Sparse BLAS level 3 operations

```
1.  double weighteddist(int x,int y,int classy) {
2.    double dis=0.0,d;
3.    IniVecFloat(temp_vec1, dim, 0);
4.    cblas_dcopy(dim, trainingset[x], 1, vector2, 1);
5.    cblas_daxpy(dim, -1, trainingset[y], 1, vector2, 1);
6.    BLAS_dusmv(blas_no_trans, 1.0, weights[classy],
                 vector2, 1.0, temp_vec1, 1.0 );
7.    dis = cblas_ddot(dim, vector1, 1, vector2, 1);
8.    dis=pweights[y]*sqrt(dis);
9.   return dis;
10. }
```

Figure A.1: Sparse BLAS distance calculation

Appendix B

Benchmark databases

In this appendix, the pattern recognition databases that are employed for the classification experiments of chapter 7 are briefly described. The UCI and STATLOG repositories contain well known benchmark databases for pattern classification algorithms.

B.1 Pattern classification

Diabetes

The diabetes data set is a UCI database containing measurements performed on Pima Indians in order to predict the possible onset of diabetes. The prototypes are 8-dimensional and include following measurements:

- number of times pregnant,
- plasma glucose concentration at 2 hours in an oral glucose tolerance test,
- diastolic blood pressure,
- triceps skin fold thickness,
- 2-hour serum insulin,

- body mass index,
- diabetes pedigree function,
- age.

The target variable is the onset of diabetes disease. There are 768 instances available in the database.

Balance

The balance data set also is included in the UCI repository. The classification objective in this case is the prediction if the balance flips to the left, to the right or is balanced based on 4 measurements:

- left weight,
- left distance,
- right weight,
- right distance.

There are 625 instances available in the database.

DNA

The DNA database contains categorical information on DNA strings, the classification goal is to predict that a particular DNA string is either an exon or an intron or neighter of both. The data contain DNA sequences of proteins. These proteins are labelled categorically. The resulting binary prototypes are 180-dimensional and 3186 examples are available.

Ionosphere

This database contains radar measurements from the ionosphere. Those measurements are either "good", showing some type of structure, or "bad" showing no structure at all. This is the classification task. The measurement data vectors are 34-dimensional. There are 351 instances available in the database, subdivided in to 200 training and 151 test instances.

Liver

The aim of this classification problem is to detect liver disorders due to alcohol consumption based on blood measurements and alcohol consumption patterns from patients. There are 345 instances available each containing 6 features from individual patients. The first 5 measurements are blood measurements and the last one is the mean alcohol consumption in number of half-pint drinks per day.

Satimage

The Satimage data set is a database from Statlog. It contains multi-spectral values of pixels in a 3x3 neighborhood in a satellite image. The measurements are performed in 4 spectral bands leading to 36-dimensional feature vectors containing numeric values in the range 0–255. The goal is to classify the soil in the central pixel. There are 7 different classes of soil defined:

- red soil,
- cotton crop,
- grey soil,
- damp grey soil,
- soil with vegetable stubble,
- mixture class (all types present),
- very damp grey soil.

There are 6435 examples in the databases subdivided into 4435 training examples and 2000 test examples.

Vehicle

The Vehicle problem tries to classify vehicles based on measurements of their silhouette. There are 19 measurements performed on the image of a vehicle:

- compactness,
- circularity ((average radius)**2/area),
- distance circularity (area/(av.distance from border)**2),
- radius ratio ((max.rad-min.rad)/av.radius),
- principal axis aspect ratio ((minor axis)/(major axis)),
- maximal length aspect ratio ((length perp. max length)/(max length)),
- scatter ratio ((inertia about minor axis)/(inertia about major axis)),
- elongatedness (area/(shrink width)**2),
- principal axis rectangularity (area/(pr.axis length*pr.axis width)),
- maximal length rectangularity (area/(max.length*length perp. to this)),
- scaled variance along major axis ((2nd order moment about minor axis)/area),
- scaled variance along minor axis ((2nd order moment about major axis)/area),
- scaled radius of gyration ((mavar+mivar)/area),
- skewness about major axis ((3rd order moment about major axis)/sigma_min**3),
- skewness about minor axis ((3rd order moment about minor axis)/sigma_maj**3),
- kurtosis about minor axis ((4th order moment about major axis)/sigma_min**4),
- kurtosis about major axis ((4th order moment about minor axis)/sigma_maj**4),
- hollows ratio ((area of hollows)/(area of bounding polygon)),

There are four classes of vehicles: opel, saab, bus and van. There are 846 examples available partitioned in 9 sets of 94 examples for 9-fold cross validation.

B.2 Speech recognition

TiDigits

The TIDigits database was designed at Texas Instruments. Its purpose is evaluating algorithms for speaker-independent recognition of connected digit sequences. The vocabulary contains 11 words: zero, oh and the digits one to ten. The speech is obtained from 326 speakers, from which 111 men, 114 women, 50 boys and 51 girls.

We have used this database to compare algorithms for k-nearest neighbor searches. We have only used the men and women data. Preprocessing of the speech in the database leads to 1516015 feature vectors in 25 dimensions.

Resource management

The Research Management database for speech recognition from DARPA is a database for medium-vocabulary sized speech recognition [79]. The problem setting is the automatic interaction between humans and a system for naval resource management. The vocabulary contains 1000 words.

The speech data itself is obtained from native English speakers from 4 different geographic regions, with different gender (70% male, 30% female). The set of speakers is subdivided into a set of training speakers and different sets of test speakers. There are 109 training speakers each reading 57 sentences, about 4 hours of speech. There are 1600 available sentences, and the distribution of the sentences among speakers occurred randomly.

There are four test sets, labelled "feb89", "oct89", "feb91" and "sep92" each containing 30 sentences from 10 speakers. The speakers used for the test sets are other speakers than those used for the training set.

The phone set and lexicon are derived from the CMUv0.4 phonetic lexicon.

The templates for TBSR are based on the phones after an automatic segmentation of the speech database with a HMM system. There are 155888 templates available in the training material.

	N	N_c	D
Resource Management	155888	47	25
Wall Street Journal 5k	592223	43	25

Table B.1: Speech recognition databases

Wall street journal

The Wall Street Journal speech recognition database was first released in 1992 by Paul et al. [73]. The database contains read speech from Wall Street Journal newspaper articles. We only used the "5k" medium-sized vocabulary part of the database with a closed dictionary containing about 5000 words.

The training material is obtained from 84 different speakers with in total about 15 hours of available training material. The test material contains a development set called "dev92" containing 410 sentences and a test set called "nov92" containing 330 sentences from 10 different speakers each.

The applied phone set contains 43 phones and the lexicon is derived from the CMUv0.6d phonetic lexicon.

The templates for TBSR are based on the phones after an automatic segmentation of the speech database with a HMM system. There are 1884836 templates available in the training material.

An overview of the problem size of both speech recognition databases is shown in table B.1.

Bibliography

[1] Shunichi Amari. A theory of adaptive pattern classifiers. *IEEE Transactions on Electronic Computers*, EC-16(3):299–307, June 1967.

[2] Guillermo Aradilla, Jithendra Vepa, and Hervé Bourlard. Improving speech recognition using a data driven approach. In *Proceedings of Eurospeech 2005*, pages 3333–3336, Lisboa, September 2005.

[3] Hagai Aronowitz. Text independent speaker recognition using speaker dependent word spotting. In *Proceedings of ICSLP 2004*, Jeju Island, Korea, October 2004.

[4] S. Arya. Ann: Library for approximate nearest neighbor searching v0.2, 1998.

[5] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1998.

[6] S. Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *Proceedings of the 4th Annual ACM Symposium on Discrete Algorithms*, pages 271–280, Austin, January 1993.

[7] A. Asuncion and D.J. Newman. UCI machine learning repository, 2007.

[8] Scott Axelrod and Benoît Maison. Combination of hidden markov models with dynamic time warping for speech recognition. In *Pro-*

ceedings if ICASSP 04, pages 173–176, Montreal, Quebec, Canada, May 2004.

[9] Lahit R. Bahl, Peter F. Brown, Peter V. de Souza, and Robert L. Mercer. Maximum mutual information estimation of hidden markov model parameters for speech recognition. In *Proceedings of ICASSP 86*, pages 49–52, Tokyo, Japan, April 1986.

[10] L.R. Bahl, P.V. de Souza, P.S. Gopalakrishnan, D. Nahamoo, and M.A. Picheny. Decision trees for phonological rules in continuous speech. In *Proceedings of ICASSP 91*, pages 185–188, Toronto, Canada, May 1991.

[11] G.H. Ball and D.J. Hall. Isodata, a novel method of data analysis and classification. Technical report, Stanford University, 1965.

[12] R. Bellman. *Adaptive Control Processes*. Princeton University Press, 1961.

[13] J.L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[14] C. Bohm, S. Berchtold, and D.A. Keim. Searching in high-dimensional spaces - index structures for improving the performance of multimedia databases. *ACM Computing Surveys*, 33(3):322–373, September 2001.

[15] Peter F. Brown. *The Acoustic-modeling problem in automatic speech recognition*. PhD thesis, Carnegie Mellon University, March 1988.

[16] W. Chou, C.-H. Lee, and B.-H. Juang. Minimum error rate training based on n-best string models. In *Proceedings of ICASSP 93*, pages 652–655, Minneapolis, Minnesota, USA, April 1993.

[17] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. John Wiley & Sons, 1991.

[18] K.H. Davis, R. Biddulph, and S. Balashek. Automatic recognition of spoken digits. *Journal of the Acoustical Society of America*, 24(6):637–642, November 1952.

[19] Steven B. Davis and Paul Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(4):357–366, August 1980.

[20] Mathias De Wachter. *Example Based Continuous Speech Recognition*. PhD thesis, Katholieke Universiteit Leuven, May 2007.

[21] Mathias De Wachter, Kris Demuynck, and Dirk Van Compernolle. Boosting hmm performance with a memory upgrade. In *Proceedings of International Conference on Spoken Language Processing*, pages 1730–1733, Pittsburgh, U.S.A., September 2006.

[22] Mathias De Wachter, Kris Demuynck, Dirk Van Compernolle, and Patrick Wambacq. Data driven example based continuous speech recognition. In *Proceedings of EUROSPEECH 03*, pages 1133–1136, Geneva, September 2003.

[23] Mathias De Wachter, Kris Demuynck, Patrick Wambacq, and Dirk Van Compernolle. A locally weighted distance measure for example based speech recognition. In *Proceedings of ICASSP 2004*, pages 181–184, Montreal, Canada, May 2004.

[24] Mathias De Wachter, Mike Matton, Kris Demuynck, Patrick Wambacq, Ronald Cools, and Dirk Van Compernolle. Template based continuous speech recognition. *IEEE Transactions on Audio, Speech and Language Processing*, 15(4):1377–1390, May 2008.

[25] Kris Demuynck, Jacques Duchateau, and Dirk Van Compernolle. Optimal feature sub-space selection based on discriminant analysis. In *Proceedings of EUROSPEECH 99*, pages 1311–1314, Budapest, September 1999.

[26] Kris Demuynck, Jan Roelens, Dirk Van Compernolle, and Patrick Wambacq. Spraak: An open source speech recognition and automatic annotation kit. In *proceedings of Interspeech 2008*, page 495, Brisbane, Australia, September 2008.

[27] Jacques Duchateau. *HMM Based Acoustic Modelling in Large Vocabulary Speech Recognition.* PhD thesis, K.U.Leuven, ESAT, November 1998.

[28] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification.* Wiley-Interscience, 2001.

[29] I. Duff, M. Heroux, and R. Pozo. An overview of the sparse basic linear algebra subprograms: The new standard from the blas technical forum. *ACM Transactions on Mathematical Software*, 28(2):239–267, June 2002.

[30] Christian Dugast, Peter Beyerlein, and Reinhold Haeb-Umbach. Application of clustering techniques to mixture density modelling for continuous-speech recognition. In *Proceedings of ICASSP 95*, pages 524–527, Detroit, Michigan, May 1995.

[31] A. Christian Duncan, Michael T. Goodrich, and Stephen Kobourov. Balanced aspect ratio trees: Combining the advantages of k-d trees and octrees. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 300–309, Baltimore, January 1999.

[32] J.H. Friedman, J.L. Bentley, and Finkel R.A. An algorithm for finding best matches in logarithmic expected time. *ACM Transactions on Mathematical Software*, 3(3):209–226, September 1977.

[33] Qiang Fu, Antonio Moreno-Daniel, Biing-Hwang Juang, Jian-Lai Zhou, and Frank Soong. Generalization of the minimum classification error (mce) training based on maximizing generalized posterior probability (gpp). In *Proceedings of Interspeech 2006*, pages 681–684, Pittsburgh, Pennsylvania, September 2006.

[34] Keinosuke Fukunaga and Thomas E. Flick. An optimal global nearest neighbor metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(3):314–318, May 1984.

[35] Sadaoki Furui. 50 years of progress in speech and speaker recognition. In *Proceedings of SPECOM 2005*, pages 1–9, Patras, Greece, October 2005.

[36] V. Gaede and O. Gunther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[37] Xuedong Huang, Alex Acero, and Hsiao-Wuen Hon. *Spoken Language Processing*. Prentice Hall, 1st edition, 2001.

[38] Dumitada Itakura. Minimum prediction residual principle applied to speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):67–72, February 1975.

[39] A.K. Jain and R.C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1st edition, 1988.

[40] A.K. Jain, M.N. Murty, and P.J. Flynn. Data clustering: A review. *ACM Computing Surveys*, 31(3):264–319, September 1999.

[41] Frederick Jelinek. Continuous speech recognition by statistical methods. *Proceedings of the IEEE*, 64(4):532–556, April 1976.

[42] Hui Jiang, Xinwei Li, and Chaojun Liu. Large markov hidden markov models for speech recognition. *IEEE Transactions on audio, speech and language processing*, 14(5):1584–1595, september 2006.

[43] Biing-Hwang Juang, Wu Chou, and Chin-Hui Lee. Minimum classification error rate methods for speech recognition. *IEEE Transactions on Speech and Audio Processing*, 5(3):257–265, May 1997.

[44] Biing-Hwang Juang and Shigeru Katagiri. Discriminative learning vor minimum error classification. *IEEE Transactions on signal processing*, 40(12):3043–3054, December 1992.

[45] Shigeru Katagiri, Chin-Hui Lee, and Biing-Hwang Juang. New discriminative training algorithms based on the generalized probabilistic descent method. In *Neural Networks for Signal Processing, Proceedings of the 1991 IEEE Worshop*, pages 299–308, September 1991.

[46] Benjamin King. Step-wise clustering procedures. *Journal of the Americal Statistical Association*, 62(317):86–101, March 1967.

[47] Ron Kohavi, Pat Langley, and Yeogirl Yun. The utility of feature weighting in nearest-neighbor algorithms. In *Proceedings of the Ninth European Conference on Machine Learning*, pages 85–92. Springer-Verlag, 1997.

[48] R. Koshiba, M. Tachimori, and H. Kanazawa. A flexible method of creating hmm using block-diagonalization of covariance matrices. In *Proceedings of ICSLP 98*, pages 2947–2950, Sydney, Australia, November 1998.

[49] Leonard, R. Gary. A Database for Speaker Independent Digit Recognition. In *Proceedings of the IEEE ICASSP'84*, pages 328–331, San Diego, March 1984.

[50] J.E. Levinson. Continuously variable duration hidden markov models for automatic speech recignition. *Computer Speech and Language*, 1(1):29–45, March 1986.

[51] Jinyu Li, Marco Sabato Marco, Siniscalchi, and Lee Chin-Hui. Approximate test risk minimization through soft margin estimation. In *Proceedings of ICASSP 2007*, pages 653–656, Honolulu, Hawaii, USA, April 2007.

[52] Jinyu Li, Yuan Ming, and Chin-Hui Lee. Soft margin estimation of hidden markov model parameters. In *Proceedings of Interspeech 2006*, pages 2422–2425, Pittsburgh, Pennsylvania, USA, september 2006.

[53] Wolfgang Macherey, Lars Haferkamp, Ralf Schlüter, and Hermann Ney. Investigations on error minimizing training criteria for discriminative training in automatic speech recognition. In *Proceedings of interspeech 2005*, pages 2133–2136, Lisbon, Portugal, September 2005.

[54] Jinchang Mao and Anil K. Jain. A self-organizing network for hyperellipsoidal clustering. *IEEE Transactions on Neural Networks*, 7(1):16–29, January 1996.

[55] Jean-Francois Mari and Jean-Paul Haton. Automatic word recognition based on second-order hidden markov models. *IEEE Transactions on Speech and Audio Processing*, 5(1):22–25, January 1977.

[56] Mike Matton and Ronald Cools. A comparison of k-nearest neighbour algorithms with performance results on speech data. Technical Report TW-381, KULeuven, department of Computer Science, January 2004.

[57] Mike Matton, Mathias De Wachter, Dirk Van Compernolle, and Ronald Cools. A discriminative locally weighted distance measure for speaker independent template based speech recognition. In *Proceedings of ICSLP 04*, pages 429–432, Jeju, Korea, October 2004.

[58] Mike Matton, Mathias De Wachter, Dirk Van Compernolle, and Ronald Cools. Maximum mutual information training of distance measures for template based speech recognition. In *Proceedings of International Conference on Speech and Computer*, Patras, Greece, October 2005.

[59] Mike Matton, Dirk Van Compernolle, and Ronald Cools. A minimum classification error based distance measure for template based speech recognition. In *Proceedings of Interspeech 2008*, Australia, September 2008.

[60] Mike Matton, Dirk Van Compernolle, and Ronald Cools. Minimum classification error training in example based speech and pattern recognition using sparse weight matrices. *Journal of Computational and Applied Mathematics*, 2009. submitted.

[61] Erik McDermott. *Discriminative training for speech recognition*. PhD thesis, Waseda University, Graduate school of science and engineering, March 1997.

[62] James McNames. A fast nearest-neighbour algorithm based on a principal axis search tree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(9):964–976, September 2001.

[63] Donald Michie, D. J. Spiegelhalter, C. C. Taylor, and John Campbell, editors. *Machine learning, neural and statistical classification.* Ellis Horwood, Upper Saddle River, NJ, USA, 1994.

[64] J. Ming and Smith F. Modelling of the interframe dependence in an hmm using conditional gaussian mixtures. *Computer Speech and Language*, 10(4):229–247, October 1996.

[65] Geert Monsieur. Zoekrobot voor audio. Master's thesis, Katholieke Universiteit Leuven, Leuven, Belgium, June 2005.

[66] Cory Myers, Lawrence R. Rabiner, and Aarin E. Rosenberg. Performance tradeoffs in dynamic time warping algorithms for isolated word recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 28(6):623–635, December 1980.

[67] Arthur Nádas, David Nahamoo, and Michael A. Picheny. On a model-robust training method for speech recognition. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 36(9):1432–1436, September 1988.

[68] Y. Normandin. Maximum mutual information estimation of hidden markov models. In Chin-Hui Lee, Frank K. Soong, and Kuldip K. Paliwal, editors, *Automatic Speech and Speaker Recognition*, chapter 3. Kluwer Academic Publishers, 2nd edition, 1997.

[69] Yves Normandin. *Hidden Markov Models, Maximum Mutual Information Estimation and the Speech Recognition Problem.* PhD thesis, McGill University, Montreal, Department of Electrical Engineering, March 1991.

[70] Julian James Odell. *The Use of Context in Large Vocabulary Speech Recognition.* PhD thesis, University of Cambridge, March 1995.

[71] Roberto Paredes and Enrique Vidal. A class-dependent weighted dissimilarity measure for nearest neighbor classification problems. *Pattern Recognition Letters*, 21(12):1027–1036, November 2000.

[72] Roberto Paredes and Enrique Vidal. Learning prototypes and distances: A prototype reduction technique based om nearest neighbor error minimization. *Pattern Recognition*, 39:180–188, 2006.

[73] D.B. Paul and J.M. Baker. The design for the wall street journal-based csr corpus. In *Proceedings of ICSLP 92*, Banff, Canada, October 1992.

[74] Karl Pearson. On lines and planes of closest fit to systems of points in space. *Philosophical Magazine*, 2(6):559–572, 1901.

[75] D. Povey and Woodland P.C. Frame discrimination training of hmms for large vocabulary speech recognition. Technical Report CUED/F-INFENG/TR332, Cambridge University Engineering Department, 1999.

[76] D. Povey and P.C. Woodland. Frame discrimination training of hmms for large vocabulary speech recognition. In *Proceedings of ICASSP 99*, pages 333–336, Phoenix, Arizona, USA, March 1999.

[77] D. Povey and P.C. Woodland. Minimum phone error and i-smoothing for improved discriminative training. In *Proceedings of ICASSP 02*, Orlando, Florida, USA, May 2002.

[78] Daniel Povey. *Discriminative Training for Large Vocabulary Speech Recognition*. PhD thesis, University of Cambridge, March 2003.

[79] Patty Price, William M. Fisher, Jared Bernstein, and David s. Pallett. The darpa 1000-word resource management database for continuous speech recognition. In *Proceedings of ICASSP 88*, New York City, USA, April 1988.

[80] Lawrence Rabiner and Bing-Hwang Juang. *Fundamentals of Speech Recognition*. Prentice Hall, 1st edition, 1993.

[81] V. Ramasubramanian, Kaustubh Kulkarni, and Bernhard Kaemmerer. Acoustic modeling by phoneme templates and modified one-pass dp-decoding for continuous speech recognition. In *Proceedings of ICASSP 2008*, Las Vegas, Nevada, USA, April 2008.

[82] W. Reichl and G. Ruske. Discriminative training for continuous speech recognition. In *Proceedings of EUROSPEECH 95*, pages 537–540, Madrid, Spain, September 1995.

[83] Francesco Ricci and Paolo Avesani. Data compression for local metrics for nearest neighbor classification. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(4):380–384, April 1999.

[84] Hiroaki Sakoe and Seibi Chiba. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, Signal Processing*, 26(1):43–49, February 1978.

[85] Hanan Samet. The quadtree and related hierarchical data structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

[86] Ralf Schlüter. *Investigations on Discriminative Training Criteria*. PhD thesis, RWTH Aachen, September 2000.

[87] Ralf Schlüter and Wolfgang Macherey. Comparison of discriminative training criteria. In *Proceedings of ICASSP 1998*, Seattle, Washington, USA, May 1998.

[88] B.W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1st edition, 1986.

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

[90] R.-H. Wang, L.-S. He, and H. Fujisaki. A weighted distance measure based on the fine structure of featurespace: Application to speaker recognition. In *Proceedings of ICASSP 1990*, Albuquerque, New Mexico, USA, April 1990.

[91] Rui Wang, Xuan Zhu, Yining Chen, Jia Liu, and Runsheng Liu. Fast likelihood computation method using block-diagonal covariance matrices in hidden markov model. In *Proceedings of ICSLP 2002*, Taipei, Taiwan, August 2002.

[92] S.J. Young and P.C. Woodland. State clustering in hidden markov model-based continuous speech recognition. *Computer Speech and Language*, 8(4):369–383, October 1994.

[93] Steve Young. Token passing: a simple conceptual model for connected speech recognition systems. Technical Report tr38, Cambridge University, Engineering Department, july 1989.

Curriculum Vitae

Onderwijs- en onderzoeksloopbaan

- 1998–2002
  Licentiaat Informatica, K.U.Leuven
- 2002–2003
  Master in Artificial Intelligence, K.U.Leuven
- 01/10/2003–15/11/2007
  Doctoraatsbursaal K.U.Leuven
  Afdeling Numerieke Analyse en Toegepaste Wiskunde
- 16/11/2007–04/10/2009
  Wetenschappelijk medewerker K.U.Leuven
  Afdeling Numerieke Analyse en Toegepaste Wiskunde
- Vanaf 05/10/2009
  Onderzoeker bij VRT Medialab

Publicatielijst

Internationale tijdschriften

- Mathias De Wachter, Mike Matton, Kris Demuynck, Patrick Wambacq, Ronald Cools, and Dirk Van Compernolle, Template Based Continuous Speech Recognition, IEEE Transactions on Audio, Speech and Language Processing, volume 15, pages 1377-1390, May 2007
- Mike Matton, Dirk Van Compernolle, and Ronald Cools, Minimum Classification Error Training in Example Based Speech and Pattern Recogni-

tion Using Sparse Weight Matrices, Journal of Computational and Applied Mathematics, submitted

Proceedings van internationale conferenties

- Mike Matton, Mathias De Wachter, Dirk Van Compernolle, and Ronald Cools, A discriminative locally weighted distance measure for speaker independent template based speech recognition, Proceedings of ICSLP-2004, Jeju, Korea, October 2004
- Mike Matton, Mathias De Wachter, Dirk Van Compernolle, and Ronald Cools, Maximum Mutual Information Training of Distance Measures for Template Based Speech Recognition, Proceedings of SPECOM 2005, Patras, Greece, October 2005
- Mike Matton, Dirk Van Compernolle, and Ronald Cools, A Minimum Classification Error Based Distance Measure for Template Based Speech Recognition, Interspeech 2008, Brisbane, Australia, September 2008

Abstracts gepresenteerd op internationale conferenties

- Mike Matton, Dirk Van Compernolle, and Ronald Cools, Minimum Classification Error Training in Example Based Speech and Pattern Recognition Using Sparse Weight Matrices, abstract, ICCAM 2008, Gent, Belgium, July 2008

Abstracts gepresenteerd op andere conferenties

- Mike Matton, and Ronald Cools, Template Based Speech Recognition, Internal PhD Symposium, Leuven, Belgium, April 2005

Technische rapporten

- Mike Matton, and Ronald Cools, A comparison of k-nearest neighbour algorithms with performance results on speech data, Technical Report TW 381, Leuven, Belgium, 2004
- Mike Matton, Dirk Van Compernolle, and Ronald Cools, Minimum Classification Error Training in Example Based Speech and Pattern Recognition Using Sparse Weight Matrices, Technical Report TW 548, Leuven, Belgium, 2009

Nederlandse samenvatting

Afstandsmaten voor voorbeeldgebaseerde spraak- en patroonherkenning

1 Inleiding

1.1 Automatische spraakverwerking

Spraakherkenning is alomtegenwoordig in onze maatschappij. Spraak wordt gebruikt door mensen om efficiënt met elkaar te kunnen communiceren en mensen zijn dan ook heel goed hierin.

Het communiceren met automatische computersystemen verloopt echter op een heel andere manier. Het traditionele toetsenbord en muis maken maar moeizaam plaats voor andere communicatiemogelijkheden. Automatische spraakverwerking laat gebruikers echter toe om via natuurlijke spraak met een computer te kunnen communiceren om bijvoorbeeld een commando te geven, een brief te dicteren, in te loggen via stemherkenning en zo meer...

In deze thesis zijn we geïnteresseerd in het probleem van automatische spraakherkenning, waarbij de computer automatisch de woorden en zinnen die een gebruiker uitspreekt, herkent.

1.2 Een basismodel voor automatische spraakherkenning

In een eerste stap wordt een continu spraaksignaal omgezet in een serie kenmerkenvectoren. Dit heet de voorverwerkingsstap. Er wordt ongeveer om de 10 milliseconden een kenmerkenvector gegenereerd uit het signaal. We zullen deze rij kenmerkenvectoren aanduiden met $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_N$. Bij automatische spraakherkenning probeert men uit deze rij vectoren de meest waarschijnlijke woordenrij te halen. Een woordenrij duiden we aan met $\mathbf{w}$. De kans op een bepaalde woordenrij duiden we aan met $P(\mathbf{w}|\mathbf{x})$ Deze kans kunnen we met de regel van Bayes uitsplitsen in de akoestische waarschijnlijkheid $p(\mathbf{x}|\mathbf{w})$ en een a priori kans $P(\mathbf{w})$. Dit gaat als volgt:

$$\hat{\mathbf{w}} = \arg\max_{\mathbf{w}} P(\mathbf{w}|\mathbf{x}) = \arg\max_{\mathbf{w}} \frac{p(\mathbf{x}|\mathbf{w})P(\mathbf{w})}{p(\mathbf{x})} = \arg\max_{\mathbf{w}} p(\mathbf{x}|\mathbf{w})P(\mathbf{w}) \tag{1}$$

De beide delen van het rechterlid van deze vergelijking (1) vormen de twee grote bouwstenen van traditionele spraakherkenningssystemen. De waarschijnlijkheid $p(\mathbf{x}|\mathbf{w})$ wordt berekend met een akoestisch model en de waarschijnlijkheid $P(\mathbf{w})$ met een taalmodel. De gangbare methode voor akoestische modellering is het gebruik van verborgen Markov modellen (HMM).

Bij dit onderzoek wordt een alternatief onderzocht voor het gebruik van HMM's voor akoestische modellering: voorbeeldgebaseerde spraakherkenning. We zullen eerst een korte schets geven van HMM's en hun tekortkomingen.

1.3 HMM-gebaseerde spraakherkenning

Verborgen Markov modellen zijn statistische modellen die heel geschikt zijn voor het modelleren van rijen van kenmerkenvectoren. De modellen bestaan uit een aantal toestanden $1, 2, \ldots, N_S$ met een statistische observatieverdeling voor elke toestand en een aantal verbindingen tussen deze toestanden met bijhorende waarschijnlijkheden. Concreet bestaat een HMM Φ uit volgende componenten:

- een verzameling toestanden $\mathbb{S} = \{1, 2, \ldots, N_S\}$,

- een observatieruimte Ω, bepaald door de kenmerkenvectoren van de observaties,
- een verzameling transitiekansen $\mathbf{A} = \{a_{ij}\}$ die de kans op het nemen van de transitie van toestand i naar toestand j bepalen; $i, j \in \mathbb{S}$,
- een verzameling uitvoerdichtheidsfuncties $\mathbf{B} = \{b_i(\mathbf{x})\}$, waarbij $b_i(\mathbf{x})$ de waarschijnlijkheid is van het voorkomen van observatie $\mathbf{x}$ bij het binnenkomen van toestand i.

Bij spraakherkenning worden verborgen Markov modellen gebruikt om de waarschijnlijkheid van een sequentie kenmerkenvectoren $\mathbf{x} = \mathbf{x}_1, \ldots, \mathbf{x}_T$ voor een gegeven HMM te berekenen. Daarvoor wordt een som genomen van de waarschijnlijkheid van elke mogelijke rij toestanden $\mathbf{s} = s_1, \ldots, s_T$:

$$p(\mathbf{x}|\Phi) = \sum_{\mathbf{s}} P(\mathbf{s}|\Phi)p(\mathbf{x}|\mathbf{s}, \Phi). \tag{2}$$

Een alternatieve manier voor het berekenen van de waarschijnlijkheid van een akoestische observatie houdt enkel rekening met de waarschijnlijkheid van de best mogelijke toestandenrij. Dit noemt men de Viterbi-benadering. Formeel wordt deze als volgt neergeschreven:

$$p(\mathbf{x}|\Phi) = \max_{\mathbf{s}} P(\mathbf{s}|\Phi)p(\mathbf{x}|\mathbf{s}, \Phi). \tag{3}$$

1.4 Tekortkomingen van verborgen Markov modellen

Hoewel verborgen Markov modellen reeds meer dan een decennium de feitelijke standaard zijn voor het creëren van akoestische modellen hebben ze ook een aantal tekortkomingen. Deze tekortkomingen hebben te maken met veronderstellingen die deze modellen maken over de akoestische gegevens. Hieronder identificeren we een drietal tekortkomingen van HMM's.

Een eerste tekortkoming heeft te maken met het modellering van de lengte van een rij akoestische kenmerkenvectoren. De kans dat een sequentie een aantal keer in dezelfde toestand blijft neemt exponentieel af bij toenemende lengte, m.a.w. langere sequenties zijn minder waarschijnlijk. Dit is niet noodzakelijk zo in werkelijkheid, waar de lengte van een

uitgesproken woord of zin per spreker (en zelfs voor elke spreker individueel) kan verschillen.

Een tweede tekortkoming is de eerste orde Markov veronderstelling, die stelt dat de kans om naar een bepaalde toestand te gaan enkel afhankelijk is van de vorige toestand. Dit maakt de modellen eenvoudiger, maar stemt niet noodzakelijk overeen met de werkelijkheid.

De derde tekortkoming heeft te maken met de voorwaardelijke onafhankelijkheidsveronderstelling: de veronderstelling dat een kenmerkenvector niet afhankelijk is van zijn buren ervan, maar enkel van de toestand in het HMM. Voor spraak is dit uiteraard onjuist.

2 Voorbeeldgebaseerde spraakherkenning

2.1 Inleiding

Niettegenstaande de tekortkomingen van verborgen Markov modellen is het gebruik van deze modellen erg populair. De tekortkomingen hebben ons echter geïnspireerd om op zoek te gaan naar een andere aanpak.

Onze aanpak bestaat erin geen gebruik te maken van akoestische modellen. Daarentegen vergelijken we nieuwe invoerspraak met beschikbare voorbeeldspraak. Om aan herkenning te kunnen doen, wordt de gegevensbank met voorbeeldspraak gesegmenteerd in fonemen. Elk foneem vormt een voorbeeld. De herkenning gebeurt door een rij fonemen te zoeken uit deze voorbeelddatabank die zo goed mogelijk overeenkomt met de invoerspraak. Dit is naar analogie met verborgen Markov modellen, waar de meest waarschijnlijke rij modellen gezocht wordt.

De softwarearchitectuur voor het voorbeeldgebaseerde spraakherkenningssysteem dat voor dit eindwerk gebruikt werd, werd ontworpen door De Wachter [20]. De architectuur bestaat uit twee grote componenten: een bottom-up voorbeeldselectie en een token passing zoekalgoritme.

In de volgende secties wordt de architectuur van dit programma besproken. Eerst wordt echter een belangrijke bouwsteen van voorbeeldgebaseerde spraakherkenning geïntroduceerd: het dynamic time warping algoritme, dat toelaat om twee spraaksegmenten met elkaar te vergelijken. Daarna worden de twee componenten van de voorbeeldgebaseerde spraak-

herkenner kort uitgelegd: de bottom-up voorbeeldselectie wordt gepresenteerd in § 2.3 en het zoekalgoritme wordt beschreven in § 2.4.

2.2 Dynamic time warping

Net zoals het evalueren van verborgen Markov modellen gebeurt met Viterbi-decodering hebben we een algoritme nodig om twee spraaksegmenten, elk bestaande uit een rij kenmerkenvectoren, met elkaar te vergelijken. Het dynamic time warping (DTW) algoritme is hiervoor uitermate geschikt.

Een mogelijkheid om een afstand te definiëren tussen twee niet noodzakelijk even lange spraaksegmenten is om beide segmenten met elkaar te aligneren en de afstanden tussen de overeenkomende kenmerkenvectoren allemaal op te tellen. Enkele vanzelfsprekende voorwaarden worden aan deze alignering opgelegd. De alignering kan enkel vooruit in de tijd gebeuren en het mag niet mogelijk zijn om stukken van de invoer of het referentiesegment over te slaan.

In de praktijk wordt deze afstand berekend door een matrix op te stellen. In deze matrix stellen de rijen de kenmerkenvectoren van het referentiesegment voor. De kolommen stellen de kenmerkenvectoren van het invoersegment voor. Elk element van de matrix is de afstand tussen de respectievelijke kenmerkenvectoren van referentie en invoer. Een afstand tussen beide segmenten wordt berekend door een pad doorheen deze matrix te creëren en alle afstanden op dat aligneringspad op te tellen. Dit wordt geïllustreerd in figuur 2.1 op p. 28. Verder worden ook nog enkele lokale padbeperkingen ingevoerd. De bekendste zijn deze van Sakoe en Chiba, en deze van Itakura. Ze worden geïllustreerd in figuur 2.2 op p. 29.

Gewoonlijk wordt de afstand gedefinieerd als de minimum cumulatieve afstand over alle mogelijke aligneringspaden in de lokale afstandsmatrix. Dit wordt uitgedrukt in vergelijking (2.5). Het DTW algoritme is een efficiënt algoritme om deze afstand uit te rekenen gebaseerd op de techniek van dynamisch programmeren. Het is gebaseerd op de vaststelling dat DTW heel gemakkelijk op een recursieve manier uit te schrijven valt. Wanneer de Itakura lokale padbeperkingen worden gebruikt, kan de DTW

afstand ook als volgt worden neergeschreven:

$$D(\mathbf{x}_1^{\phi_x(t)};\mathbf{y}_1^{\phi_y(t)}) = \min \left\{ \begin{array}{l} D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_0 \\ D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)-1}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_1 \\ D(\mathbf{x}_1^{\phi_x(t)-1};\mathbf{y}_1^{\phi_y(t)-2}) + d^2(\mathbf{x}_{\phi_x(t)};\mathbf{y}_{\phi_y(t)})\gamma_2 \end{array} \right\}., \tag{4}$$

waarbij $\phi_x(t)$ en $\phi_x(t)$ transformatiefuncties zijn die aangeven welke positie de x, respectievelijk de y component heeft in de matrix bij tijdsstap t. De paremeters γ_0, γ_1 en γ_2 zijn gewichten. Het voornaamste om op te merken uit vergelijking (4) is dat deze eigenlijk bestaat uit een (gewogen) som van afstanden tussen kenmerkenvectoren.

Standaard gebruikt het DTW-algoritme een (gekwadrateerde) Euclidische afstandsmaat. Voor dit onderzoek zijn we voornamelijk op zoek gegaan naar alternatieven voor deze Euclidische afstandsmaat. De ontwikkelde afstandsmaten worden beschreven in secties 3, 4 en 5.

2.3 Bottom-up voorbeeldselectie

De overstap van HMM-gebaseerde naar voorbeeldgebaseerde spraakherkenning heeft een enorme invloed op de zoekruimte. Terwijl HMM-gebaseerde spraakherkenning typisch één model gebruikt per (contextafhankelijk) foneem, wordt dit model bij voorbeeldgebaseerde spraakherkenning vervangen door duizenden voorbeelden per foneem. Dit maakt de zoekruimte uiteraard problematisch veel groter.

Om de grootte van de zoekruimte enigszins onder controle te houden wordt een bottom-up voorbeeldselectie ingebouwd in de software architectuur van het voorbeeldgebaseerde spraakherkenningssysteem. Bij deze bottom-up selectie wordt een initiële selectie gemaakt bestaande uit voorbeelden die enigszins met een deel van de invoer overeenkomen. Dit wordt verwezenlijkt door het tijdsfilteralgoritme.

Het tijdsfilteralgoritme verwacht voor elke kenmerkenvector van de invoerspraak een lijst met de dichtste buren kenmerkenvectoren in de databank. Aan de hand van deze lijsten van dichtste buren gaat het algoritme op zoek naar diagonale patronen van overeenkomsten tussen invoer en alle voorbeelden in de databank. Dit wordt geïllustreerd in figuur 2.8 op p. 38.

De geselecteerde voorbeelden worden opgeslagen in een activatiegrafe. Deze grafe bevat een knoop voor elke kenmerkenvector van de invoerspraak en een boog voor elke selectie van een voorbeeld uit de databank. Een eenvoudig voorbeeld van zo'n activatiegrafe wordt gegeven figuur 2.9 op p. 39.

2.4 Token passing zoekalgoritme

De tweede grote component in het voorbeeldgebaseerde spraakherkenningssysteem is het zoekalgoritme. In de gebruikte architectuur wordt een token passing zoekalgoritme gebruikt. Token passing is een zoekstrategie die populair is in HMM-gebaseerde spraakherkenning en ook perfect toepasbaar is op voorbeeldgebaseerde spraakherkenning.

De basis van dit algoritme is het gebruik van tokens. Een token is een (partiële) representatie van een hypothese over (een deel van) de invoer. Een hypothese wordt voorgesteld door een gecombineerde verwijzing naar enkele grafes. De combinatie van grafes bestaat in dit geval uit de activatiegrafe gegenereerd door de voorbeeldselectie en een voorstelling van het woordenboek en en taalmodel in grafevorm. Intuitief zoekt het tokenpassing dus de meest waarschijnlijke sequentie van geselecteerde voorbeelden, rekening houdend met de mogelijke combinatie opgelegd door het woordenboek en het taalmodel.

3 Afstandsmaten

3.1 Inleiding

In dit onderzoek zijn we op zoek naar geschikte afstandsmaten tussen kenmerkenvectoren bij voorbeeldgebaseerde spraakherkenning en patroonclassificatie. Bij voorbeeldgebaseerde spraakherkenning duikt een afstand op twee plaatsen in de architectuur op. In de eerste plaats bij de berekening van de DTW-afstand, die eigenlijk een gewogen som is van afstanden tussen kenmerkenvectoren. Een tweede plaats waar een afstandsmaat nodig is, is bij de berekening van de dichtste buren voor de bottom-up voorbeeldselectie.

3.2 Globaal gewogen afstandsmaten

De Euclidische afstandsmaat, die standaard gebruikt wordt voor deze toepassingen, heeft om diverse redenen een te beperkte expressiekracht. Vooreerst laat deze afstandsmaat niet toe om verschillende kenmerken een verschillend gewicht te geven. Elk kenmerk wordt als evenwaardig beschouwd. In veel van de beschouwde toepassingen klopt dit niet. Een tweede beperking van de Euclidische afstand is dat alle kenmerken als onafhankelijk van elkaar worden beschouwd.

Aan deze tekortkomingen kan tegemoet gekomen worden door de afstand te wegen met een gewichtenmatrix Λ op de volgende manier:

$$d(\mathbf{x}, \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Lambda (\mathbf{x} - \mathbf{y})}. \tag{5}$$

Deze afstandsmaat heeft echter ook nog een belangrijke tekortkoming. Ze behandelt de databank als één grote puntenwolk. Bij de toepassingen die we voor dit onderzoek beschouwen bestaat de ruimte echter uit verschillende klassen van kenmerkenvectoren, waarbij elke klasse typisch andere eigenschappen heeft. Spraak bestaat bijvoorbeeld uit fonemen, en elk fomeen heeft zijn eigen specifieke eigenschappen. Dit wordt geïllustreerd in figuur 3.2 op p. 50, waar de puntenwolk bestaat uit twee duidelijke onderscheiden klassen.

3.3 Klasseafhankelijke afstandsmaten

Zoals hierboven vermeld heeft een globaal gewogen afstandsmaat te weinig uitdrukkingskracht voor het modelleren van afstanden in patroonherkenningsproblemen. Om dit op te lossen maken we gebruik van klasseafhankelijke afstandsmaten. De definitie van de afstandsmaat is analoog aan deze voor een globaal gewogen afstand, enkel wordt nu voor elke klasse een aparte gewichtenmatrix gebruikt. De afstandsmaat wordt op de volgende manier gedefinieerd:

$$d(\mathbf{x}; \mathbf{y}) = \sqrt{(\mathbf{x} - \mathbf{y})^T \Lambda_p (\mathbf{x} - \mathbf{y})}, \quad \Lambda_p \in \mathbb{R}^{D \times D}, \tag{6}$$

waarbij Λ_p de gewichtenmatrix is die bij de klasse van $\mathbf{y}$ hoort.

3.4 Keuze van de gewichtenmatrices

De keuze van de structuur van de gewichtenmatrices is erg belangrijk. Een eenvoudige keuze lijkt het gebruik van een diagonale matrix. Deze matrixvorm heeft echter als nadeel dat er geen correlaties tussen de individuele kenmerken kunnen worden gemodelleerd.

Een oplossing voor dit probleem is het gebruik van volle gewichtenmatrices. Deze keuze heeft dan weer als nadeel dat de berekening van de afstanden een ordegrootte trager wordt, nl. $\mathcal{O}(D^2)$ in plaats van $\mathcal{O}(D)$, met D de dimensie van de kenmerkenvectoren. Bovendien zorgt het gebruik van volle matrices ook voor een ordegrootte meer parameters die moeten getraind worden. Voor kleine KL-Asses kan dit een probleem vormen voor het betrouwbaar trainen van de parameters.

Om aan deze problemen tegemoet te komen hebben we een nieuw concept geïntroduceerd: het gebruik van ijle gewichtenmatrices met een blok-diagonale structuur. Het idee is dat er enkel een parameter op positie (i, j) in de matrix wordt ingevoegd (verschillend van 0) als er effectief ook een correlatie is tussen deze parameters.

4 Partitionering in klassen

In sectie 3 hebben we klasseafhankelijke afstandsmaten gedefinieerd. Er werd echter nog niet vermeld hoe de gegevensbank in klassen kan worden opgedeeld (gepartitioneerd). Het partitioneren van een gegevensbank gebeurt in veel gevallen door middel van een clusteringsalgoritme. Er kan onderscheid gemaakt worden tussen gesuperviseerde clustering, waarbij a priori informatie over de gegevens wordt gebruikt en niet gesuperviseerde clustering, waarbij dit niet gebeurt. Beide vormen van clustering en enkele algoritmes worden hieronder kort besproken.

4.1 Gesuperviseerde clustering

Bij gesuperviseerde clustering wordt gebruik gemaakt van a priori informatie over de gegevens, zoals uitgesproken foneem, geslacht van de spreker, ... Dit is meta-informatie die vooraf beschikbaar is of eruit afgeleid wordt.

Bij spraakherkenning wordt gebruik gemaakt van een gegevensbank met voorbeelddata. Deze voorbeelddata is bovendien geannoteerd met de correcte classificatie, namelijk de uitgesproken zin(nen). Verder is er ook een woordenboek beschikbaar dat aangeeft hoe elk woord uitgesproken wordt in termen van fonemen. Het aantal fonemen varieert per taal, maar ook binnen een taal kunnen afhankelijk van het model een verschillend aantal fonemen worden gebruikt. Voor het Engels worden meestal 40 tot 45 fonemen gedefinieerd.

Deze fonemen zouden kunnen gebruikt worden als klassen voor de afstandsmaat. Een probleem is echter dat de segmentatie van de voorbeelddata in fonemen niet bekend is. Meestal wordt enkel de segmentatie in zinnen voorzien. Om de segmentatie in fonemen hieruit af te leiden bestaan er verschillende technieken. We bespreken twee vaak toegepaste technieken.

Manuele segmentatie

Bij manuele segmentatie wordt de segmentatie van de zinnen in fonemen handmatig uitgevoerd door een specialist. Deze methode is echter niet erg populair omdat het manueel labelen een erg arbeidsintensieve taak is. Daarom wordt meestal een automatische segmentatieprocedure gebruikt.

Geforceerde alignering

Geforceerde alignering is een procedure om automatisch de segmentatie in fonemen te definiëren. Men maakt hierbij gebruik van een bestaand spraakherkenningssysteem en de foneemgrenzen die gevonden worden door dit systeem, worden opgeslagen.

Uiteraard kan deze procedure enkel gebruikt worden indien er al een bestaand spraakherkenningssysteem met akoestische modellen beschikbaar is. De eerste akoestische modellen moeten dan uiteraard op een manueel gesegmenteerde databank getraind worden. Voor het Engels vormt dit echter geen probleem omdat er goede akoestische modellen beschikbaar zijn.

Zowel manuele segmentatie als geforceerde alignering laten toe om voor

elke kenmerkenvector uit de voorbeelddata een foneem-gebaseerde klasse te bepalen. Het aantal klassen is echter beperkt. Fonemen zijn erg afhankelijk van de context waarin ze voorkomen. De uitspraak van een foneem ondervindt veel invloed van het foneem dat ervoor en het foneem dat erna uitgesproken wordt. Dit fenomeen heet coarticulatie. Het lijkt een goed idee om deze coarticulatie mee in rekening te brengen bij het bepalen van de klassen. Dit is analoog aan HMM-gebaseerde spraakherkenning, waar een HMM gedefinieerd wordt voor elk contextafhankelijk foneem. Een foneemklasse, die ook afhankelijk is van het foneem ervoor en het foneem erna, wordt een trifoon genoemd. Indien we echter elk trifoon als een aparte klasse zouden beschouwen, ontstaat voor sommige klassen het probleem van dataschaarste. Dit wil zeggen dat er te weinig gegevens beschikbaar zijn om de modellen van die klasse betrouwbaar te kunnen trainen, of, in ons geval, te weinig gegevens om de gewichten van de afstandsmaat te kunnen trainen. Bij HMM's wordt dit opgelost door sommige toestanden van het model in verschillende klassen te hergebruiken. De technieken die daarvoor gebruikt worden, zijn ook bruikbaar bij voorbeeldgebaseerde spraakherkenning. Hieronder bespreken we de techniek van fonetische beslissingsbomen, een techniek die voor de experimenten voor dit onderzoek werd gebruikt.

Fonetische beslissingsbomen

Fonetische beslissingsbomen werden gecreëerd voor het hergebruiken van toestanden in verschillende contextafhankelijke foneemmodellen om het probleem van dataschaarste bij weinig voorkomende contextafhankelijke trifonen te voorkomen.

Bij een fonetische beslissingsboom wordt de volledige databank, bestaande uit alle kenmerkenvectoren, over de boom verdeeld. Elke interne knoop in de boom bevat een vraag over het (contextonafhankelijke) foneem in kwestie. Enkele mogelijke vragen zijn: "Ligt er een klemtoon op het foneem", "Is het foneem een fricatief", "Is het foneem ervoor een plosief", enz ...

Het bouwen van de boom start met 1 enkele knoop met daarin alle gegevens. Vervolgens wordt een vraag gekozen uit de lijst met alle mogelijke

vragen. De keuze van de vraag gebeurt door middel van een criterium. De meeste van deze criteria hebben te maken met de statistische verdeling van de gegevens voor en na de splitsing.

Omdat de vragenlijst voor deze beslissingsbomen contextafhankelijke vragen bevat, splitst deze methode de gegevens op in contextafhankelijke klassen. Traditioneel worden deze klassen gebruikt voor het creëren van HMM modellen. We kunnen deze onderverdeling in klassen echter ook gebruiken voor het partitioneren van de gegevens voor de klassegebaseerde afstandsmaat.

4.2 Niet gesuperviseerde clustering

De gesuperviseerde clusteringtechnieken die in de vorige sectie besproken werden verdelen de gegevens in klassen gebaseerd op de beschikbare meta-informatie over de gegevens. Het is echter waarschijnlijk dat er in de gegevens ook clusters van punten zitten die niet door meta-informatie beschreven worden.

Niet gesuperviseerde clusteringtechnieken vormen een oplossing voor dit probleem. Deze technieken zijn in staat om clusters in de gegevens te vinden zonder dat er vooraf informatie over beschikbaar is.

In de literatuur worden heel wat clusteringsalgoritmes beschreven. Een erg bekend algoritme is het *k-means* clusteringsalgoritme. Dit algoritme heeft echter als nadeel dat het aantal clusters op voorhand vastgelegd moet worden. Dit kan echter meestal niet. Bovendien heeft het algoritme ook de ongewenste eigenschap dat "lange" clusters gesplitst worden in meerdere clusters.

Eén van de meest bekende varianten die beide problemen oplost, is het ISODATA algoritme.

Een tweede grote groep clusteringsalgoritmes zijn de zogenaamde hiërarchische clusteringsalgoritmes. Deze algoritmes splitsen de gegevens op in clusters op een hiërarchische manier. Voor dit onderzoek wordt enkel gebruik gemaakt van het ISODATA clustering algoritme.

4.3 Een gecombineerde aanpak

Zowel gesuperviseerde als niet gesuperviseerde clusteringsmethoden hebben hun voor- en nadelen. Voor ons onderzoek hebben we beide technieken gecombineerd. In een eerste fase wordt gebruik gemaakt van gesuperviseerde technieken, waardoor een initiële, betekenisvolle opsplitsing van de gegevens gemaakt wordt.

Het is echter waarschijnlijk dat er zich in deze opsplitsing nog clusters bevinden die niet met meta-informatie gevonden kan worden. Daarom hebben we in een tweede fase voor elke klasse nog eens een niet gesuperviseerd clusteringsalgoritme gebruikt, het ISODATA algoritme.

4.4 Experimentele resultaten

Resultaten van de uitgevoerde experimenten geven aan dat het beter is om gebruik te maken van contextafhankelijke informatie over de fonemen voor het definiëren van de klassen. De experimenten waarbij gebruik gemaakt wordt van fonetische beslissingsbomen leveren consistent betere resultaten op dan deze waarbij enkel geforceerde alignering gebruikt wordt.

5 Leren van gewichten van afstandsmaten

In secties 3 en 4 hebben we een klassegebaseerde afstandsmaat gedefinieerd en uitgelegd hoe de beschikbare gegevens in klassen kan worden opgedeeld. Een onbehandeld probleem is hoe de gewichten van deze afstandsmaten getraind kunnen worden.

5.1 Inleiding

De problemen bij spraakherkenning en patroonherkenning kunnen gemakkelijk aanzien worden als classificatieproblemen. Hiervoor zijn de gegevens uit de voorbeelddatabank opgesplitst in klassen, die we $\omega_1, \ldots, \omega_K$ zullen noemen. Merk op dat deze klassen niet overeenkomen met klassen die voorkomen bij de klassegebaseerde afstandsmaten. Bij de afstandsmaten zijn de eenheden kenmerkenvectoren. Bij spraakherkenning bestaan deze klassen typisch uit fonemen, woorden of zinnen.

De kans dat de observatie $\mathbf{x}$ tot de klasse ω_k behoort wordt genoteerd als $P(\omega_k|\mathbf{x})$. Uit deze kans kan heel gemakkelijk een beslissingsregel worden afgeleid als volgt:

$$\hat{\omega} = \arg\max_{\omega_k} P(\omega_k|\mathbf{x}). \tag{7}$$

Bij het trainen van een classificator komt het erop aan om de optimale kansdichtheid te kiezen voor deze verdeling. Hiervoor bestaan verschillende technieken. Een eerste vaak gebruikte verzameling technieken is gebaseerd op het criterium van de maximale waarschijnlijkheid. Een tweede groep technieken is de groep van discriminatieve trainingstechnieken. Deze beide groepen van trainingsmethoden worden hieronder beschreven.

5.2 Het criterium van de maximale waarschijnlijkheid

Bij het criterium van de maximale waarschijnlijkheid worden de parameters van de waarschijnlijkheidsfunctie zo gekozen dat de waarschijnlijkheid van de gegevens uit deze klasse gemaximaliseerd wordt. De waarschijnlijkheid van een verzameling trainingsdata $Y = \mathbf{x}_1, \ldots, \mathbf{x}_N$ wordt als volgt gedefinieerd:

$$P(Y|\Theta) = \prod_{i=1}^{N} p(\mathbf{x}_i|\Theta), \tag{8}$$

waarbij Θ de parameters van de kansverdeling voorstellen.

Indien verondersteld wordt dat de gegevens een Gaussiaanse verdeling volgen, kan eenvoudig worden aangetoond dat de waarschijnlijkheid gemaximaliseerd wordt wanneer het steekproefgemiddelde en de steekproefcovariantiematrix als parameters voor de verdeling worden gebruikt.

Gebaseerd op de veronderstelling dat de gegevens een normale verdeling hebben, heeft De Wachter een afstandsmaat gedefinieerd [20]. Deze afstandsmaat ziet er als volgt uit:

$$d^2(\mathbf{x};\mathbf{y}) = \frac{1}{2}(\mathbf{x}-\mathbf{y})^T \Sigma_k^{-1}(\mathbf{x}-\mathbf{y}) + \log|\Sigma_k|^{1/2}, \tag{9}$$

met Σ_k de covariantiematrix van de klasse waartoe $\mathbf{y}$ behoort.

Er zijn echter ook enkele belangrijke nadelen verbonden aan het gebruik van de methode van de maximale waarschijnlijkheid. Ten eerste werkt dit criterium enkel optimaal als ook de gekozen kansdichtheidsfunctie correct is. In de praktijk is deze functie echter niet gekend. Een gekozen Gaussiaanse verdeling is waarschijnlijk te eenvoudig om de onderliggende verdeling te modelleren.

Een tweede nadeel van dit criterium is dat het enkel optimaal werkt indien een oneindige hoeveelheid trainingsgegevens beschikbaar is. In praktijk moeten "voldoende" gegevens beschikbaar zijn. Bij spraakherkenning bevatten sommige klassen echter onvoldoende gegevens om de parameters van de verdeling betrouwbaar te schatten.

5.3 Discriminatieve trainingsprocedures

Om tegemoet te komen aan de tekortkomingen van het criterium van de maximale waarschijnlijkheid zijn discriminatieve methodes heel geschikt. Deze onderscheiden zich van het criterium van de maximale waarschijnlijkheid doordat ze niet enkel de waarschijnlijkheid van de correcte klasse proberen te maximaliseren, maar tegelijk ook de waarschijnlijk van de incorrecte klassen proberen te minimaliseren. Doordat discriminatieve trainingsprocedures gegevens van zowel de correcte als de incorrecte klassen gebruiken, zijn ook meer gegevens beschikbaar om de parameters betrouwbaar te trainen.

We bespreken hier twee populaire methodes die bij HMM-gebaseerde spraakherkenning gebruikt worden: het criterium van de maximale gemeenschappelijke informatie en het criterium van de minimale classificatiefout.

Het criterium van de maximale gemeenschappelijke informatie

Bij het criterium van de maximale gemeenschappelijke informatie (MMI) wordt geprobeerd om de gemeenschappelijke informatie tussen Ω en X, de stochastische variabelen die respectievelijk de klasse en de observaties voorstellen, te maximaliseren. Deze gemeenschappelijke informatie wordt

als volgt gedefinieerd:

$$\begin{aligned} I(\Omega; X) &= H(\Omega) - H(\Omega|X) \\ &= \sum_{k,i} P(\omega_k, \mathbf{x}_i) \log \frac{P(\omega_k|\mathbf{x}_i)}{P(\omega_k)p(\mathbf{x}_i)}, \end{aligned} \tag{10}$$

waarbij H de (conditionele) entropie voorstelt.

Gebaseerd op deze gemeenschappelijke informatie kan een criterium worden gedefinieerd:

$$J_{\text{MMIE}}(\Lambda) = \sum_{k,i} \left(\log p_\Lambda(\mathbf{x}_i|\omega_k) - \log p_\Lambda(\mathbf{x_i})\right), \tag{11}$$

waarbij Λ de verzameling parameters van de verdeling voorstelt. In ons geval bestaat deze verzamelijk uit de gewichten van de klassegebaseerde afstandsmaat.

Indien we vervolgens een dichtste buren waarschijnlijkheidsfunctie gebruiken, kunnen we dit criterium als volgt opschrijven:

$$J_{\text{MMI}}(\Lambda) = \sum_{i=1}^{N} \left(\log \frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{=})^D} - \log \sum_k \frac{1}{d(\mathbf{y}_i; \mathbf{y}_i^{(k)})^D} \right), \tag{12}$$

waarbij $\mathbf{y}_i^{=}$ staat voor de dichtste buur van $\mathbf{y}$ in dezelfde klasse en $\mathbf{y}_i^{(k)}$ voor de dichtste buur van $\mathbf{y}$ in klasse k. Met de methode van de dalende gradiënten kunnen de gewichten Λ getraind worden.

Bij spraakherkenning worden afstandsmaten echter niet op de kenmerkenvectoren zelf getraind. Bij spraakherkenning moeten fonemen, woorden en zinnen correct geclassificeerd worden. Het is echter eenvoudig om het hierboven gedefinieerde MMI criterium te herschrijven zodat het ook op fonemen kan toegepast worden. Dit doen we door de afstandsmaat te vervangen door de DTW afstand tussen een foneem $\mathbf{y}$ en zijn dichtste buur. Dit criterium ziet er dan als volgt uit:

$$J_{\text{MMI}}(\Lambda) = \sum_{i=1}^{N} \left(\log \frac{1}{D_{\text{DTW}}(\mathbf{y}_i; \mathbf{y}_i^{=})} - \log \sum_k \frac{1}{D_{\text{DTW}}(\mathbf{y}_i; \mathbf{y}_i^{(k)})} \right). \tag{13}$$

Het criterium van de minimale classificatiefout

Het criterium van de minimale classificatiefout probeert rechtstreeks het aantal classificatiefouten te minimaliseren. Om de classificatiefout te bepalen wordt gebruik gemaakt van discriminanten $g_k(\mathbf{x}, \Lambda)$. De classificator kiest de klasse ω_k waarvoor deze discriminant het hoogste is. Met deze discriminanten wordt een classificatiemaat $d_k(\mathbf{x}, \Lambda)$ gecreëerd die de correctheid van de classificatie aangeeft. De voor dit onderzoek gebruikte classificatiemaat ziet er als volgt uit:

$$d_k(\mathbf{x}, \Lambda) = -g_k(\mathbf{x}, \Lambda) + g_j(\mathbf{x}, \Lambda), \tag{14}$$

met ω_j de klasse met de hoogste discriminant is als de correcte klasse ω_k buiten beschouwing gelaten wordt.

Vervolgens wordt met deze classificatiemaat de classificatiefout berekend. Als deze kleiner is dan 0 is de classificatie correct. Als $d_k(\mathbf{x}, \Lambda)$ groter is dan 0, is de classificatie incorrect. Omdat een stapfunctie niet afleidbaar is, wordt een continue 0–1 verliesfunctie $\ell_k(\mathbf{x}, \Lambda)$ gebruikt. Een populaire keuze is de sigmoid verliesfunctie. Het totale verlies over alle voorbeelden in de databank is dan meteen ook het criterium dat moet geoptimaliseerd worden:

$$J_{\text{MCE}}(\Lambda) = L(\Lambda) = \sum_{k=1}^{K} \sum_{i=1}^{N_k} \ell_k(\mathbf{x}_{k,i}, \Lambda). \tag{15}$$

Wanneer de a posteriori verdeling gebruikt wordt en we de waarschijnlijkheid bepalen met een dichtste buren waarschijnlijkheidsfunctie kan de classificatiemaat als volgt worden neergeschreven:

$$d_k(\mathbf{x}, \Lambda) = \log \frac{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D}{d(\mathbf{x}; \mathbf{x}_{\text{nn}}^{\neq})^D}, \tag{16}$$

met D de dimensie van de kenmerkenvectoren.

Analoog kunnen we bij spraakherkenning ook een foneemgebaseerd criterium neerschrijven, waarbij $\mathbf{x}$ de akoestische realisatie van een foneem voorstelt en dus bestaat uit een rij kenmerkenvectoren:

$$d_k(\mathbf{x}, \Lambda) = \log \frac{D_{DTW}(\mathbf{x}; \mathbf{x}_{\text{nn}}^{=})^D}{D_{DTW}(\mathbf{x}; \mathbf{x}_{\text{nn}}^{\neq})^D}. \tag{17}$$

	D	N	k
Kd-tree	exponentieel	logaritmisch	kwadratisch
Roadmap	kubisch [23]	logaritmisch	lineair
Brute Force	lineair	lineair	lineair

Tabel 2: Vergelijking van verschillende k-dichtste burenalgoritmes

Indien het verlies van deze classificatiemaat gesommeerd wordt over alle voorbeelden krijgen we een discriminatief trainingscriterium dat kan geoptimaliseerd worden met de methode van de dalende gradiënten.

6 Het k-dichtste burenprobleem

Het k-dichtste burenprobleem is een erg lastig probleem dat de invloed ondervindt van de "vloek van de dimensionaliteit". Omdat het k-dichtste burenprobleem een essentieel onderdeel vormt van de gepresenteerde architectuur voor voorbeeldgebaseerde spraakherkenning hebben we een vergelijkende studie gemaakt van een aantal algoritmes voor het k-dichtste burenprobleem. Het resultaat van deze studie is samengevat in tabel 2 op p. 196, die de tijdscomplexiteit van de kd-tree, Roadmap en brute kracht algoritmes weergeeft in functie van de dimensie D, het aantal punten N en het aantal gevraagde dichtste buren k. Merk op dat zowel het kd-tree als het Roadmap algoritme een gunstige logaritmische tijdscomplexiteit hebben in het aantal punten N, maar dit steeds ten koste gaat van een behoorlijk slechte tijdscomplexiteit in het aantal dimensies D. Daardoor blijft in veel gevallen het brute force (brute kracht) algoritme het snelst beschikbare algoritme. De vloek van de dimensionaliteit slaat ook hier toe.

7 Experimentele resultaten en besluit

7.1 Experimentele resultaten

Om de voorgestelde afstandsmaten te testen hebben we experimenten uitgevoerd op twee types classificatieproblemen. De eerste groep zijn patroonclassificatieproblemen waarbij ongeziene patronen moeten gekoppeld worden aan een klasse uit een vooraf gedefiniëerde groep klassen. Om de werking van de afstandsmaten uit te testen hebben we hier een dichtste buren classificatie-algoritme gebruikt. De belangrijkste resultaten van deze groep experimenten worden beschreven in tabel 7.3 op p. 142, waarin de classificatiefout wordt weergegeven met respectievelijk de Euclidische en twee discriminatief getrainde afstandsmaten met diagonale en blokdiagonale gewichtenmatrices wordt weergegeven. We kunnen uit de tabel besluiten dat er in bijna alle gevallen een significante relatieve verbetering optreedt.

Vervolgens werden deze afstandsmaten ook uitgetest op voorbeeldgebaseerde spraakherkenning. We hebben twee referentiedatabanken gebruikt: Resource Management en Wall Street Journal. De resultaten worden weergegeven in tabellen 7.6 op p. 146 en 7.7 op p. 146. We hebben experimenteel vastgesteld dat een discriminatief getrainde afstandsmaat significant beter werkt dan een Euclidische afstandsmaat.

7.2 Besluit

We hebben onderzoek gedaan naar afstandsmaten voor voorbeeldgebaseerde spraakherkenning. We hebben daarbij gebruik gemaakt van een klassegebaseerde afstandsmaat die op verschillende manieren verbeterd werd. Vooreerst hebben we het concept van ijle gewichtenmatrices geïntroduceerd, dat toelaat om correlaties in de gegevens te modelleren zonder een computationeel dure afstandsmaat te creëren zoals met het gebruik van volle gewichtenmatrices het geval zou zijn.

Vervolgens hebben we verschillende strategieën onderzocht om de gegevens te partitioneren in klassen. De beste strategie bleek het gebruik van fonetische beslissingsbomen om contextafhankelijke klassen te maken per foneem. Met behulp van een geforceerde alignering werden de gegevens in

de klassen ingedeeld.

Ten derde hebben we onderzoek verricht naar verschillende trainingsalgoritmes om de gewichten van deze afstandsmaten te leren, waarbij de klemtoon lag op discriminatieve procedures. We hebben het schatten van de waarschijnlijkheden aangepast door middel van een dichtste buren waarschijnlijkheidsfunctie. Deze aangepaste procedures laten het trainen van de gewichten van de afstandsmaat toe.

Experimentele resultaten met behulp van deze afstandsmaten tonen een relatieve verbetering van 10% wanneer vergeleken wordt met de (gekwadrateerde) Euclidische afstand (de de facto standaard is voor DTW).

In de marge van dit onderzoek hebben we in een studie ook de tijdscomplexiteit van verschillende dichtste buren zoekalgoritmes onderzocht. Daaruit bleek dat veel van deze algoritmes last hebben van de vloek van de dimensionaliteit, waardoor de tijdscomplexiteit in functie van het aantal dimensies erg slecht is. Daardoor is het brute kracht algoritme voor een hoog aantal dimensies meestal efficiënter dan de andere algoritmes.