NTNU
Norwegian University of
Science and Technology

Detection of phonetic features for automatic classification of Norwegian dialects

Øystein Staven

Master of Science in Electronics
Submission date: June 2016
Supervisor: Torbjørn Svendsen, IET

Norwegian University of Science and Technology
Department of Electronics and Telecommunications

Contents

Problem description

All people, independent of their language, are equally physically equipped to produce speech. This means that it is possible to find universal features of speech which are acoustically defined, and based on speech production. One example of these features are so-called articulatory features, primarily defined by manner and place of articulation.

At NTNU, the research projects Spoken Information Retrieval by Knowledge Utilization in Statistical Speech Processing (SIRKUS) and Atomic Units for Language Universal Speech Processing (AULUS), and several PhD and master theses have studied detection of articulatory and phonetic features, and how these can be used for speech recognition and language identification. So far, none of these studies have included the Norwegian language.

Differences between Norwegian dialects are large, mainly lexically (dialectal words), but also with regard to phoneme realization and intonation. To obtain satisfactory speech recognition for Norwegian, the speaker must speak normalized, and avoid dialectal words. If it is possible to automatically identify the speaker's dialect, this makes it possible for the speech recognizer to adapt lexicon and pronunciation to the user, and hence give lower error rate. The Norwegian database "NB tale" contains speech from all dialectal regions. The recordings are also phonetically labeled, such that it is possible to map acoustics to phonemes, and thus to articulatory/phonetic features. Statistical detectors can be trained with this data.

The task is to use "NB tale" for development of a system for automatic classification of Norwegian dialects, based on phonetic features. The work will be based on an existing system for automatic language recognition, and an important part of the task is to adapt this system to include Norwegian language, initially for language recognition. It is interesting to evaluate the quality of the detection system for Norwegian, compared to other languages.

Assignment given: 15. January 2016

Supervisor: Professor Torbjørn Svendsen

Sammendrag

Dette prosjektet utforsker en utvidelse av et eksisterende språkidentifiseringssystem, for bruk med den norske databasen NB tale/NAFTA, med det endelige målet å klassifisere norske dialekter. Akustiske hendelser blir mappet til fonetiske trekk som er universelle for alle språk. En gjenkjenner blir trent med et dypt nevronettverk (DNN/ANN), som er koblet til en skjult Markov Modell (HMM). Testdata blir dekodet med det nevnte systemet, som kalles "frontend". Disse fonetiske trekkene blir brukt i en høydimensjons dokumentvektor, som kan brukes i språkidentifikasjon. I "backenden" blir en one-versus-all Support Vector Machine trent for hvert språk, for å skille mellom disse språkmerkede dokumentvektorene. En anti-target og en target Gaussian Mixture Model (GMM) blir så trent for å gjøre en endelig språkdesisjon. Frontenden ble trent på seks forskjellige språk fra OGI-databasen, og ble testet med OGIs CV sett, i tillegg til den engelske TIMIT databasen, og del 1 av den norske NB tale/NAFTA. Den beste frontenden viste seg å være et kontekstuavhengig tretilstands system. Backenden ble trent på Callfriend databasen med spontantale, og ble testet på LID 2003 evalueringssett, i tillegg til del 3 av NAFTA. Den beste LID språkgjenkjenningsytelsen viste seg å opptre på norsk og japansk datamateriale.

Abstract

This project explores the expansion of an existing language recognition system for use with the Norwegian data set NB tale / NAFTA, with the ultimate goal being dialect classification. Acoustic events are tokenized into phonetic features which are universal for all languages. A recognizer is trained with a deep/artificial neural network (DNN/ANN), which is connected to a Hidden Markov Model (HMM). Test data is decoded using the aforementioned system, called the frontend. The features are used in a high-dimensional document vector, which can be used for language identification. In the backend, a one-versus-all Support Vector Machine (SVM) is trained for each language, to discriminate between these language-labeled documents. Target and anti-target Gaussian Mixture Models (GMM) are thus trained, which are used for a final language identification (LID) decision. The frontend was trained with six languages from the OGI database, and tested with the OGI CV set, in addition to the English TIMIT database, and part 1 of the NAFTA database. The best frontend system proved to be a context-independent tristate configuration. The backend was trained with the Callfriend database of spontaneous speech, and tested on the LID 2003 evaluation set, in addition to part 3 of NAFTA. The best LID performance was achieved with Norwegian and Japanese data.

Acknowledgements

This report couldn't have been written without the help and support of family, friends and colleagues. A big thank you is in order, to my sisters Ingvild and Vigdis, my father Kjell, and my mother Therese for their support throughout my entire education. You are much appreciated.

I am also grateful for the time spent by my patient supervisor, professor Torbjørn Svendsen, in helping me solve all the different problems that were encountered along the way in this thesis. The author of a large part of the software that my work is based on, professor Marco Siniscalchi, also deserves a mention. Thanks for swift replies to the innumerable e-mails I sent your way. Also, thanks to my colleagues, PhD students Negar Olfati and Abdoreza Sabzi, who gave me an introduction to the software pipeline, and provided assistance along the way. Lastly, thanks to the various authors whose work this thesis is based on. A list of them is found throughout the pages, and at the very end in the bibliography.

List of Figures

List of Tables

Acronyms

ANN	Artificial Neural Network. 1, 23, 82
CI	Context Independent. 2, 74, 81
EM	Expectation-Maximization algorithm. 1, 78
GMM	Gaussian Mixture Model. 1, 2, 28, 78, 79
HMM	Hidden Markov Model. 1, 81
LID	Language Identification. 27, 28, 79, 82
LRE	Language Recognition. 1, 2, 81
LSA	Latent Semantic Analysis. 2, 77
MLP	Multi Layered Perceptron. 9
RBM	Restricted Boltzmann Machine. 82
RC	Right Context. 2, 74, 81
SVD	Singular Value Decomposition. 26, 78
SVM	Support Vector Machine. 2, 81
UAR	Universal Attribute Recogniser. 1–3, 23, 44, 81
VAD	Voice Activity Detection. 75, 77
VSM	Vector Space Modelling. 2, 81

Chapter 1

Introduction

The aim of this report is to explore the application of neural networks to detection of features of speech, called articulatory features. Neural networks are a subcategory of machine learning, which attempts to replace hand crafted functions with iterative algorithms which approximate these functions themselves, with or without human corrective guidance. In speech recognition, Hidden Markov Models (HMMs) with Gaussian Mixture Models (GMMs) are familiar and extensively used techniques to model probability distributions over sequences of observations. The GMM contains probabilistic parameters which are usually found iteratively through the Expectation-Maximation EM algorithm. Given some dataset, the posterior probabilities for which class a speech frame or sequence of frames belongs to can be found. An alternative to this approach is using Artificial Neural Networks (ANNs). While much more processing- and time intensive, research shows that they can produce more accurate posteriors than GMMs [15]. In this report we shall explore this fact and basic theory of ANNs, as well as HMMs and GMMs. Above, in figure 1.1, we see a top-level description of a Language Recognition (LRE) system. Its

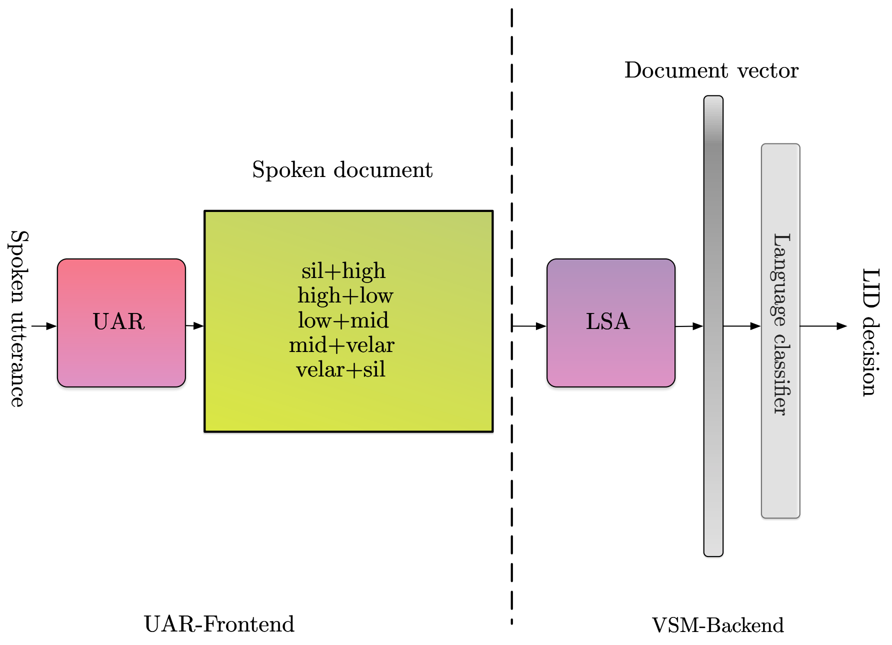


Figure 1.1: LRE system

function is, as the name suggests, to recognize which language it is receiving as input. Each language has its own acoustic signature, which the system tries to learn in a *training* phase. The ANN constitutes the centerpiece of the UAR. It is a powerful discrimination tool, and we use it in this thesis to discriminate between articulatory features. The utility of articulatory features lies in their universality, in that each language shares these features. An acoustic signature can be learned in a Vector Space Modelling (VSM) approach, through finding the frequency of the features, or an ordered, contiguous sequence of such features in a *spoken document* using Latent Semantic Analysis (LSA). This half of the system is referred to as the backend. Now, before this process can be done, we need to determine or estimate the phonetic features. They are found by running data through the trained Universal Attribute Recogniser (UAR) frontend. As we see in figure 1.1, these can include some context from either future or past frames, or both. They can also only consider the current frame, in a Context Independent (CI) method. In this thesis, CI and Right Context (RC) are used.

The crucial, limiting factor here is the frontend. Siniscalchi et al.[24] writes that 100 % LRE can be achieved with a perfect attribute tokenizer. A perfect UAR is, of course, unrealistic. As such, a great deal of work must be done on the frontend to obtain a satisfactory LRE system.

The backend doesn't only comprise the LSA block. Like the frontend, it needs a discriminative block. The job of the VSM block is to project the input features into a high-dimensional vector space, where a Support Vector Machine (SVM) can easily separate the documents with a linear border. This part is called a *language classifier.* Using a one-versus-all method, each language is given a model of its own, through training it in an SVM. The verification task is to determine a GMM for a target class and an anti-target class, using the SVM distances from the positive target training examples, and the negative anti-target examples. The log-likelihood ratio of a test utterance of 30 seconds duration is compared to a threshold for the final decision [24].

With a working LRE system, it could be tested on Norwegian, and the evaluation would consist of comparing the performance on Norwegian with other languages. This is necessary to get a "benchmark" for further use. The aim is to extend this working system to include Norwegian dialects.

Chapter 2

Pattern recognition

To understand the methods used in this thesis, knowing some basic theory behind pattern recognition and machine learning is helpful. This chapter will provide a very brief introduction and overview. The next chapters will go through the core of the UAR in detail.

The fields of machine learning and pattern recognition provide us with methods to classify data into different groups, fit curves or make predictions. We say *learning*, because we don't explicitly define a function $\mathbf{f}$ that gives us a certain output $\mathbf{y}$, given an input vector $\mathbf{x}$. Problems such as speech recognition and computer vision are too complex to solve analytically. Algorithms can help us to instead solve them iteratively. This means that a large part of learning can be done more or less automatically, but not always entirely automatically, as we shall see.

2.1 Types of learning

We can divide learning into three main subcategories: *supervised*, *unsupervised* and *reinforced*. Supervised learning involves learning from labeled example vectors that we give to the system. Digit recognition can be performed in this way. The numbers 0 to 9 would be the range of possible outputs. Example vectors are given as input, and each of them has a single corresponding label, determining which digit it represents. Here, the output can only take discrete values. This is an instance of *classification*. Another task is *regression*, where the output can take continuous values [8]. Removing noise from audio or predicting temperature are examples of such a task.

In *unsupervised* learning, we don't have labeled examples that the system can learn from. It can be viewed as the task of finding patterns and structures, or finding a higher-level representation of the input data [3]. Clustering is such a task, where it must now try to gather the data in regions based on some hypothesized "group affiliation". *Density estimation* is used to determine the distribution of data within the input space, and is also a type of unsupervised learning. *Reinforcement learning* is the last category. Here we have no labeled examples, with the form *(input, correct output)*, but instead *(input, some output, grade for this output)* [3]. This form is relevant for credit assignment, or learning how to play a game. Each action you take in chess has an associated long-term consequence and reward. The goal is to maximize this reward. Likewise, in credit assignment, past economic behaviour, yearly income, outstanding loans and other factors come together to assign a score according to the company or person's credit value. In other learning problems, these factors aren't always easy to ascertain, and even though a computer has learnt them and numbered them, we often can't relate them to anything tangible. If one looks at the weights from a neural network, for example, one will often not be able to interpret them.

2.2 Types of modelling

Depending on our goals and what resources we have available, we will use different types of modelling. Bishop [8] defines three approaches to solving decision problems in two different categories: discriminative and generative modelling.

2.2.1 Discriminative

- First use training data in an *inference* problem to learn a model for the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$ and then use decision theory to assign each new $\mathbf{x}$ to one of the classes.
- Find a *discriminant function* $f(\mathbf{x})$, that maps each input $\mathbf{x}$ directly onto a class label.

2.2.2 Generative

- First solve the *inference* problem of determining class-conditional densities $p(\mathbf{x}|\mathcal{C}_k)$, for each $\mathcal{C}_k$ individually. Also infer the prior class probabilities $p(\mathcal{C})$. Then use Bayes' rule

$$p(\mathcal{C}_k|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\mathbf{x})} \tag{2.1}$$

to find the posterior class probabilities $p(\mathcal{C}_k|\mathbf{x})$. The reason these models are called generative, is because we can generate new data from sampling the distributions.

2.3 Error measures

Figure 2.1 illustrates the learning problem, with noise taken into account. The main point here, is that the ideal hypothesis, or target function, which is the decision surface that separates data, is *unknown*.

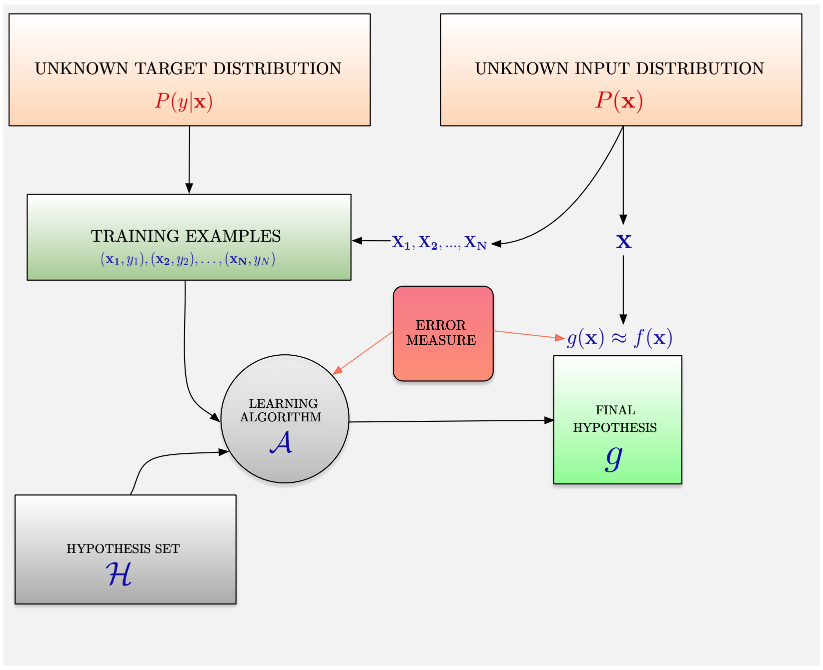

UNKNOWN TARGET DISTRIBUTION
$P(y|\mathbf{x})$
UNKNOWN INPUT DISTRIBUTION
$P(\mathbf{x})$
TRAINING EXAMPLES
$(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_N}, y_N)$
$\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$
$\mathbf{x}$
ERROR MEASURE
$g(\mathbf{x}) \approx f(\mathbf{x})$
LEARNING ALGORITHM
$\mathcal{A}$
FINAL HYPOTHESIS
g
HYPOTHESIS SET
$\mathcal{H}$

Figure 2.1: A general supervised learning problem. [1]

[1]Adopted with permission from Yaser S. Abu-Mostafa [3] at `http://www.amlbook.com`

An error measure quantifies how far our hypothesis is from the target function. We define two different errors. If we have run an algorithm to iteratively produce a viable hypothesis, we can compute the in-sample error E_{in} as how many of our training examples are classified wrongly, using this hypothesis. The out-of-sample error E_{out} tells us how well the hypothesis performs on examples that are outside of our dataset. Since, as mentioned, we very often don't know the entire population of possible examples, we must use probability. If the target is noisy, i.e. the target is affected by, rather than determined by the input data, we can treat y as a random variable. Thus we have a *target distribution* $P(y|\mathbf{x})$ rather than a target function $f(x) = y$, and a data point is generated by the joint distribution $P(x, y) = P(x)P(y|x)$. A general interpretation is that the noisy target is equal to the deterministic target plus noise [3].

Chapter 3

Artificial neural networks

Artificial neural networks is now quite an old subfield of pattern recognition, with its earliest roots lying in McCulloch and Pitts' work on mimicking biological neurons with simple neural networks using electrical circuits. Throughout the 50's and 60's, computers became more powerful, paving the way for simulation of neural networks with software. In 1962, Widrow and Hoff developed a well-known learning rule, later popularized by Geoffrey Hinton et al. as *back-propagation*, for multi-layer perceptrons [23] [7]. Like all new technologies, neural networks had great promise, and was imagined to ultimately be able to model the human brain. A silicon chip can do a single computation faster than the human brain, but the latter has the massively parallel advantage of an estimated 10^{11} neurons [30]. Enthusiasm died down as technical and mathematical obstacles were discovered. One such famous obstacle was the XOR problem, which cannot be solved by using a single linear perceptron. The fact that processors were much slower to compute gradients was also a problem, making it impossible to evaluate solutions effectively.

We shall explore the basic theory of perceptrons and neural networks in the following sections. First though, let's establish the origin of neural networks: the perceptron.

3.1 Perceptron

The perceptron is a single-layer, threshold network. By threshold we mean a binary step function, seen as the activation function in figure 3.1. In its simplest form, where the input data are used directly, it has very limited capabilities, due to not being able to express non-linear decision boundaries. In Bishop's [8] and [7], the perceptron is defined using fixed nonlinear transforms called *basis functions*, rather than using the input data directly. We will use this more general definition for this chapter, as it still holds if we assume $\phi = \mathbf{x}$ for all examples $\mathbf{x}$. The network weighs each input with some value, thus assigning some unique importance to each data point. The output of the perceptron is given by:

$$y(\mathbf{x}) = g\left(\sum_{j=0}^{M} w_j \phi_j(\mathbf{x})\right) = g(\mathbf{w}^T \phi) \tag{3.1}$$

Where ϕ denotes the vector formed from the activations $\phi_0, ..., \phi_M$, and ϕ_0 often is set to 1 as a bias/threshold unit. The nonlinear activation function is an anti-symmetric threshold given by

$$g(a) = \begin{cases} -1, & \text{when} \quad a < 0 \\ 1, & \text{when} \quad a \geq 0 \end{cases} \tag{3.2}$$

To find a well performing hypothesis, we use a simple algorithm called the perceptron learning algorithm. It starts with an arbitrarily initialized weight vector, takes one misclassified example (ϕ_j, y_j) at a time and uses it to update the weight vector w:

$$\mathbf{w}^{\tau+1} = \mathbf{w}^{\tau} + \phi^n t^n \tag{3.3}$$

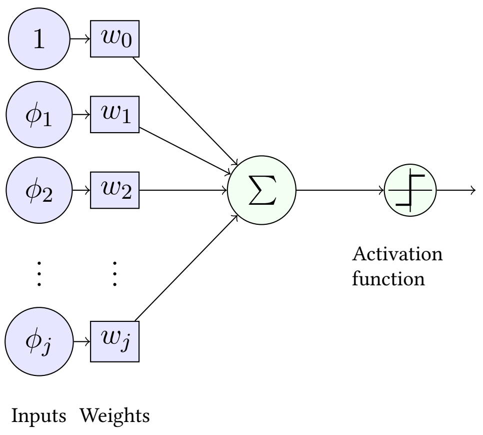


Figure 3.1: Perceptron.

Where ϕ^n is a vector which is misclassified by the perceptrons, τ represents the time step of the algorithm, and t^n are the desired target values. It can be proven ([7] pages 100-101) that the perceptron algorithm will always converge, provided that the data is linearly separable. The *decision boundary* becomes a one-dimensional hyperplane that divides the data in two classes. If the data is not linearly separable, the perceptron learning algorithm will never converge, and we will have to use more advanced methods, covered later. Considering basis functions as non-linear transforms, data that is non-linearly separable in the input space can be separated linearly in the feature space by the perceptron. As Bishop points out, the real problem with the perceptron is that these processing elements are fixed in *advance* and cannot be adapted to a new problem or data set [7].

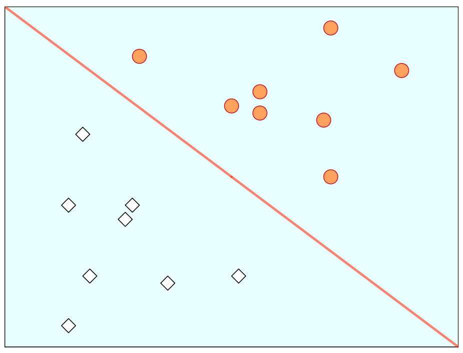


Figure 3.2: Linear decision boundary.

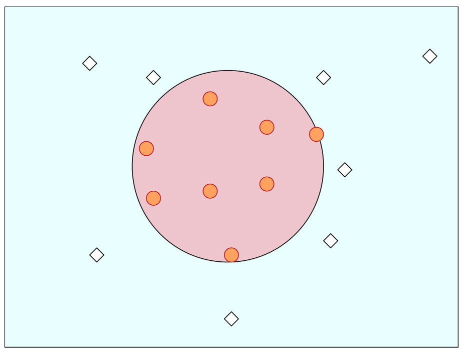


Figure 3.3: Nonlinear decision boundary.

3.2 Logistic regression

Logistic regression is a bit of a misnomer, since it has to do with classification rather than regression. Its name has historical rather than mathematical roots. It falls under the probabilistic discriminative category, where we find a conditional probability directly, without bothering with finding the class-conditional densities and class priors. Before we delve into logistic regression and the cross-entropy error function, we must first define the *sigmoid*, an important *activation function* and building block in neural networks.

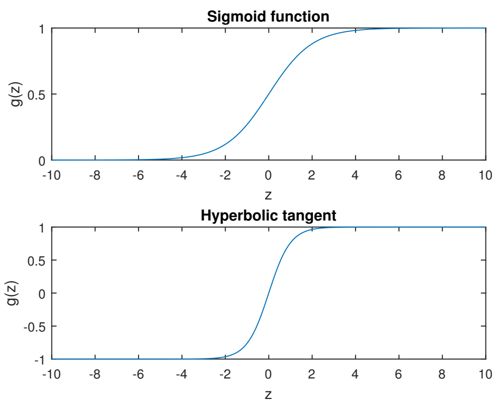


Figure 3.4: Sigmoid and hyperbolic tangent.

3.2.1 Definition of the sigmoid

Figure 3.4 shows the sigmoid function. Its advantage over a linear function is that it is twice differentiable. The aforementioned error gradient benefits from this, as we shall soon see. Instead of a hard threshold in a binary decision, or no threshold at all in linear regression, logistic regression uses a soft threshold with the sigmoid. As seen from the figure, the sigmoid facilitates a probabilistic output. It is defined thusly:

$$\sigma(a) = \frac{e^a}{1+e^a} \tag{3.4}$$

It "squashes" the input and bounds it along its asymptotes. The output result will most often be close to 0 or 1, making probabilities directly readable. Yann LeCun [20] recommends using asymmetric sigmoids like the hyperbolic tangent function, over the standard sigmoid.

$$f(x) = tanh(x) \tag{3.5}$$

For introductory purposes, we shall concentrate on the standard sigmoid.

3.2.2 Binary classification

Let's first elaborate on how the sigmoid function is built. Consider first a binary system of two classes, $\mathcal{C}_1$ and $\mathcal{C}_2$ with corresponding labels $t_n \in 0, 1$. The posterior probability of class $\mathcal{C}_1$ is

$$P(\mathcal{C}_1|\mathbf{x}) = \frac{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}|\mathcal{C}_1)p(\mathcal{C}_1) + p(\mathbf{x}|\mathcal{C}_2)p(\mathcal{C}_2)} = \frac{1}{1+exp(-a)} \tag{3.6}$$

where we define a as [25]

$$a = \ln \frac{p(\mathbf{x}_n|\mathcal{C}_1)p(\mathcal{C}_1)}{p(\mathbf{x}_n|\mathcal{C}_2)p(\mathcal{C}_2)} \tag{3.7}$$

The most used error function for classification is *cross-entropy*. It is built on the maximum likelihood method. We assume a coding scheme of $t = 1$ if the vector belongs to $\mathcal{C}_1$ and $t = 0$ for $\mathcal{C}_2$ Assuming independently generated data points $(\mathbf{x}_n, y_n)$[1] and $y_n = p(\mathcal{C}_1|x_n) = \sigma(\mathbf{w}^T\mathbf{x})$, the log likelihood of the target labels can be expressed as [25][8][3]:

$$\ln p(t_1, ..., t_n|\mathbf{w}) = \sum_n^N (t_n \ln y_n + (1-t_n)\ln(1-y_n)) \tag{3.8}$$

[1]Note that this is not strictly true for speech waveforms. At time $t+1$, the signal is still dependent on the previous signal at t, among other things due to coarticulation.

We maximize this to find the parameters w. As the logarithm of the probability is a monotonic function, maximizing it is the same as maximizing the probability itself. We can express the error, or loss as a function of the parameters as:

$$E(\mathbf{w}) = \frac{1}{N} \sum_{n=1}^{N} \ln \left(1 + e^{-y_n \mathbf{w}^T \mathbf{x}_n}\right) \tag{3.9}$$

Where $\frac{1}{N}$ scales the error function with respect to the number of observations. Note that this is an error function that will lead to severe overfitting if the data are linearly separable. This can be mitigated by including a prior and finding a MAP solution for $\mathbf{w}$, or by adding a regularization term [25][8].

3.2.3 Multiclass classification

Imagine a system where we have K classes, and each data vector can be assigned to one of these classes. The system will have one output y_k for each class, being anywhere between 0 and 1, representing a probability that the k_{th} attribute is present. For classes $K > 2$, we get a more general form called the *normalized exponential*, or *softmax function* [8]:

$$p(\mathcal{C}_i|\mathbf{x}_n) = \frac{p(\mathbf{x}_n|\mathcal{C}_i)p(\mathcal{C}_i)}{\sum_k p(\mathbf{x}_n|\mathcal{C}_k)p(\mathcal{C}_k)} = \frac{\exp(a_i)}{\sum_k \exp(a_k)} \tag{3.10}$$

Where

$$a_i = \mathbf{W}_i^T \mathbf{x}_n \tag{3.11}$$

and

$$a_k = \mathbf{W}_k^T \mathbf{x}_n \tag{3.12}$$

The parameter $\mathbf{W}$ is here a matrix of parameters $\mathbf{W} = [\mathbf{w}_1, ..., \mathbf{w}_K]$ of hyper-planes separating a K number of classes[25]. The multiclass cross-entropy error/loss function is defined as:

$$E(\mathbf{W}, \mathbf{x}_n, y_n) = -\ln \frac{exp(\mathbf{W}_{y_n'}^T \mathbf{x}_n)}{\sum_{y'} exp(\mathbf{W}_{y'}^T \mathbf{x}_n)} \tag{3.13}$$

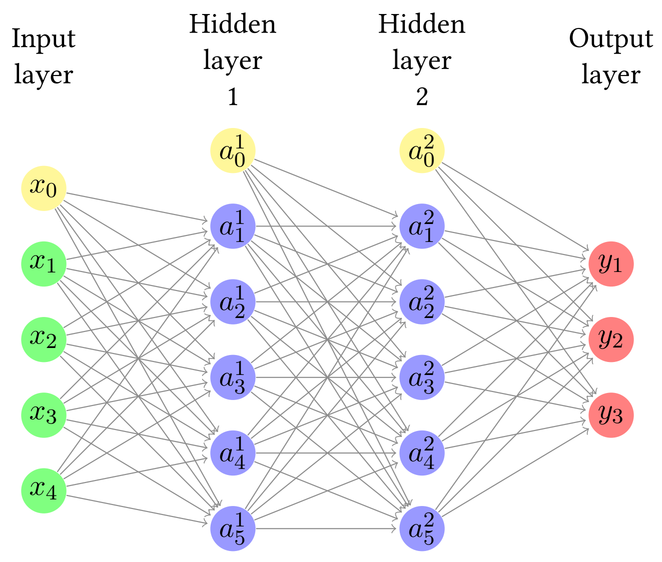
or normalized as

$$\frac{1}{K} \sum_{k=1}^{K} \frac{1}{N_k} \sum_{n \in S_k} E(\mathbf{W}, \mathbf{x}_n, y_n) \tag{3.14}$$

where $N = \sum_k N_k$, N_k is the number of training samples for language k, S_k is a subset of training utterances corresponding to class k and N is the total number of training utterances [25].

3.3 Neural networks

Neural networks are often called *multilayered perceptrons*, which is another misnomer, because they contain multiple layers of logistic regression models with continuous nonlinearities (logistic sigmoids), rather than multiple perceptrons with discontinuous nonlinearities (sign/step function) [8]. We can think of a neural network as a "softened" MLP, due to the twice-differentiable sigmoid. The utility of this fact will become apparent when we now examine the *gradient descent* algorithm together with *backpropagation*. The figure below is an example of a neural network with two hidden layers. The input layer and all the hidden layers have a bias set to 1, denoted as x_0 for the input layer, and a_0 for the hidden layer. This simplifies algorithm implementation. The optimal number of hidden layers and hidden units per layer depends on the data, and the complexity of the data. Deciding on too complex a model can, as always in pattern recognition, lead to serious overfitting. Some testing is often required to make parameter decisions, although some more or less heuristic rules of thumb have been developed by Hinton et al.



3.3.1 Gradient descent

Gradient descent is a general technique for minimizing twice differentiable functions such as the error in logistic regression [3]. The general gradient descent formula is comparable to the perceptron:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\Delta E(\mathbf{w}^{(\tau)}) \tag{3.15}$$

Where η denotes the *learning rate*, which determines the magnitude of change at next iteration of the algorithm. The ultimate goal is to find the global minimum of the cost function. Depending on the initialization of the weights and choice of error function, we may very well end up at a local minimum instead. Using the cross-entropy as error measure produces a convex error surface, which again makes converging to a global minimum a lot easier. The learning rate (or learning coefficient) can either be constant or varying, depending on how far we are from the global minimum. What we have described in formula 3.15 is *batch gradient descent*, which means that we are using the whole data set at once, and subsequently evaluating the error function. An alternative is to use *stochastic gradient descent*, which means that we are evaluating the error function after making a forward pass of one example vector:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\Delta E_n(\mathbf{w}^{(\tau)}) \tag{3.16}$$

This can be done either with data vectors in sequence or uniformly at random chosen from the set of data vectors (which is more in line with the *stochastic* terminology). An intermediate approach is to use mini-batches of data.

3.3.2 Backpropagation

To get the gradient needed in the gradient descent algorithm, the *backpropagation algorithm* is often used. The general idea behind it is that each neuron in the hidden layers is assigned some “responsibility” for the error. Bishop [8] calls this a message passing scheme, where information is alternately sent forwards and backwards through the system. We send the data forward to produce some error measure, and we send it backwards to find out which weights to adjust. Each neuron’s contribution to the error can be found partially differentiating the error with respect to the weights. The following terminology and approach is taken from Bishop [7], and applies to a general feed-forward network. It concludes in a batch error measure. Consider a neuron which computes a weighted sum of its inputs

$$a_j = \sum_i w_{ji} z_i \tag{3.17}$$

where z_i is the activation of a unit, which sends a connection to unit j, and weight w_{ji} is the weight associated with that connection. The sum is transformed by a non-linear activation function $g(\cdot)$ to give the activation function z_j of unit j in the form

$$z_j = g(a_j) \tag{3.18}$$

For batch gradient descent, the error summed over all examples is:

$$E = \sum_n E_n \tag{3.19}$$

with n example vectors, and E_n is a function of the output labels (in the case of classification). We define

$$z_i = \frac{\partial a_j}{\partial w_{ji}} \tag{3.20}$$

$$\delta_j = \frac{\partial E_n}{\partial a_j} \tag{3.21}$$

Then we get the error with respect to the weight as

$$\frac{\partial E_n}{\partial w_{ji}} = \delta_j z_i \tag{3.22}$$

Thus, we need to calculate the value of ∂_j for each hidden unit (neuron) and each output unit and then apply equation 3.22. For the output units we define

$$\delta_k = \frac{\partial E_n}{\partial a_k} = g\prime(a_k)\frac{\partial E_n}{\partial y_k} \tag{3.23}$$

Where $g\prime = g(a)(1 - g(a))$. And for the hidden units, using the chain rule we get

$$\partial_j = \frac{\partial E_n}{\partial a_j} = \sum_k \frac{\partial E_n}{\partial a_k}\frac{\partial a_k}{\partial a_j} \tag{3.24}$$

summed over all units k to which unit j sends connections. Substituting eq. 3.21 into eq. 3.24, we get the *backpropagation formula*:

$$\delta_j = g\prime(a_j)\sum_k w_{kj}\delta_k \tag{3.25}$$

Knowing the errors or $\delta' s$ for the output nodes, we can recursively apply eq. 3.25 to find the $\delta' s$ for each of the hidden units.

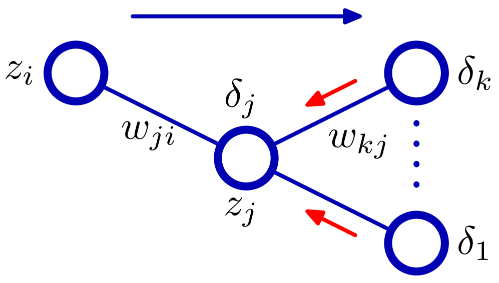


Figure 3.5: Backpropagation[2]

The algorithm is summarized below.

A batch error can then be found as the sum of errors over all examples (provided that all units have the same activation function):

$$\frac{\delta E}{\delta w_{ji}} = \sum_n \frac{\delta E_n}{\delta w_{ji}} \tag{3.26}$$

[2]Reproduced with permission from Christopher M. Bishop [8] at `http://research.microsoft.com/en-us/um/people/cmbishop/prml/webfigs.htm`

Algorithm 3.1: Backpropagation

```
1 Apply input vector $x_n$ and forward propagate to find activations of all hidden and output
  nodes using eq. 3.17 and 3.18
2 Evaluate $\delta_k$ for the output units with eq. 3.23
3 Backpropagate $\delta$'s to obtain $\delta_j$ for each hidden unit using eq. 3.25
4 Evaluate the required derivatives with eq. 3.22
```

3.4 Learning rate and weight initialization

Following a forward pass of training data, and after the cost is determined, we must change the weights to get a different result the next time around. As mentioned, the learning rate or learning coefficient is a constant that determines how drastic the change in weight will be, in the direction of the negative gradient. Choosing a small learning rate will usually ensure convergence to the global minimum, but choosing it too small will take considerably longer time than a more dynamic approach. Using a larger weight to begin with, and scaling it with each epoch (each pass of total data set) proves more efficient. Weight initialization is also an important topic, and forms the basis for much of new research done in this area. The parameters, or weights, should not be initialized to zero; doing this will leave us with zero derivatives and hence no progress. Randomizing them works, but is considered to be too inefficient to use in large-scale networks. In the next chapter we will discuss techniques with which to initialize them with specialized functions , leaving the "grunt work" to the subsequent deep network[26].

3.5 Batch methods

If we update the weights with an average gradient after a full pass of the entire dataset, we have done batch learning. Online (or stochastic) learning means that we randomly choose a data vector and update the weights after each iteration. Papers by both LeCunn [17] and Wilson [29] asserts the superiority of online learning to batch learning, although the former requires a smaller learning rate. As Wilson explains, batch learning can only take one step for each epoch, with each step being a straight line. It can calculate the true gradient, but it does not know how far it can go in that direction before it diverges. The batch method is also much slower (because of redundancy) and less accurate (because of noise). An alternative to both online training and batch training is to divide the available dataset into mini-batches, accumulate the gradient contributions from each data point, and then update the weights. With mini-batches, we update the weights after n data vectors, where $1 < n < data\,set\,length$. Hinton [12] asserts that big mini-batches is nearly always the preferable method due to computational efficiency with tools such as a GPU (graphical processing unit), while Wilson [29] claims that for large data-sets, online learning is always preferable with gradient descent.[3].

3.6 Overfitting

Overfitting occurs when we fit the data more than is warranted. A classical problem is that of regression, where an erratic polynomial curve contaminated with noise hides a signal underneath. Overfitting will produce a poor and non-generalizable model, with too much consideration given to sudden deviations. It will have zero in-sample error but huge out-of-sample error. A complex model can thus use its many degrees of freedom to worsen our learning model, in spite of our large data set. Our model complexity has to match the quantity and quality of our data [3]. As was briefly mentioned in section **??**, regularization is a tool to combat this. Simply guessing the

[3]Keep in mind that this paper was published in 2003, when GPUs were less prevalent

optimal regularization coefficient is a heuristic approach, in which we attempt to penalize what parts of the data is of less importance to our model. There are also more mathematically justified methods, which we will not discuss here. Underfitting happens, conversely, when a model is too simple compared to the data at hand.

3.7 Restricted Boltzmann machines

Restricted Boltzmann machines (RBMs) are a special type of Boltzmann machines, which in turn are energy-based models with hidden variables (or hidden units). They are parameterized models representing probability distributions, and can be used to learn important aspects of an unknown target distribution based on samples from this target distribution. These samples come from our training data. In training the Boltzmann machine, we adjust its parameters until its probability distribution fits the training data [10]. The hidden units models dependencies between the components of observation, and can be viewed as non-linear feature detectors.
Making a prediction or decision consists in setting the value of observed variables (visible units) and finding values of the remaining variables that minimize the energy. In other words, desirable or plausible configurations should have low energy [18] [6] [27]. Energy-based probabilistic models define a probability distribution through an energy function, with hidden and visible units as variables:

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{3.27}$$

Where Z is the partition function, analogous to the normalization factor used in statistical mechanics.

$$Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})} \tag{3.28}$$

RBMs are generative, stochastic neural networks, in that they generate new data, sampled from the learned distribution (joint distribution of inputs and their labels) [10]. The "restricted" in RBMs refers to the fact that no neurons in the same layer are connected to each other. This makes learning less computationally expensive. A neuron of a different layer does, however, have connections to all neurons in the other layer, as we see in figure **??**. As mentioned in the previous chapter, RBMs can be used as pre-training algorithms for regular neural networks, as they find higher-order correlations in the data. The probability distributions learned by the RBM is used as initialization for the deep network parameters (weights), and is then fine-tuned.
The partition function is intractable, but we can use the bipartite graph structure of RBMs to compute and sample from the conditional distributions $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ [5].

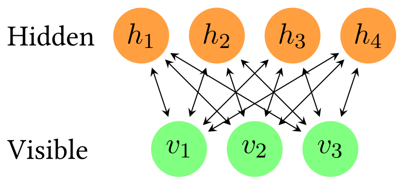


Figure 3.6: Restricted Boltzmann topology with 3 visible units and 4 hidden units.

For binary RBMs the variables $(\mathbf{v},\mathbf{h})$ take values between zero and one, and their energy is given by[13]:

$$E(\mathbf{v},\mathbf{h}) = -\sum_{i \in visible} a_i v_i - \sum_{j \in hidden} b_j h_j - \sum_{i,j} v_i h_j w_{ij} \tag{3.29}$$

The network assigns a probability to every possible pair of hidden and visible vector via the energy function in eq. 3.27 [13]. The derivative of the log probability of the training vector with respect to a weight is as follows[19]:

$$\frac{1}{N}\sum_{n=1}^{n=N}\frac{\partial log\, p(\mathbf{v}^n)}{\partial w_{ij}} = \langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model} \tag{3.30}$$

where the brackets denote expectations under the distribution specified by the subscript. We then get the learning rule for stochastic ascent in the log probability of the data:

$$\Delta w_{ij} = \epsilon\left(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{model}\right) \tag{3.31}$$

Where ϵ is the learning rate.
Hinton et al. describes an approach to train RBMs: To get unbiased samples of $\langle v_i h_j\rangle_{data}$ given a randomly selected training case $\mathbf{v}$, set the binary state h_j of each hidden unit j to one with probability

$$p(h_j = 1|\mathbf{v}) = \sigma(b_j + \sum_i \mathbf{v}_i \mathbf{w}_{ji}) \tag{3.32}$$

where $\mathbf{v}_i\mathbf{h}_j$ is then an unbiased sample. Likewise, for the visibile unit:

$$p(v_i = 1|\mathbf{h}) = \sigma(a_j + \sum_i \mathbf{h}_j \mathbf{w}_{ji}) \tag{3.33}$$

It is difficult to get an unbiased sample from $\langle v_i h_j\rangle_{model}$. An approach called contrastive divergence is used for this. The procedure is described in Hinton et al. [19].

3.7.1 Contrastive divergence

Start by setting the states of the visible units to a training vector. The binary states are computed in parallell using eq. 3.32. When binary states have been chosen for the hidden units, a "reconstruction" is produced by setting each v_i to one with a probability given by eq. 3.33. Then the states of the hidden units are updated. The change in a weight is given by:

$$\Delta w_{ij} = \epsilon(\langle v_i h_j\rangle_{data} - \langle v_i h_j\rangle_{recon}) \tag{3.34}$$

A simplified version is used for the biases. Contrastive divergence is often shortened to CD, with a subscript which indicates how many single full steps of alternate Gibbs sampling are done, after the initial update of the hidden units. In this thesis, CD_1 is used.

3.7.2 Real valued data

In speech processing and recognition, real-valued data is used in the form of MFCC or log-bank features. These are not binary, and thus an intermediate step needs to be done to interface them with an RBM. The energy function can be modified to define a Gaussian-Bernoulli RBM (GRBM)[19]:

$$E(\mathbf{v},\mathbf{h}) = \sum_{i\in visible}\frac{(v_i - a_i)^2}{2\sigma_i^2} - \sum_{j\in hidden} b_j h_j - \sum_{i,j}\frac{v_i}{\sigma_i}h_j w_{ij} \tag{3.35}$$

Our conditional distributions are then

$$p(h_j|\mathbf{v}) = \sigma\left(b_j + \sum_i \frac{v_i}{\sigma_i}w_{ij}\right) \tag{3.36}$$

$$p(v_i|\mathbf{h}) = \mathcal{N}\left(a_i + \sigma_i\sum_j h_j w_{ij}, \sigma_i^2\right) \tag{3.37}$$

where $\mathcal{N}(\mu,\sigma^2)$ is a Gaussian. As Hinton describes, learning the standard deviation of a GRBM is problematic. For pretraining using CD_1, the data are normalized to zero mean and unit variance, the standard deviations are set to one when computing $P(\mathbf{v}|\mathbf{b})$.

3.8 Fitting it together

Now we describe how a deep belief network (DBN) is formed, and how it is used in conjunction with a standard backpropagation layer to form a pretrained deep neural network.
A DBN is trained in this procedure[5]: maximize $\mathbb{E}_{\mathbf{v}\sim p_{data}} log p(\mathbf{v})$ using constrastive divergence. The parameters of the RBM define the parameters of the first layer of the DBN. The next RBM is trained to approximately maximize

$$\mathbb{E}_{\mathbf{v}\sim p_{data}}\mathbb{E}_{\mathbf{h}^{(1)}\sim p^{(1)}(\mathbf{h}^{(1)}|\mathbf{v})} log\, p^{(2)}(\mathbf{h}^{(1)}) \tag{3.38}$$

where $p^{(1)}$ is the probability distribution represented by the first RBM and $p^{(2)}$ is the probability distribution represented by the second RBM. The second RBM is thus trained to model the distribution defined by sampling the hidden units of the first RBM, where the first RBM has the data as visible units. This process can be done for several RBMs. To interface the generative DBN with the backpropagation layer, we use the weights learned from the pretraining process as initial weights that are then "fine-tuned" for discrimination of classes. The initial weights are given by [5]:

$$\mathbf{h}^{(1)} = \sigma\left(b^{(1)} + \mathbf{v}^{\mathsf{T}}\mathbf{W}^{(1)}\right) \tag{3.39}$$

$$\mathbf{h}^{(l)} = \sigma\left(b_i^{(l)} + \mathbf{h}^{(l-1)\mathsf{T}}\mathbf{W}^{(l)}\right) \forall \in 2, ..., m \tag{3.40}$$

A good reference for practical considerations in training RBMs is Hinton's training guide [13].

Chapter 4

Support Vector Machines

Support vector machines (SVM) make decisions directly and do not provide posterior probabilities [8], like a neural network does. A neural network can form an infinite number decision boundaries that successfully separate data, be it linearly or non-linearly separated. You may (and probably will) end up with a different boundary each time you train a neural network, depending on how weights are initialized and what data sequence is applied to the input. So which boundary is optimal with regards to generalization? An SVM solves this problem through maximing the *margin*. The concept is simple: we have marked our training data as either $+1$ or -1, depending on which class it belongs to. Data which belongs to the class in question is denoted by a positive sign, and data which is not in the class is denoted by a negative sign. Consider first a linear decision boundary that separates them. The optimal boundary is the one which is exactly halfway between the nearest points that are associated which each of the two classes. Thus, the boundary will have a maximum margin on both sides, ensuring as little room for misclassification as possible. SVMs are therefore in the family of *maximum margin classifiers*. The margin is the distance from the decision boundary to the nearest data point, i.e. the data points *support* the margin.

4.1 Hard margin case

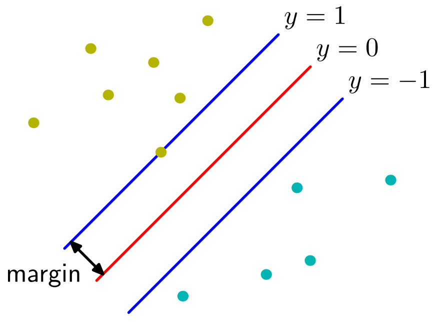


Figure 4.1: SVM boundary with hard margin[1]

If we allow some misclassified data points, or some data points just inside the margin, we have an instance of a *soft* margin classifier. This is the setting described in the last paragraph. First, let's define a hard margin classifier. Consider a linear decision boundary, that discriminates between classes $\mathcal{C}_1$ and $\mathcal{C}_2$. For this chapter we will keep the bias (or offset) separate from the

[1]Reproduced with permission from Christopher M. Bishop [8] at `http://research.microsoft.com/en-us/um/people/cmbishop/prml/webfigs.htm`

weight vector. The following terminology is adopted from [8] and [25]. Consider a training set of N input vectors $\mathbf{x}_1, ..., \mathbf{x}_N$ with corresponding target values $t_1, ..., t_N$ where $t_n \in \{-1, 1\}$, and new data points are classified to the sign of $y(\mathbf{x})$. We get a hyperplane defined as:

$$y(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b = 0 \tag{4.1}$$

which means that

$$y(\mathbf{x}) = \begin{cases} \mathbf{w}_n^T\mathbf{x}_n + b > 0, & \text{for} \quad t_n = +1 \\ \mathbf{w}_n^T\mathbf{x}_n + b < 0, & \text{for} \quad t_n = -1 \end{cases} \tag{4.2}$$

The maximum margin can be found through minimizing an error function:

$$\underset{\mathbf{w},b}{\arg\min} \left\{ \sum_{n=1}^{N} E_\infty(y(\mathbf{x}_n)t_n - 1) + \lambda \|\mathbf{w}\|^2) \right\} \tag{4.3}$$

Where λ is a regularization parameter, and E_∞ is a function that is zero if $z \geq 0$ and ∞ otherwise. This means that all observations must be correctly classified. If they are not, we get an infinite penalty. They are therefore called *support vectors*. If we have outliers, we can use a soft-margin classifier.

4.2 Soft margin case

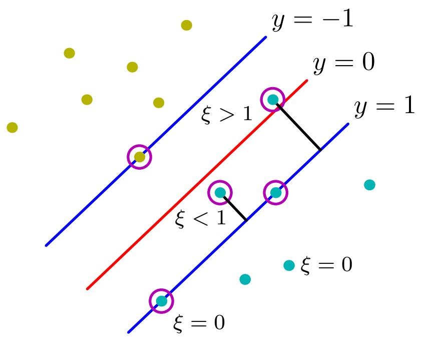


Figure 4.2: SVM boundary with soft margin[2]

As mentioned, a soft margin SVM accepts some outliers or overlap. We describe points that are *inside* the margin with so-called slack-variables $\xi \geq$, with one for each point. $\xi_n = 0$ is assigned to points that are on, or within the correct side of the boundary (inside the margin). $\xi_n = |t_n - y(\mathbf{x}_n)$ for other points. Our constraints are then formulated as:

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \qquad n = 1, ..., N \tag{4.4}$$

Thus penalizing points that lie on the wrong side of the margin boundary, whilst also maximizing the margin. We minimize

$$C \sum_{n=1}^{N} \xi_n + \frac{1}{2}\|\mathbf{w}\|^2 \tag{4.5}$$

where parameter $C > 0$ controls the trade-off between the penalty and the margin, and plays a similar role as (or the inverse of) a regularization coefficient. If $C \to \infty$, our model is a maximum

[2]Reproduced with permission from Christopher M. Bishop [8] at `http://research.microsoft.com/en-us/um/people/cmbishop/prml/webfigs.htm`

margin classifier. The soft margin error function can be written as [3][25]:

$$\operatorname*{arg\,min}_{\mathbf{w},b} \left\{ \frac{1}{2}\|\mathbf{x}\|^2 + C \sum_{n=1}^{N} \max\{1 - y_n(\mathbf{w}^T\mathbf{x}_n + b, 0\} \right\} \tag{4.6}$$

For a full treatise of SVM algorithms, the reader is referred to Bishop [8] or Abu-Mostafa et al. [3], where solutions for non-linear boundaries are also presented.

Chapter 5

Hidden Markov Models

Hidden Markov models (HMM) form the basis of most modern speech recognition systems. The framework has been known quite some time, but modelling techniques have improved over the years. HMMs are a method of characterizing the observed data samples of a discrete-time series, such as speech. The assumption is that the data can be analyzed as a parametric stochastic process. [14] [11] The HMM is as mentioned, a stochastic model. It has transition and emission (observation) probabilities. In the case of speech, the state is the configuration of the larynx, tongue, teeth etc. Each state has a probability distribution, which determines how likely a certain a phoneme is to be "drawn" out from our bag of possible utterances. This means that the emissions or observations are conditioned on the state: as we can imagine, changing the frequency of vocal folds' excitation or positioning of the tongue greatly alters the generated sound. If the HMM is in a given state s_i at time t, it has a probability associated with transitioning to state s_j at $t + 1$, which is only dependent on the *current* state. Take the case of a spoken word, "hello". A *language model* describes the a priori probability $P(W)$ of that word occurring in English vernacular. This word is quite common, and so it has a high prior probability. The spoken word is encoded as a feature vector, and input to our HMM. The likelihood $P(O|\lambda)$ can through Bayes' rule be used to find the posterior probability $P(\lambda|O)$, which describes how probable it is that the model parameters λ generated the observed acoustic vector, and hence, the word in question. It is helpful to lay out the terminology before we go further. The terminology below is adopted from [14] and [16].

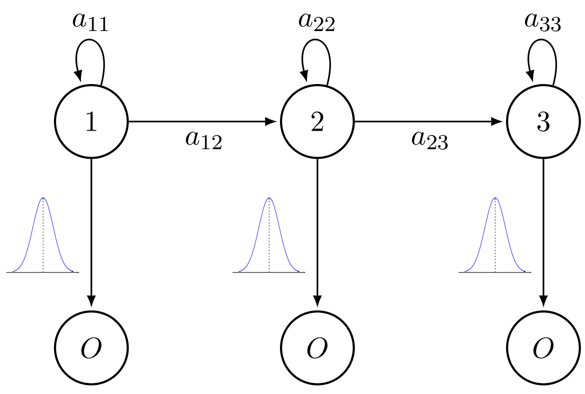


Figure 5.1: Left-to-right HMM with gaussian emission probabilities

- $\mathbf{V} = V_1, V_2, ..., V_v$ is the output observation alphabet, or vocabulary
- $\mathbf{Q} = q_1, q_2, ..., q_N$ are the set of N states
- q_0, q_F are start and end states not associated with observations

- $\mathbf{A} = a_{ij}$ is the transition probability matrix, where $a_{ij} = P(q_t = j | q_{t-1} = i)$
- $\mathbf{B} = b_i(o_t)$ is the output probability matrix, where $b_i(o_t)$ is the probability of emitting the observation o_t at time t
- $\mathbf{O} = o_1, o_2, ..., o_t, ..$ is the observed output of the HMM.
- $\mathbf{S} = s_1, s_2, ..., s_t, ...$ is the state sequence. It is not observed, but *hidden.*
- $\lambda = (\mathbf{A}, \mathbf{B})$ is the parameter set

We will solve two problems with the HMM: *decoding* and *forced realignment.*

- In decoding, we want to discover the best hidden state sequence Q given the observation sequence O and an HMM $\lambda = (A, B)$.
- In forced realignment, we iteratively find better state boundaries, given a posterior vector of $b_i(o_t)$ and a word transcription.

More on these two problems later. For both of these we need the Viterbi algorithm, which will be explained now. The following approach is adopted from [16].

5.1 Viterbi algorithm

The Viterbi algorithm is a dynamic programming algorithm. It processes the observation sequence left to right, and fills out a trellis. Each cell of the trellis $v_t(j)$ represents the probability that the HMM is in state j after seeing the first t observations and passing through the most probable state sequence $q_0, q_1, ..., q_{t-1}$, given the parameters λ [16]. The value of each cell $v_t(j)$ is computed by recursively taking the most probable path that leads us to this cell [16].

$$v_t(j) = \max_{q_0, q_1, \ldots, q_{t-1}} P(q_0, q_1, ..q_{t-1}, o_1, o_2, .., o_t, q_t \;=\; j | \lambda) \tag{5.1}$$

The $\max$ indicates that we find the most probable path over all possible previous state sequences. If we have already computed the probability of being in every state at time $t - 1$, we can now compute the Viterbi probability by taking the most probable path of the extensions of the paths that lead to the current cell. For a given state q_j at time t, the Viterbi probability $v_t(j)$ can be computed as [16]:

$$v_t(j) = \max_{i=1}^{N} v_{t-1}(i) a_{ij} b_j(o_t) \tag{5.2}$$

where $v_{t-1}(i)$ is the previous Viterbi path probability from the previous time step. In summary, the goal of the Viterbi algorithm is to find the best state sequence $q = (q_1, q_2, .., q_T)$ given the set of observations $O = (o_1, o_2, .., o_T)$. It also needs to find the probability of this state sequence, which is the joint probability of the state and observation sequences [16]. Viterbi is identical to the related Forward algorithm, except it takes the *max* over previous path probabilities where the Forward algorithm takes the *sum* [16].

5.2 The hybrid ANN/HMM approach

In figure 5.1 we see that each of the emission probabilities of the observed outputs O has a single Gaussian distribution. This is a simple model. In reality, this is much too simplistic. Few, if any, phonemes or attributes, can be accurately modelled with such a distribution. One approach with which to increase the granularity is to use Gaussian Mixture Models (GMM), where mixtures of different gaussians form an arbitrarily precise representation of the emission probability. In our approach with the HMM though, we will not use this. We will instead use the posterior vector obtained from the neural network as our emission probabilities. Now, this is a hard decision in the sense that the neural network only outputs discrete probabilities for each attribute (or each state of the attributes). In a standard neural network approach, one would simply take the maximum probability target and assign the input data vector to the corresponding class. The ANN can include temporal context by splicing data vectors (this is done in this thesis), but to obtain a document vector for use in the backend, we need to decode a sequence of words (or in this case, attributes). Feed-forward ANNs are good at discriminating between classes, but they cannot decode temporal sequences alone. This is where our HMM comes in: it provides a reasonable structure for representing sequences of words or speech sounds [21]. The ANN computes posterior probabilities of a state j given the observation vectors $P(q_j|o_t)$. What we really need for the HMM though, is the observation likelihood, $b_j(o_t)$, or $P(o_t|q_j)$. But we can use Bayes' rule to compute the latter from the former [16]. The ANN computes

$$p(q_j|o_t) = \frac{p(o_t|q_j)\, p(q_j)}{p(o_t)} \tag{5.3}$$

Rearranging the terms, we get

$$p(o_t|q_j) = \frac{p(q_j|o_t)\, p(o_t)}{p(q_j)} \tag{5.4}$$

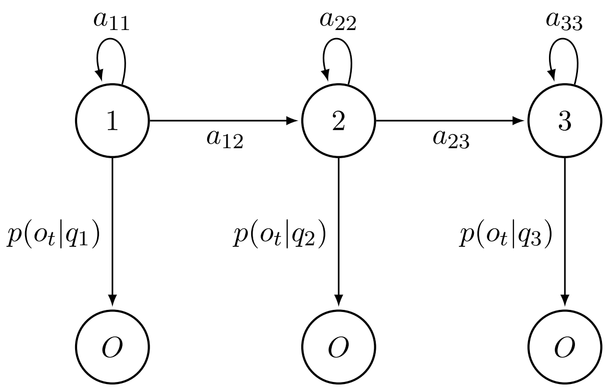


Figure 5.2: Left-to-right HMM with ANN posteriors

The numerator in the right-hand side of eq. 5.4 is the posterior from the ANN, and the denominator on the right-hand side is the prior over the class. We cannot compute $p(o_t|q_j)$, but we can use the ANN posterior along with the ANN prior to form a *scaled likelihood*, seen on the left-hand side of eq. 5.4 . The probability of the observation $p(o_t)$ is constant for all classes during recognition[16], and will not change the classification [21], and so it can be ignored, without causing harm.

5.2.1 Forced realignment

If we use only a single state in our decoding, we are essentially reducing the HMM into a dummy model for Viterbi decoding. It's still useful for decoding a word sequence, but a more powerful method is to use the tristate HMM with forced alignment. Forced Viterbi alignment is a simplification of the Viterbi decoding algorithm, since it only has to figure out the correct attribute state sequence, but it doesn't have to discover the word sequence, which is given. What we get is the best state path corresponding to the training observation sequence [16]. Consider nine frames of an attribute. We start with a uniform segmentation of our training data, meaning that each state has equally many frames, in this case three frames each. We run the data vector through the already trained ANN, producing a posterior probability for each frame. Then, we use the posteriors as emission probabilities in the Viterbi algorithm along with the word or attribute transcription. The borders between the states are now moved, and we use this new prior vector to train the ANN. This can be done iteratively with good results, ultimately producing a better discrimination and decoding system.

5.2.2 Why use this method?

Traditionally, HMMs were used with GMMs. GMMs can model any distribution accurately, provided enough components. So why replace it with an ANN? As Morgan/Bourlard [21] write, there are two main reasons to do this.

- Standard HMM approaches require strong assumptions about the statistical character of the input, in that they assume that successive acoustic vectors are uncorrelated. An ANN can splice together vectors to provide some context, and thus the network learns something about the correlations between them.
- Probabilities will be optimized to maximize discrimination between sound classes, rather than to closely match the distribution within each class. This type of training can be more conservative of parameters than for example the GMM.

Chapter 6

Articulatory features

The UAR is based on finding salient, universal features in speech for use in language recognition. We will now introduce the different attributes that constitute the classes to be discriminated with the ANN. The descriptions are adopted from Jurafsky et al.[16]. They include both *voiced* and *unvoiced* sounds. Voiced sounds are made by vibrating the vocal chords, while unvoiced sounds are not made with this vibration. Manner of articulation refers to how airflow is restricted, while place of articulation refers to the point of maximal restriction of airflow.

6.1 Manner of articulation

- **Affricate:** Stops followed immediately by fricatives
- **Stop:** a consonant where airflow is blocked for a short time
- **Flap:** sounds made by a quick motion of the tongue against the alveolar ridge
- **Fricative:** sounds where airflow is restricted but not blocked
- **Glide:** sounds where the direction of airflow glides over the body of the tongue before exiting the mouth
- **Liquid:** as for glides, except with the tip of the tongue
- **Nasal:** sounds made where air is allowed to pass into the nasal cavity
- **Sibilant:** high pitched fricatives
- **Vowel:** sounds made with the vocal chords vibrating

6.2 Place of articulation

- **Labial:** determined by the positioning of the lips.
- **Dental:** sounds made by pressing the tongue against the teeth
- **Coronal:** sounds made by pressing the tip of the tongue against the roof of the mouth
- **Palatal:** sounds made with the blade of the tongue against the alveolar ridge
- **Velar:** sounds made by pressing the the back of the tongue against the velum, or soft palate
- **High:** voiced sounds where the tongue is positioned at a *high* point in the mouth (compared to other voiced sounds)

- **Mid:** voiced sounds where the tongue is positioned at a *mid* point in the mouth (compared to other voiced sounds)
- **Low:** voiced sounds where the tongue is positioned at a *low* point in the mouth (compared to other voiced sounds)
- **Glottal:** sounds made by closing the glottis, i.e. bringing the vocal folds together

6.3 Silence

Silence is also listed as an attribute in this work. Its property lies in its name, and is simply the absence of audible sound.

6.4 Other

One last attribute that functions more as an anti-attribute, is “other”. Sounds that are unintelligible or that carry no information can be labelled as “other”. Discontinuities in transcriptions with respect to time can also be labeled in this way.

Chapter 7

Latent Semantic Analysis

In latent semantic analysis (LSA), we want to find the significance of certain attributes occurring in a document vector. Rare events have more info embedded in them than do frequent events. We will see how mathematical techniques and terms such as entropy and singular vector decompositions can help us map these events into a high dimensional vector space where they can be easily separated with machine learning techniques. The events in question are speech utterances. The LSA framework was developed for information retrieval, where the aim was to match words in queries with words in documents [4]. The vector space modelling (VSM) approach used in this thesis aims to map speech utterances into a high dimensional document vector. An input spoken utterance is treated as a query, in the terminology of LSA. A feature vector with n-gram elements (a contiguous sequence of n items) carry some statistics about co-occurrences, and can say a lot about a language or dialect. Some of the following sections will borrow from Siniscalchi, Svendsen et al. [24].

7.1 Entropy

It is helpful to know the meaning of the word entropy before going further. Originally a part of the field of thermodynamics, it has been adapted as a term into information theory as well. A measure of information content in x is dependent on the probability distribution $p(x)$. Given a uniform probability distribution, an event in which $p(x = 0) = 1$, x will most certainly be 0. This means there is not much information in this theorized event. We can define a monotonic function $H(p)$ which is dependent on the distribution of x.

$$H(p) = -\sum_{x \in X} p(x_i) log\, p(x_i) \tag{7.1}$$

where we can use any base logarithm, depending on our application. Distributions $p(x_i)$ that are sharply peaked around a few values will have relatively low entropy, while those that are spread out more evenly will have higher entropy [8].

7.2 Vector space modelling

Suppose an utterance X, represented by a sequence of speech feature vectors, O, can be decoded into a spoken document, $d(X)$, consisting of a series of I acoustic units, $d(X) = t_1, .., t_i, .., t_I$. The units are drawn from a universal inventory of J attribute labels, $U = u_1, .., u_j, .., u_J$. The spoken documents can be converted into a vector of length $M = J$, where each entry contains the number of times that the m_{th} label appears in $d(X)$. We can increase indexing power by using context-dependent modelling and counting n-grams. The vector length is dependent on the number of n-grams used. If all unigrams, bigrams and trigrams are used, we have $M =$

$J + J \times J + J \times J \times J$. The LSA function $f(w_m)$ gives us a measure of the significance of having a particular n-gram in a transcribed document, $d(X)$. Each document is then represented by an M-dimensional feature vector $\mathbf{x} = [f(w_1), .., f(w_m), .., f(w_M)]^t$ where $\mathbf{x}^t$ denotes the transpose of $\mathbf{x}$ for each spoken document [24].

7.3 Singular value decomposition

The singular value decomposition (SVD) is closely related to principal component analysis (PCA). The aim of PCA is to find the dimension in which data changes most, or equivalently, where we have biggest variance in that particular dimension's values. PCA is a form of feature selection, where we attempt to get rid of redundancy or less informative dimensions [3]. Removing a non-informative dimension will not hurt our discrimination process (described later). PCA constructs a small number of linear features to summarize the input data, and it rotates the axes through a linear transformation so that the important dimensions in the new coordinate system can be retained while the less important ones get discarded [3]. Singular value decomposition is a general way of changing the basis vectors of our new coordinate system. Formally [3], if we have a data matrix W, we can represent it as a product of three matrices. Assume that $W \in \mathbb{R}^{N \times R}$ with $N \geq R$. Then,

$$W = USV^T \tag{7.2}$$

where $U \in \mathbb{R}^{n \times d}$ has orthonormal columns, $V \in \mathbb{R}^{d \times d}$ is an orthogonal matrix, and S is a non-negative diagonal matrix. The diagonal elements of S are the *singular* values of W, ordered from largest to smallest, and the number of non-zero singular values is the rank of W, which we can call d. U contains the left singular values of W as its columns, and V contains the right singular vectors of W, as its columns.

7.4 Latent semantic analysis

A term-count vector is generated by counting the number of times each term appears in the speech document. A term can be a unigram, bigram or trigram. Term-count vectors of manner and place will be separated in this thesis, but they can be merged to increase discrimination power [24]. As was mentioned in section 7.1, we can use entropy as a measure of the significance of a certain event. Regular words like "the" or "a" are commonly used in the English language, and in Norwegian words like "å" or "du" are not very informative as to which dialect we are classifying. N-grams are therefore weighted by their likely discriminating power. A term-document matrix W, is constructed from all N training utterances. The term-document matrix $W = \{w_{i,j}\}$ consists of weighted count values given by

$$w_{i,j} = \left[1 + \frac{1}{logN} \sum_{j=1} N \frac{n_{ij}}{n_{i.}} log \frac{nij}{n_{i.}}\right] \frac{n_{ij}}{n_{.j}} \tag{7.3}$$

where n_{ij} is the number of n-gram i occurs in document j, and $n_{i.}$ is the number of times that n-gram i appears in the N training documents, and $n_{.j}$ is the number of n-gram in document j. The weight is close to zero if the given term has a uniform distribution through our data, and is close to one if document is very rare. The term-document matrix has a dimension of $M \times N$, where M is equal to the number of n-gram occurences considered (bigram, trigram and so on). In this thesis, we will use up to trigram. If p is the number of attributes, $M = p + p^2 + p^3 + p^4$. Many of the high number of terms never occur in the data in the training documents, leading to both a high-dimensional and sparse term-document matrix. This can be improved with the SVD:

$$W \approx \hat{W} = USV^T \tag{7.4}$$

where U is $M \times Q$, S is $Q \times Q$, V is $N \times Q$ and $Q << M$ is the rank of $\hat{W}$.

7.4.1 Test documents

A test document $\tilde{d}$ can be used to construct a document vector

$$\tilde{v} = \tilde{d}^T U \tag{7.5}$$

which is referred to as a pseudo document vector. This can be used in evaluation of the LID system.

Chapter 8

Gaussian Mixture Models

Gaussian mixture models (GMM) facilitates a richer distribution than what a single Gaussian distribution can. They will constitute the last LID stage in our system. The following approach is adopted from Bishop [8]. They can be written as a linear superposition of Gaussians as:

$$p(\mathbf{x}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k) \tag{8.1}$$

Consider a binary random variable **z** with a 1-of-K representation, where one of z_k is equal to one and the others are zero. The values of z_k then satisfies $z_k \in \{0,1\}$ and $\sum_k z_k = 1$. A marginal distribution over **z** can then be specified in terms of *mixing coefficients* π_k:

$$p(z_k = 1) = \pi_k \tag{8.2}$$

where parameters $\{\pi_k\}$ must satisfy

$$0 \leq \pi_k \leq 1 \tag{8.3}$$

and

$$\sum_{k=1}^{K} \pi_k = 1 \tag{8.4}$$

We can write the distribution of **z** as

$$p(\mathbf{z}) = \prod_{k=1}^{K} \pi_k^{z_k} \tag{8.5}$$

The conditional distribution of **x** for a particular value of **z** is a Gaussian, and can be written as

$$p(\mathbf{x}|z_k = 1) = \mathcal{N}(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k) \tag{8.6}$$

or equivalently

$$p(\mathbf{x}|\mathbf{z}) = \prod_{k=1}^{K} \mathcal{N}(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k)^{z_k} \tag{8.7}$$

The marginal distribution of **x** can be obtained by summing the joint distribution over all possible states of **z** to give

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{z}) p(\mathbf{x}|\mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k) \tag{8.8}$$

Which is a marginal distribution of the form seen in eq. 8.1. For every data point $\mathbf{x}_n$, there is a corresponding latent variable $\mathbf{z}_n$. Let us also define another important quantity:

$$\gamma(z_k) \equiv p(z_k = 1|\mathbf{x}) = \frac{\pi_k \mathcal{N}(\mathbf{x}|\mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}|\mu_j, \mathbf{\Sigma}_j)} \tag{8.9}$$

Where we can view π_k as the prior probability of $z_k = 1$, and $\gamma(z_k)$ as the posterior probability once we have observed $\mathbf{x}$, or the "responsibility" that component k takes for "explaining" the observation $\mathbf{x}$.

If we model a set of observations $\{\mathbf{x}_1, \mathbf{x}_2, .., \mathbf{x}_N\}$ we can now model this data with a mixture of Gaussians. The data set can be represented as an $N \times D$ matrix $\mathbf{X}$ with the n^{th} row given by $\mathbf{x}_n^T$. The latent variables is denoted by an $N \times K$ matrix $\mathbf{Z}$ with rows $\mathbf{z}_n^T$. The log-likelihood function can then be defined as

$$ln\, p(\mathbf{X}|\pi, \mu, \mathbf{\Sigma}) = \sum_{n=1}^{N} ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k) \right\} \tag{8.10}$$

Maximizing this likelihood is then associated with maximizing the likelihood of the data.

8.1 Expectation-Maximization algorithm

The Expectation-Maximization (EM) algorithm is a method to find the maximum likelihood solutions for models with latent variables. We will now see how it can be used for the GMM.

Algorithm 8.1: EM algorithm

1 Initialize the means μ_k, covariances $\mathbf{\Sigma}_k$ and mixing coefficients π_k, and evaluate the initial value of the log likelihood.

2 Expectation step. Evaluate the responsibilities using the current parameters

$$\gamma(z_{nk}) = \frac{\pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k)}{\sum_{j=1}^{K} \pi_j \mathcal{N}(\mathbf{x}_n|\mu_j, \mathbf{\Sigma}_j)} \tag{8.11}$$

3 Maximization step. Re-estimate the parameters using the current responsibilities

$$\mu_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk}) \mathbf{x}_n \tag{8.12}$$

$$\Sigma_k^{new} = \frac{1}{N_k} \sum_{n=1}^{N} \gamma(z_{nk})(\mathbf{x}_n - \mu_k^{new})(\mathbf{x}_n - \mu_k^{new})^T \tag{8.13}$$

$$\pi_k^{new} = \frac{N_k}{N} \tag{8.14}$$

where

$$N_k = \sum_{n=1}^{N} \gamma(z_{nk}) \tag{8.15}$$

4 Evaluate the log-likelihood

$$ln\, p(\mathbf{X}|\pi, \mu, \mathbf{\Sigma}) = \sum_{n=1}^{N} ln \left\{ \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}_n|\mu_k, \mathbf{\Sigma}_k) \right\} \tag{8.16}$$

and check for convergence of either the parameters or the log likelihood. If the convergence criterion is not satisfied, return to step 2.

Chapter 9

Feature extraction

9.1 Mel filterbanks

The mel-scale was developed to account for the logarithmic nature of human hearing. Humans do not interpret pitch linearly outside of the range 0-1000 Hz. An approximate formula was through empirical experiments found to be [14]:

$$B(f) = 1125\, ln(1 + f/700) \tag{9.1}$$

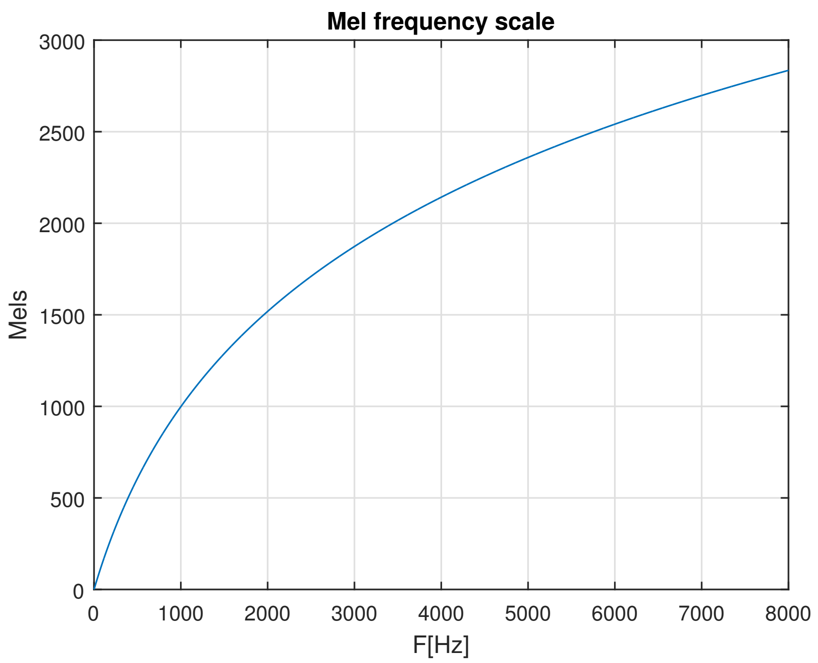


Figure 9.1: Mel scale

In using mel filterbanks, we divide our Fourier transformed signal into triangular bandpass filters spaced along the mel-scale, with the first filter being very narrow. The magnitude coefficients are then binned by correlating them with each triangular filter. Here binning means that each FFT magnitude coefficient is multiplied by the corresponding filter gain and the results accumulated. Thus, each bin holds a weighted sum representing the spectral magnitude in that filterbank channel [2].

[1]Reproduced from HTKbook with permission from the HTK team [2]

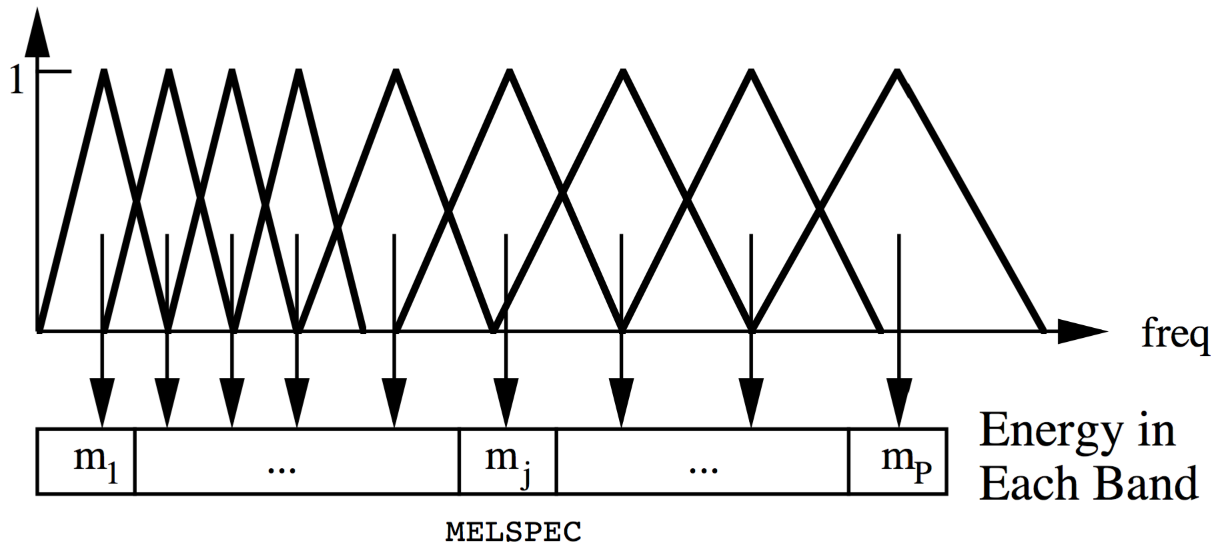


Figure 9.2: Log mel-scale filter bank[1]

Chapter 10

Implementation

The main pipeline that consists of feature extraction, data processing and interfacing with the software suites HTK, Kaldi and SVM^{light}, is written by professor Sabato Marco Siniscalchi of Kore University of Enna. Of course, adjustments had to be made to accommodate Norwegian data, and also to make custom experiments. Additional scripts have been written by the author, for data processing and result evaluation, as well as top-level experiment scripts. The main scripting languages used are Bash and Perl. Some Matlab code is also used. In the following sections the implementation, along with some features of the software suites, will be examined. In order to follow the plot, the implementation will be outlined from "start to finish", with the required software knowledge along with it. We start with label file manipulation in HTK, feature file generation before giving them as input to the neural network in Kaldi. After this, decoding will be explained.

10.1 Hidden Markov Model Toolkit

The Hidden Markov Model Toolkit (HTK) is a suite of tools mainly written for HMM-based word recognition. As such, command-line tools are available for feature extraction, Viterbi training and realignment, the Forward-Backward algorithm, word network and lattice creation, dictionary and label file manipulation, and more. The toolbox is quite large, so only the tools used in this thesis will be outlined below, according to the documentation provided by developers of HTK [2]. Most of the tools take so-called *standard options*, names of files to be processed or output, and behavioural parameters. These parameters can be written in a *configuration file*

10.1.1 HCopy

HCopy copies one or more data files to a designated output file. It can also convert supported data formats into the HTK format, concatenate or segment data files and parameterise the result. In this thesis, HCopy is used to parameterise `.wav` or `.raw` files into logbank features. The general form of an HCopy command is:

```
    HCopy -T 1 -C config -S list.scp
```

Where `-T 1` a standard trace option with a bit string, where the 1 is set to give us basic progress reporting from the tool. `-C config` means that HCopy takes a config file which is written beforehand, which sets a number of parameters. Lastly, `-S list.scp` takes in a list of files to process. An example of a config file which is used in this thesis is seen below.

```
SOURCEKIND   = WAVEFORM
SOURCEFORMAT = WAV
```

```
SOURCERATE   = 1250
BYTEORDER    = VAX
TARGETFORMAT = HTK
TARGETKIND   = FBANK_D_A_E

LOFREQ       = 64
HIFREQ       = 4000
NUMCHANS     = 15
USEPOWER     = T
USEHAMMING   = T

PREEMCOEF    = 0
TARGETRATE   = 100000
WINDOWSIZE   = 250000
SAVEWITHCRC  = F

NUMCEPS      = 12
WARPFREQ     = 1
WARPLCUTOFF  = 3400
WARPUCUTOFF  = 3400
```

Where we have a waveform encoded in the `.wav` format as input. One of the idiosyncrasies of HTK is that the sample period (here denoted by sourcerate) must be written in 100 nsec units. So if we have, as in our case, a source file with sample rate 8 kHz, this means that the sourcerate is $\frac{1}{8000*100*10^{-9}} = 1250$. The `SOURCEFORMAT` is self-explanatory, and in the frontend and backend, `WAV` and `RAW` are used. Note that the `.raw` format is headerless, meaning no sample rate info is embedded in the file, and so it must be explicitly set in the config file before using HCopy. `TARGETRATE` sets the frame rate, which means that each frame of 10 ms will later be given a label. `WINDOWSIZE` and `TARGETRATE` are independent; the window size is the segment of the waveform that is used to determine the parameter vector for the frame in question. A standard choice is 25 ms, where we have some overlap over each frame. `BYTEORDER = VAX` is used to ensure correct byte order when reading binary files. The output will be encoded into the HTK format. A crucial parameter is `TARGETKIND`, which determines what sort and how many parameters will appear in the resulting vector for each frame. Here, `FBANK` means that we are using log filterbank coefficients. The options `_D_A_E` adds delta, acceleration and energy coefficients. `LOFREQ` and `HIFREQ` denotes lower and upper frequency cutoffs for the filterbank analysis, respectively. We have `NUMCHANS = 15`, meaning 15 critical bands for the filterbank analysis. Setting `USEPOWER` and `USEHAMMING` to true means that we are using power rather than magnitude of the Fourier transform in the frequency bins, and the Hamming window over 25 ms. We use no preemphasis coefficients with the samples. `SAVEWITHCRC` can be set to attach a checksum to the output parameter file. The last four lines are only relevant if we are using MFCC features. In this thesis, `FBANK` is used exclusively.

10.1.2 HLEd

HLEd is used for manipulating label files. An HLEd command extensively throughout this work is:

```
    HLEd -n dict_CI -l '*' -i  attributeCI.mlf place.led phone.mlf
```

Where `-n` outputs the new label names to a filename specified as `dict_CI`. `-i` is prepended before the name of the new attribute level master label file. `-l '*'` simply gives us an mlf with the pattern `"*/xxx"` which means that it is independent of the location of the data files that correspond to each sentence in the mlf. `place.led` is the file which maps phonemes to attributes. An example of this is seen below:

```
RE labial b f m p v w
RE coronal d dx en l n s t z er r
RE palatal y jh ch zh sh
RE low aa ae aw ay oy
RE dental dh th
RE mid ah eh ey ow
RE high ih iy uh uw
RE velar g k ng
RE glottal hh
RE sil pau
ME sil sil sil
```

Each subsequent phoneme following the command `RE <attribute>` is mapped to that attribute. To illustrate:

```
RE <attribute> <phoneme1> <phoneme2> <phoneme N>
```

Also, the command `ME sil sil sil` merges any repetition of `sil` into one instance. An example of a context independent dictionary produced by HLEd is:

```
coronal
dental
glottal
high
labial
low
mid
palatal
sil
velar
```

This is simply a list of all attributes that result from the conversion from a phone level mlf to an attribute level mlf. To convert the context independent mlf into a context depedent one, for example right context (RC), we could execute the following:

```
    HLEd -n dict_RC -l '*' -i attributeRC.mlf RC.led attributeCI.mlf
```

Where we have essentially repeated the process from before, only this time, the file `RC.led` is simply composed of:

```
#TC
ME sil sil sil
RC
#LC
```

Here, the commented lines are the other possible configurations. `TC` meaning triphones (both left- and right contexts), and `LC` meaning left context. The uncommented lines will be evaluated by HLEd. The merge command is included here, also.

To summarize, HLEd gives us the required context independent or context dependent attribute level label files. This can be done in batch mode, and an example script to do this is found in the appendix. A script is also provided to combine mlfs of different languages into one. Care must be taken to start an mlf with

```
#!MLF!#
```

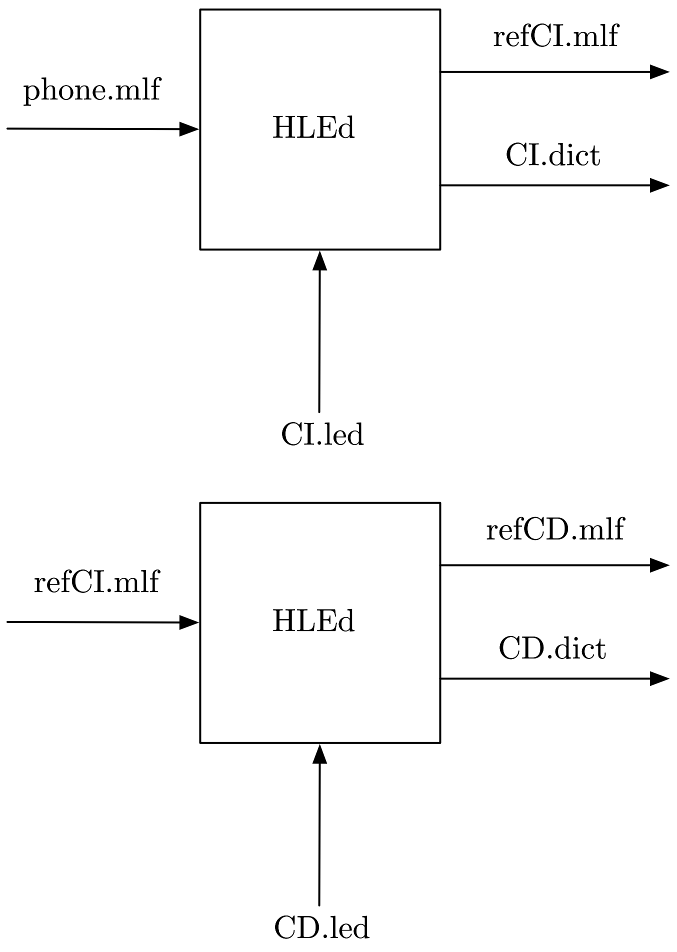


Figure 10.1: Making a context dependent mlf

and to end each sentence/file with a dot

```
.
```

What we need for the neural network, however, is information about both features and labels in *one* file. We will learn more about in the next section concerning pfiles.

10.1.3 HVite

Realignment is done with ICSI-tools, described later. It could also be done in HVite. We will however, only use HVite for decoding. HVite is a general purpose Viterbi word recogniser (or in our case, attribute recogniser). Here too, we use configuration files as input to the tool. In addition, we need a word lattice, a dictionary, htk feature files and a list of HMMs. The appropriate command for decoding is shown below.

```
HVite -l '*' -i output.mlf -C config_net -w wdnet -s 1.0 -p 0.0 -S ...
    htk_filelist.scp -H hmmproto dict hmmlist
```

Here, we have no penalty and no grammar scaling. Throughout the work, these two parameters were kept this way. They could potentially improve decoding with some experimenting. For evaluation of the results, the arguments `-o S` need to be added. The reason is that a script written by the author requires the output form that is obtained with these arguments.

10.2 Strbut2HTK

Strbut2HTK is a little program written in C, by Marco Siniscalchi. It "Gaussianises" the log-posteriors from the ANN. This is done to fit it with the HTK framework with GMMs. We have that

$$P(\mathbf{O}|\lambda) = \frac{1}{2\pi^{\frac{N}{2}}} \left|\Sigma\right|^{-\frac{1}{2}} exp\left(-\frac{1}{2}(\mathbf{O}-\mu)^T \Sigma^{-1} (\mathbf{O}-\mu) \right) \tag{10.1}$$

and

$$log\, P(\mathbf{O}|\lambda) = log\left(\frac{1}{2\pi^{\frac{N}{2}}} \left|\Sigma\right|^{-\frac{1}{2}} \right) - \frac{1}{2}(\mathbf{O}-\mu)^T \Sigma^{-1} (\mathbf{O}-\mu) \tag{10.2}$$

where N is the observation dimension, and we assume $\mu = \mathbf{0}$. Thus:

$$log\, P(\mathbf{O}|\lambda) = GCONST - \frac{1}{2}\mathbf{O}^T \Sigma^{-1} \mathbf{O} \tag{10.3}$$

GCONST is a variable in HTK, and is calculated by multiplying the determinant of the covariance matrix by $(2\pi)^N$. We set this to zero in all experiments. The variance for attribute i is set to 1, and the variance for the other attributes is set to a large number (practically infinite). In addition to this, Strbut2HTK takes in a posterior probability X, such that

$$O = \sqrt{-2P(\lambda|X)} \tag{10.4}$$

and

$$log\, P(O|\lambda) = -\frac{1}{2}O^2 = P(\lambda|X) \tag{10.5}$$

10.3 ICSI

10.3.1 Pfiles as feature files

Pfiles are a convenient format with a long history, originating in the International Computer Science Institiute (ICSI). The format is also quite simple, with the first column containing the sentence or file number, the second column containing frame number within the file, followed by N number of features according to our implementation, and ended by a label. The number of rows is of course dependent on how many files and frames we have. An example is seen below.

```
0 0 11.784 13.2271 12.7112 9.97669 9.51784 8.93385 9.32022 9.84839 ...
    9.77048 9.75651 9.52307 9.63518 9.96957 10.9014 9.98486 0.226576 ...
    0.19763 -0.0563804 -0.0555905 -0.378597 0.0535857 -0.000593472 ...
    -0.0215122 -0.072628 -0.176922 -0.176556 -0.0188554 -0.0307481 ...
    0.0389602 -0.0674485 -0.00179329 0.000112391 -0.0894245 0.0179495 ...
    0.0142691 0.050391 -0.0288078 0.0534145 0.0435024 0.0228186 0.102601 ...
    0.0817752 0.0280314 0.0438226 -0.0130251 -0.0254204 -0.0131989 ...
    -9.83435e-05 8
```

10.3.2 Pfiles as posterior files

The format can in principle encode any info vector in binary, provided proper formatting at the human-readable level. Underneath, an example of a log-posterior vector is seen. As before, sentence and frame numbers precede the posteriors.

```
0 0 -6.77103 -6.104 -6.98091 -7.79767 -7.30796 -8.22175 -12.3695 ...
    -12.7881 -12.4091 -9.38573 -8.82207 -8.04157 -5.81232 -5.12931 ...
    -6.24505 -11.6171 -10.1496 -8.67616 -9.29866 -8.76515 -7.28647 ...
    -11.0553 -11.0238 -10.7244 -1.02345 -1.14843 -1.29596 -8.82008 ...
    -8.63673 -9.39287 -5.2201 -4.60787 -4.14726
```

10.3.3 Tools

The ICSI utilities are numerous, and explaining each of them in detail is beyond the scope of this report. But the tools that are used here, will now be outlined in broad strokes.

- **pfile_create**: Creates a pfile given a feature stream and labels, or alternatively, posteriors.
- **feacat**: Feature stream converter. Can convert and concatenate HTK files into pfiles.
- **pfile_patchlabels**: Patches labels, or appends them to an existing pfile.
- **pfile_info**: Outputs number of sentences, frames, labels and features in a pfile.
- **pfile_labs**: Can expand a labelled pfile into a tristate uniformly segmented label sequence
- **pfile_print**: Can print a pfile into a human-readable format
- **pfile_realign**: Performs Viterbi alignment with
  1. An input labelled pfile with features
  2. An input posterior vector
  3. A realigned pfile as output

10.4 Kaldi

Kaldi is an open source toolkit written in C++, originating in a project at Johns Hopkins University. It includes a vast amount of code which together constitutes a package for extracting features, training neural networks, RBMs and more. It also has some interfacing capabilities with other software suites such as HTK. One of the developers, Karel Vesely, has written an implementation of Kaldi in a set of Bash scripts. They have since been slightly modified by Marco Siniscalchi to fit with his framework, and the resulting package is used in this thesis. Only small parameter changes have been made by the author. Explaining how the bits and pieces of the Kaldi framework fit together is far beyond the scope of this thesis, and the reader is referred to its documentation page at `http://kaldi-asr.org` for in-depth explanations. Vesely's implementation will, however, be explained in broad strokes.

10.4.1 Karel's implementation, nnet1

The implementation of Kaldi that is used, is Karel Vesely's code, which is mainly (but not exclusively) a set of shell scripts that calls on Kaldi functions, in a proper order. It is best documented at `http://kaldi-asr.org/doc/dnn1.html`. The RBM parameters are tuned on a 100 hour Switchboard set, which is far larger than what we have at our disposal. In this respect, it would be prudent to do one's own tuning, but it was deemed to costly to spend time on in this thesis. Vesely's original recipe can also do feature extraction, but for our purposes, this is done in HTK. The top level scripts used are [28]:

- **pretrain_dbn.sh** - which implements unsupervised RBM training with CD1 contrastive divergence. Gauss-Bernoulli units are used in the first RBM, followed by stacks of Bernoulli-Bernoulli units. Variance in the training data is compared to the variance of the reconstruction data in a minibatch, and if the latter is $> 2\times$ larger, the weights are shrinked and learning rate temporarily reduced.

- **train.sh** - fine-tunes the standard backpropagation step, with weights initialized by the RBMs. Output layer is initialized randomly. Mini-batch stochastic gradient descent with sigmoid hidden units and softmax output is used. Standard is learning rate of 0.008, size of minibatch 256. The cross validation set is used to do early stopping with a cross-entropy objective function, to prevent overfitting.

Before being input to the RBM, 5 frames are added from past and future with respect to the current frame, so that the current frame is X_{t-5}^{t+5}. As mentioned before, this gives the neural network some context to work with, so that each training vector isn't just dependent on the current frame. Note that the delta and acceleration coefficients also provide some indication of the trajectory of the speech signal.

10.4.2 Folder structure

Since we do feature extraction outside of the Kaldi environment, some intermediate steps has to be done before training. We will later learn more about the pipeline that leads us here. For now, let's look at the folder structure. The label files are called "alignment files" in Kaldi's terminology. As an example, for the place attributes, the labels are kept compressed in a folder whose name indicates its set association.

```
	kaldi-trunk/egs/ogi/s5/exp/place_cd_ali_dev/ali.1.gz
```

The `exp` folder structure is remade each time we train a new neural network. Feature files are kept in a folder named

```
    kaldi-trunk/egs/ogi/s5/data-fbank
```

which is, of course, also remade for each new neural network.

10.4.3 Running training scripts

A pretraining can be invoked with the following command

```
    dir=exp/place_pretrain-dbn

    $dir/log/pretrain_dbn.log steps/nnet/pretrain_dbn.sh --rbm-iter 3 ...
        data-fbank/develop/ $dir
```

where `pretrain_dbn.sh` is the pretraining script, iterations are set with `--rbm-iter`, and as mentioned, feature files is found in `data-fbank/<set>`. For the backpropagation layers, a training script can be invoked as:

```
$dir/log/train_nnet.log steps/nnet/train.sh --feature-transform ...
    $feature_transform --dbn $dbn --hid-layers 0 --learn-rate 0.008 ...
    data-fbank/train data-fbank/develop data-fbank/lang $ali_train ...
    $ali_cv $dir > train.log&
```

Where the pretrained RBM location is specified with `--dbn`. Here, the label files are required, since it is not an unsupervised algorithm, like the pretraining is. One peculiar thing about the setup is that the hidden layers are set in advance, in `train.sh`. Setting `--hid-layers` to zero ensures that no extra layers beyond this specification are appended. Another thing is the `num-tgt` parameter in `train.sh`. It needs to be set according to the number of attributes we are discriminating, but also according to which context. A CI attribute would only need one class, but with a RC system, we need $(Number\ of\ attributes)^2 + oth$ classes. If we further add tristate classes, we need $((Number\ of\ attributes)^2 + oth) * 3$ classes. Failing to set the proper number of targets will (luckily) crash the training process. Note that the "oth" shouldn't really be modelled as a tristate attribute. More on this in the Experiment section.

10.4.4 Resulting files

For each completely trained network, two files need to be kept for further use. One is `final.feature_transform` and the other is `final.nnet`. The former can for example look like

```
num-components 3
input-dim 48
output-dim 528
number-of-parameters 0.001056 millions
component 1 : <Splice>, input-dim 48, output-dim 528,
  frame_offsets [ -5 -4 -3 -2 -1 0 1 2 3 4 5 ]
component 2 : <AddShift>, input-dim 528, output-dim 528,
  shift_data ( min -15.1179, max 4.35164e-05, mean -4.35758, variance ...
      41.6607, stddev 6.45451, skewness -0.821935, kurtosis -1.30548 )
component 3 : <Rescale>, input-dim 528, output-dim 528,
  scale_data ( min 0.256994, max 70.4746, mean 4.20308, variance ...
      108.776, stddev 10.4296, skewness 5.53381, kurtosis 31.4043 )
```

for a CI training of one state classes. The input dimension is 48, according to our number of logbank features and additional coefficients. The output dimension is 528 because of the splicing of feature vectors: $48 * (1 + 5 + 5) = 528$. The feature transform does three things:

- Splices features

- Shifts features to have zero mean
- Scales features to have unit variance

For the same network, `final.nnet` carries contains the weights and statistics about weights that have been learned in the training process:

```
num-components 14
input-dim 528
output-dim 11
number-of-parameters 22.0877 millions
component 1 : <AffineTransform>, input-dim 528, output-dim 2048,
  linearity ( min -1.80865, max 1.3898, mean -0.000922673, variance ...
      0.00299962, stddev 0.0547688, skewness 0.259902, kurtosis 17.8454 )
  bias ( min -5.36165, max -0.302768, mean -3.26977, variance 0.335196, ...
      stddev 0.578961, skewness 0.766596, kurtosis 4.81403 )
component 2 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 3 : <AffineTransform>, input-dim 2048, output-dim 2048,
  linearity ( min -1.50755, max 2.43644, mean -0.00271999, variance ...
      0.00287751, stddev 0.0536424, skewness 1.23347, kurtosis 41.5754 )
  bias ( min -42.4032, max 59.3772, mean -4.21697, variance 15.2342, ...
      stddev 3.9031, skewness 0.527746, kurtosis 71.9775 )
component 4 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 5 : <AffineTransform>, input-dim 2048, output-dim 2048,
  linearity ( min -3.0185, max 3.12063, mean -0.00268391, variance ...
      0.00259756, stddev 0.0509663, skewness 3.31557, kurtosis 155.823 )
  bias ( min -12.1571, max 3.80626, mean -3.60882, variance 0.732204, ...
      stddev 0.855689, skewness -0.342962, kurtosis 14.5589 )
component 6 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 7 : <AffineTransform>, input-dim 2048, output-dim 2048,
  linearity ( min -2.77141, max 3.75801, mean -0.00189897, variance ...
      0.00204982, stddev 0.0452749, skewness 4.48462, kurtosis 225.94 )
  bias ( min -16.1135, max 2.68882, mean -3.91713, variance 2.09377, ...
      stddev 1.44699, skewness -2.8918, kurtosis 15.1214 )
component 8 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 9 : <AffineTransform>, input-dim 2048, output-dim 2048,
  linearity ( min -2.69317, max 3.05688, mean -0.00198038, variance ...
      0.00187974, stddev 0.043356, skewness 5.69599, kurtosis 279.882 )
  bias ( min -8.68595, max 2.35644, mean -3.7733, variance 0.464799, ...
      stddev 0.681762, skewness 0.888941, kurtosis 14.4933 )
component 10 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 11 : <AffineTransform>, input-dim 2048, output-dim 2048,
  linearity ( min -3.07659, max 2.70959, mean -0.00294674, variance ...
      0.00178236, stddev 0.042218, skewness 4.27828, kurtosis 272.425 )
  bias ( min -17.6855, max 3.14401, mean -3.85512, variance 1.26257, ...
      stddev 1.12364, skewness -3.18669, kurtosis 22.9487 )
component 12 : <Sigmoid>, input-dim 2048, output-dim 2048,
component 13 : <AffineTransform>, input-dim 2048, output-dim 11,
  linearity ( min -1.8421, max 2.12262, mean 0.00106659, variance ...
      0.0411283, stddev 0.202801, skewness 0.45185, kurtosis 10.6692 )
  bias ( min -0.963688, max 0.696117, mean -8.45302e-07, variance ...
      0.362841, stddev 0.602363, skewness -0.503332, kurtosis -1.40392 )
component 14 : <Softmax>, input-dim 11, output-dim 11,
```

Here, we have 11 output classes, since it is a CI training of 10 attributes + "oth". The number of neurons per layer is set uniformly to 2048, which explains the output and input dimensions. The layers are connected with affine transforms followed by sigmoid layers. Note that estimates are made for Cepstral Mean Variance Normalization (CMVN) internally in the Kaldi scripts, and this is why normalization is not done in feature extraction, prior to the training.

Having these two files, we are ready to use them for testing and classification.

10.5 SoX

Sound eXchange (SoX) `http://sox.sourceforge.net` is, as the developers call it, a "swiss army knife" of sound processing programs. It has ready-made implementations of downsampling and upsampling of audio, play and record functionality, and filtering included. The tools can be called upon in the command line, facilitating batch methods to processing databases of audio. Some commands used in this thesis will now be outlined. Some examples to do batch downsampling and filtering is shown in the appendix, along with a handy script for calculating the total duration of a given set of input audio files.

10.5.1 Downsampling

To downsample a wav file, a simple command can be called:

```
sox $foo.wav $bar.raw rate -s -a 8k
```

Note that since the input is a wav file, there is no need to specify input sample rate. Only the output sample rate needs to be specified, which in this case is 8 kHz.

10.5.2 Filtering

Since the Callfriend database that is used in this work is encoded in the μlaw format, which is bandlimited to $300 - 3400$ Hz, it is prudent to filter other formats in addition to downsampling it.

```
sox -r 8000 -e signed -b 16 -c 1 $foo ${bar}.raw  sinc 250-3450
```

The above command filters a 16 bit, one channel signed raw file with a sinc filter, with cutoffs at 250 and 3450 Hz, which is near the Nyquist frequency at 8 kHz. Here, telephone quality is assumed to be $300 - 3400$ Hz. The sinc filter used in this work is seen below. The upper and lower cutoffs were chosen such that they are not too "strict", and that they do not remove too much frequency content in the upper band. Some early tests were made both subjectively through listening to filtered files, and through a test with the TIMIT database, that showed better recognition with the sinc filter shown below, than with a more constrained cutoff. Even setting the upper cutoff to $3400Hz$ was noticeably more "tinny" than what the Callfriend sounds like, unprocessed.

10.6 SVM light tools

SVM light tools is an implementation of SVMs, written in C by Thorsten Joachims. It uses fast optimization algorithms, and is well suited for classification problems. The two main functions we use are outlined in the next sections.

10.6.1 Training a model

All we need is a simple command, where the number following `-j` determines the cost-factor, by which training errors on positive examples outweigh errors on negative examples [1].

```
svm_learn -j ${weight} ${set}.dat theModel
```

A set of training examples are provided in the `.dat` file, and the trained model is output. The weight factor is included to account for the imbalance of positive and negative examples in a

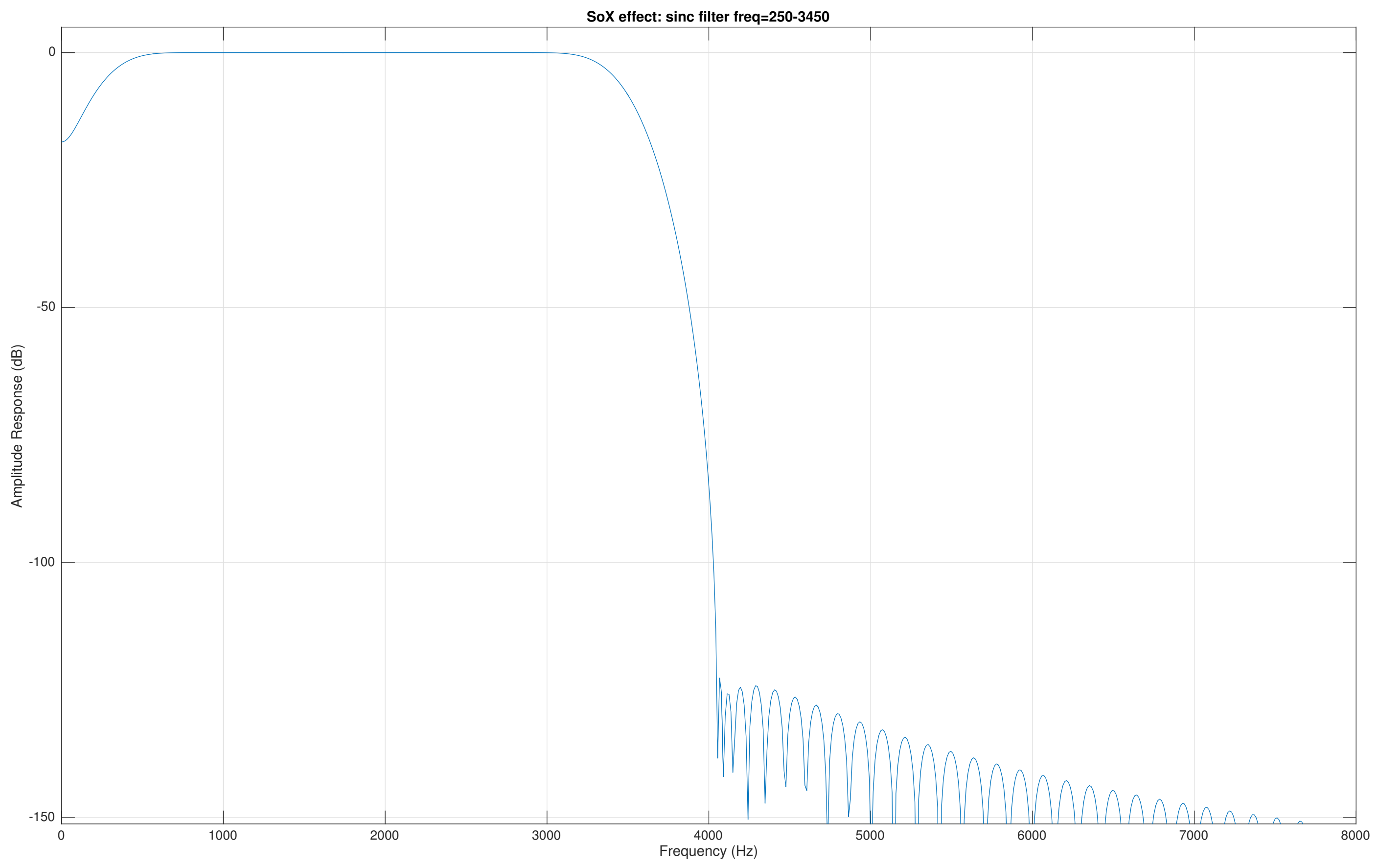


Figure 10.2: Telephone quality filter

one-versus-all training [22]. This can also be called multiplying by cost factors, where training errors on positive examples outweigh errors on negative examples [1]. Default is 1.

10.6.2 Testing a model

Again, a simple command is used for testing the model.

```
svm_classify test.dat theModel ${set}.pred
```

This is about as much knowledge that is needed to proceed with this tool. For further information, the reader is referred to the tool's website at at http://svmlight.joachims.org.

Chapter 11

Experiments: Frontend

The use of software has been explained in the last chapter. Now, we will see how it all ties together, while going through some of the methodology. We will also see some results and discuss them. The goal was to compare manner and place attributes and the performance of the system using them, but first the system needed to be proven to work for one of them, as a first step. The bulk of the work on the frontend was dedicated to

1. Experimenting with Siniscalchi's setup, connecting the pieces of the pipeline
2. Appropriating data for processing
3. Creating CI and CD master label files
4. Fitting the pipeline to tristate HMM decoding
5. Implementing Viterbi realignment and retraining the networks
6. Evaluating the realigned networks and comparing them to ordinary networks

11.1 Databases

Three databases were used for experiments in the frontend. The "story" part of the OGI database is used for training the UAR. Six different languages are included in the training, ensuring a diverse set. The selection for OGI data was kept as in Siniscalchi's setup. The Norwegian testing data is taken from part 1 of NAFTA, which consists of transcribed, planned out sentences. As such, it is good for testing the UAR. TIMIT is a purely English database with transcriptions. It is sampled in 16 kHz, and had to be downsampled to 8 kHz as well as filtered to resemble OGI's μlaw format. The same goes for NAFTA, except it is sampled at 48 kHz.

Table 11.1: Durations in hours for OGI training sets

English	2.64
German	1.30
Hindi	0.89
Japanese	0.83
Mandarin	0.79
Spanish	1.38
Total	7.83

Table 11.2: Durations in hours for OGI CV sets

English	0.21
German	0.10
Hindi	0.06
Japanese	0.06
Mandarin	0.04
Spanish	0.10
Total	0.57

Table 11.3: Durations in hours for test sets

NAFTA	3.29
TIMIT	1.44

11.2 Feature extraction

Feature extraction has been explained in the Implementation chapter, where we used HCopy to extract log melbank features. The example of a parameter set for HCopy shown there, is used for every language and dialect. To automate the process, Marco Siniscalchi has written a script called `ogi_place.sh`. It takes in a list of training, testing and cross-validation files, a dictionary and an MLF. It calls on HCopy to extract features with the configuration outlined in the implementation chapter. These features are concatenated to produce a single file per data set (training,test or cross-validation). It not only produces feature files in the pfile format, with labels. It also checks for discontinuities in the input MLFs, and labels them with the "oth" mark. With this comes some warnings:

```
WARNING: Discontinuity - frames 4845:4877 in file GEcall-101-G.story-bt.
WARNING: The feature file 'GEcall-101-G.story-bt' is shorter than ...
    labels: 4881 / 4882.
```

These discontinuities are often silence that are not labelled from frame zero to the start in the original transcriptions. They can also be caused by mappings done to the originals in converting them to an HTK friendly format, or in compacting the original phone set. Below we see an excerpt of the phone level German OGI MLF that is used as input to HLEd:

```
482520000 483890000 A:
483890000 484560000 d
487700000 488240000 pau
```

The raw, original transcription included in the OGI database, looks like:

```
48252 48389 A:
48389 48439 dc
48439 48456 d
48456 48770 .br
48770 48824 .pau
```

Here, we see that `dc` and `d` has been compacted together. Notice that some extra zeroes are appended to fit with the HTK time format mentioned in the implementation chapter. The phoneme `.br` has also been cut out, leaving us with a discontiuity before the `pau` segment (which will be replaced with `sil` later on). There are many such instances in the OGI mappings. Regrettably, they were not provided with the code, and so a thorough review of them could not be performed. The "oth" label is primarily intended to represent sounds other than intelligible speech or silence (such as coughing), and so training an ANN with speech or silence labelled as "oth" is thought to potentially harm the training, albeit perhaps to a negligible degree. After pfiles are made, with features and labels embedded, they are moved to a new folder, `Pfile2Kaldi`, for converting the pfiles to the Kaldi format. A script to perform all of this is condensend into `eyscript.sh`. There, three types of files are made. One is a file that gives the borders between different sentences. `ali_train_labs.scp` is an example.

```
GEcall-1-G :11
GEcall-10-G :13253
GEcall-100-G :25593
```

Each file is given a number which indicates where the corresponding feature and frame sequence starts, in the `.ark` file. The `.ark` file is a generic format that can contain posteriors, features or labels. An outtake from a label sequence:

```
GEcall-1-G 79 79 79 79 79 79 79 99 99 99 29 29 29 29 29 29 4 4 4 4 4 4 ...
    4 4 39 39 39 39 39 39 39 99
```

It starts with the name of the file, followed by the labels. The frame posteriors from an English file of 33 posteriors are similarly listed as:

```
ENcall-167-G  [
  -8.428092 -6.21559 -7.195301 -8.495845 -7.85171 -9.073958 -14.80102 ...
      -14.32045 -13.61943 -8.056107 -9.656265 -8.614976 -6.54748 ...
      -5.498335 -9.241637 -11.98215 -12.986 -11.51024 -9.811481 ...
      -10.03226 -7.903199 -11.67692 -11.08974 -11.50559 -1.216823 ...
      -0.6279603 -1.884112 -10.92035 -8.802866 -11.24253 -8.064992 ...
      -7.670111 -4.984917
```

If we want a tristate training, we must divide the posteriors into three uniform segments to begin with. This is done with `pfile_labs -st 3 -i train.pfile -o trainthree.pfile`. A modified version of `eyscript.sh`, `eyscript_tristate.sh`, adds this to the process, before doing the same conversion into Kaldi format, mentioned before. A last step encodes the features into binary, for use in ANN training.

```
copy-feats ark:train.ark.txt ark,scp:train.ark,feats.scp
```

The labels are also compressed into the `.gz` format.

```
gzip -c ali_train_labs.ark > ali.1.gz
```

11.3 ANN training

For each data set (training, test and cv), we now have a folder containing the file position sequence `feats.scp`, and the features `<dataset>.ark`. The training can now commence. We use the commands mentioned in chapter 10 to start pre-training. The parameters set in `pretrain_dbn.sh` are set to be the same for *all* experiments. They are listed in table 11.4, where L2 penalty refers

Table 11.4: RBM parameters

Num hidden layers	6
Num neurons per layer	2048
Standard dev GBRBM	0.01
Standard dev other RBMs	0.1
Input type to first RBM	Gaussian
RBM iterations	3
RBM learning rate for Gaussian units	0.01
RBM learning rate for other units	0.4
L2 penalty	0.0002

to a penalty function that is half of the sum of the squared weights times a coefficient called the weight-cost. This is called weight decay, which limits the growth of the weights [13]. The number of neurons and layers are the same as in the pretraining script, for `train.sh`. The target number is set according to number of classes. As mentioned before, we always splice 5 features on each side of the current frame before inputing it to the GBRBM. All of the steps needed to run a complete training is included in the script `masterplace.sh`, found in the appendix. It takes arguments CI, RC, LC or TC to train the network with a requested context, and it can also train with either single- or tristate attributes. There is an extra binary argument called "compacted" which refers to the LEDs. If attributes are combined to produce a lower resolution class set, this is referred to as compaction in the script. Unfortunately, there was not time to experiment with this, and the full resolution was kept throughout all experiments. An option to bypass ANN training can also be chosen, mostly for control purposes. One benefit of automating training with shell scripts is of course time saving, but it also ensures that training is done correctly each time, without forgetting important steps. Care must be taken, though, to check for anomalies during and after training. For example, one issue mentioned in the implementation chapter, is that adding –num-layers=6 to the train.sh command appends unwanted layers to the network, and was only seen after training the network. This has since been avoided.

11.4 Realignment

Realignment is only done for the tristate networks. For them, we need to find better segmentation borders between attribute states. This is, as mentined, done with the Viterbi algorithm implemented in the `pfile_realign` binary. The realignment training is also automated in `realign_train.sh`, found in the appendix. It is best understood by reading the code, but a rundown will be done below. Note that in this example, we use the place attributes with right context, giving 303 classes including "oth".

Algorithm 11.1: Realignment

```
1  Do feature extraction, pfile conversion to kaldi format and tristate conversion, and train
   the first network with masterplace.sh. Keep the features in a Kaldi folder
2  Do a forward pass for each data set (train, test, cv) through the obtained network
3  Copy the posteriors obtained after the forward pass to a temporary folder
4  Copy the pfiles that were converted to kaldi format and used to train the previous net-
   work, to a temporary folder
5  Convert the posteriors obtained after the forward pass to pfile format
6  Calculate class priors over the frames from the previous pfile
7  Realign with pfile_realign, using the pfile formatted posteriors, the previous pfile con-
   taining features and labels
8  Convert the realigned pfiles (train, cv, and test) to kaldi format, using only the new labels.
   The features from the first step is kept and reused through the process, since they do not
   change.
9  Train a new ANN using the new segment borders
10 Repeat while noticeable improvement
```

An example of how realignment changes the segment boundaries is seen below. Now, it may not make much sense to model silence as a tristate attribute either. But the example at least illustrates what happens to an attribute when its segment borders are realigned. The same is done for the other attributes.

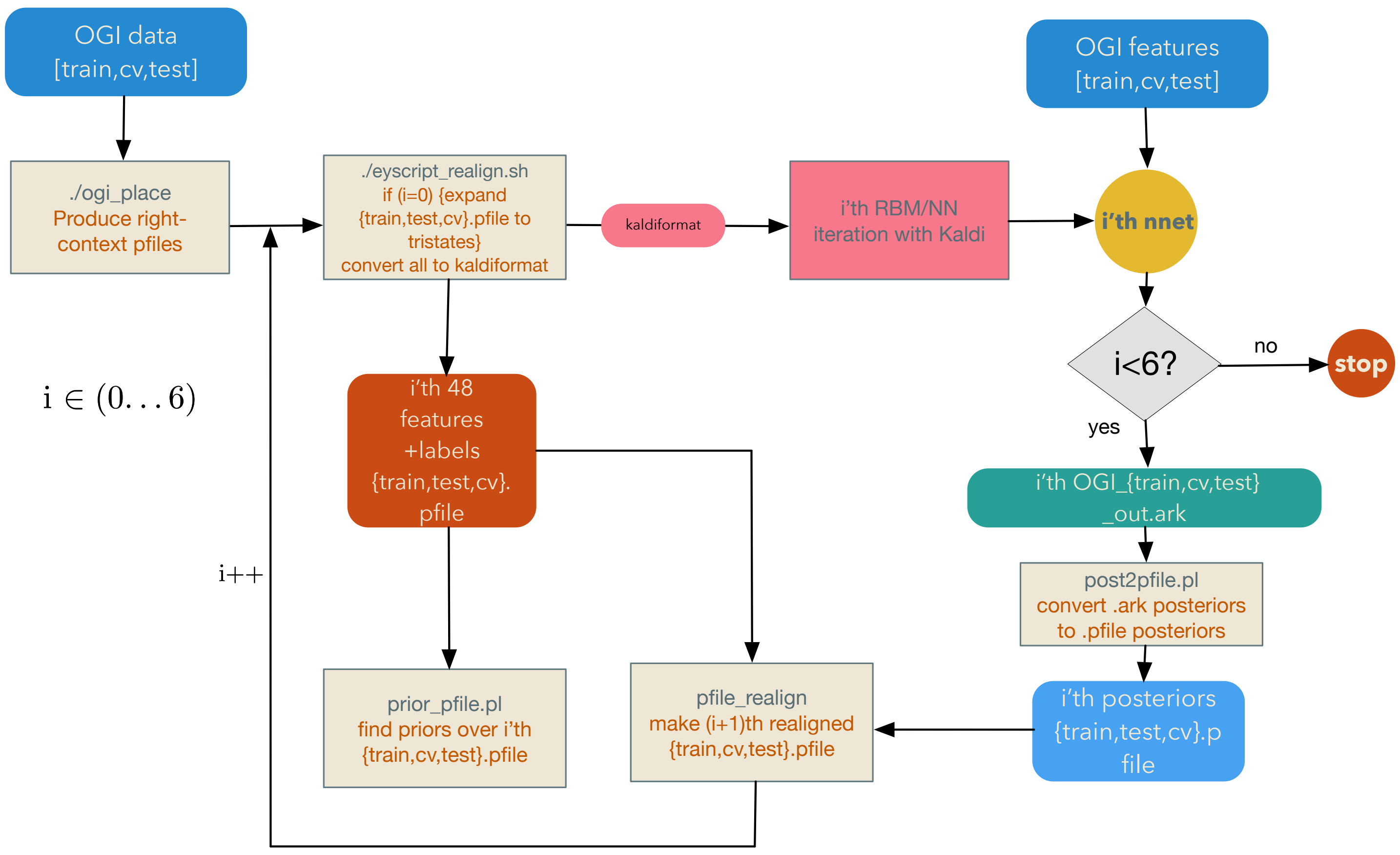

OGI data
[train,cv,test]
./ogi_place
Produce right-context pfiles
./eyscript_realign.sh
if (i=0) {expand {train,test,cv}.pfile to tristates}
convert all to kaldiformat
kaldiformat
i'th RBM/NN iteration with Kaldi
OGI features
[train,cv,test]
i'th nnet
i<6?
no
stop
yes
i ∈ (0...6)
i'th 48 features +labels {train,test,cv}. pfile
i++
i'th OGI_{train,cv,test} _out.ark
post2pfile.pl
convert .ark posteriors to .pfile posteriors
prior_pfile.pl
find priors over i'th {train,cv,test}.pfile
pfile_realign
make (i+1)th realigned {train,cv,test}.pfile
i'th posteriors {train,test,cv}.pfile

Figure 11.1: Realignment procedure

Table 11.5: CI 3-state realignment of "ENcall-112-G.story-bt.wav"

Sentence nr	Frame nr	Uniform label	1st realigned label	6th realigned label
0	0	sil_1	sil_1	sil_1
0	1	sil_1	sil_2	sil_2
0	2	sil_1	sil_2	sil_2
0	3	sil_1	sil_2	sil_2
0	4	sil_1	sil_2	sil_2
0	5	sil_1	sil_2	sil_2
0	6	sil_1	sil_2	sil_2
0	7	sil_1	sil_2	sil_2
0	8	sil_1	sil_2	sil_2
0	9	sil_2	sil_2	sil_2
0	10	sil_2	sil_2	sil_2
0	11	sil_2	sil_2	sil_2
0	12	sil_2	sil_2	sil_2
0	13	sil_2	sil_2	sil_2
0	14	sil_2	sil_2	sil_2
0	15	sil_2	sil_3	sil_2
0	16	sil_2	sil_3	sil_2
0	17	sil_2	sil_3	sil_2
0	18	sil_3	sil_3	sil_2
0	19	sil_3	sil_3	sil_3
0	20	sil_3	sil_3	sil_3
0	21	sil_3	sil_3	sil_3
0	22	sil_3	sil_3	sil_3
0	23	sil_3	sil_3	sil_3
0	24	sil_3	sil_3	sil_3
0	25	dental_1	dental_1	dental_1
0	26	dental_1	dental_2	dental_2

11.5 Decoding

In all, 31 networks were trained, including all realignment iterations. The first round of training was done without English data to see how the networks fare as truly general, universal attribute recognisers. It was planned to test the networks with both Norwegian and TIMIT. The networks are:

- CI 1-state no english
- CI 3-state no english
- RC 1-state no english
- RC 3-state no english
- CI 3-state all data
- RC 3-state all data

where it is implicit that the 3-state networks were realigned. Before decoding, feature extraction using HCopy is done, with the same configuration as before ANN training, only with `.raw` files instead of `.wav` files. Then, the features that result are concatenated into the `.ark` format, using a C program written by Marco Siniscalchi, `htk2ark`.

```
htk2ark language_dataset_fea.scp language_dataset_concat.ark
```

The ark file is then input to the ANN with a forward pass, producing posterior vectors for each frame. To decode with HVite, the posteriors are converted into HTK format, with the Kaldi command

```
copy-feats-to-htk --output-ext=lop --output-dir=logposterior/ ...
    ark:language_dataset_out.ark
```

An additional step is needed to "Gaussianize" the posteriors for use with HVite (HVite is designed for use with GMMs).

```
strbut2htk $posterior_number language_dataset_lop.scp
```

We need 4 different configurations for HVite, because we train CI and RC networks, each with either single state or tristate classes. Let's look at the steps needed for configuration. First, the dictionary referred to as `dict`:

```
coronal coronal
dental dental
glottal glottal
high high
labial labial
low low
mid mid
palatal palatal
sil sil
velar velar
```

In a conventional word recognizer, the dictionary would be a set of mappings from a higher level representation to a lower one. That is, from a word down to phonemes. In our case, attributes is as low as we go, and this explains the identical columns; we have a one-to-one mapping. If we use a context-dependent system, we can either specify the expanded attributes manually, or simply add a few lines to the *configuration* file that is input to HVite. Note that if we are decoding a CI system, this is not needed. The configuration file, referred to as `config_net`, looks like:

```
FORCECXTEXP = T
ALLOWXWRDEXP = T
```

Where `FORCECXTEXP` means "force context expansion", and `ALLOWXWRDEXP` means "allow cross-word expansion". When both are set true, full cross-word context expansion will be performed. HNet is the module that does this expansion. It infers the required expansion from the dictionary and the associated list of HMMs, `hmmlist`. The grammars for CI and RC are listed in tables 11.6 and 11.7.

Table 11.6: CI attributes

Number	**Attribute**
0	coronal
1	dental
2	glottal
3	high
4	labial
5	low
6	mid
7	palatal
8	sil
9	velar

The word network can be made in HTK with HParse, which creates an `.slf` file that contains all legal sequences. The CI grammar is illstrated in figure 11.2. The grammar for RC is identical, except of course, that it has more HMM models.

Lastly, we look at the HMM definitions in `hmmproto`. The first few lines of the `hmmproto` for 3-state CI look like:

```
~o <VecSize> 30 <USER>
~s "coronal__2"
<Mean> 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<Variance> 30
1.0 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30
<GConst> 0
~s "coronal__3"
<Mean> 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<Variance> 30
1e30 1.0 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30
<GConst> 0
~s "coronal__4"
```

Table 11.7: RC attributes

Number	Attribute	Number	Attribute
0	coronal+coronal	50	low+coronal
1	coronal+dental	51	low+dental
2	coronal+glottal	52	low+glottal
3	coronal+high	53	low+high
4	coronal+labial	54	low+labial
5	coronal+low	55	low+low
6	coronal+mid	56	low+mid
7	coronal+palatal	57	low+palatal
8	coronal+sil	58	low+sil
9	coronal+velar	59	low+velar
10	dental+coronal	60	mid+coronal
11	dental+dental	61	mid+dental
12	dental+glottal	62	mid+glottal
13	dental+high	63	mid+high
14	dental+labial	64	mid+labial
15	dental+low	65	mid+low
16	dental+mid	66	mid+mid
17	dental+palatal	67	mid+palatal
18	dental+sil	68	mid+sil
19	dental+velar	69	mid+velar
20	glottal+coronal	70	palatal+coronal
21	glottal+dental	71	palatal+dental
22	glottal+glottal	72	palatal+glottal
23	glottal+high	73	palatal+high
24	glottal+labial	74	palatal+labial
25	glottal+low	75	palatal+low
26	glottal+mid	76	palatal+mid
27	glottal+palatal	77	palatal+palatal
28	glottal+sil	78	palatal+sil
29	glottal+velar	79	palatal+velar
30	high+coronal	80	sil
31	high+dental	81	sil+coronal
32	high+glottal	82	sil+dental
33	high+high	83	sil+glottal
34	high+labial	84	sil+high
35	high+low	85	sil+labial
36	high+mid	86	sil+low
37	high+palatal	87	sil+mid
38	high+sil	88	sil+palatal
39	high+velar	89	sil+velar
40	labial+coronal	90	velar+coronal
41	labial+dental	91	velar+dental
42	labial+glottal	92	velar+glottal
43	labial+high	93	velar+high
44	labial+labial	94	velar+labial
45	labial+low	95	velar+low
46	labial+mid	96	velar+mid
47	labial+palatal	97	velar+palatal
48	labial+sil	98	velar+sil
49	labial+velar	99	velar+velar

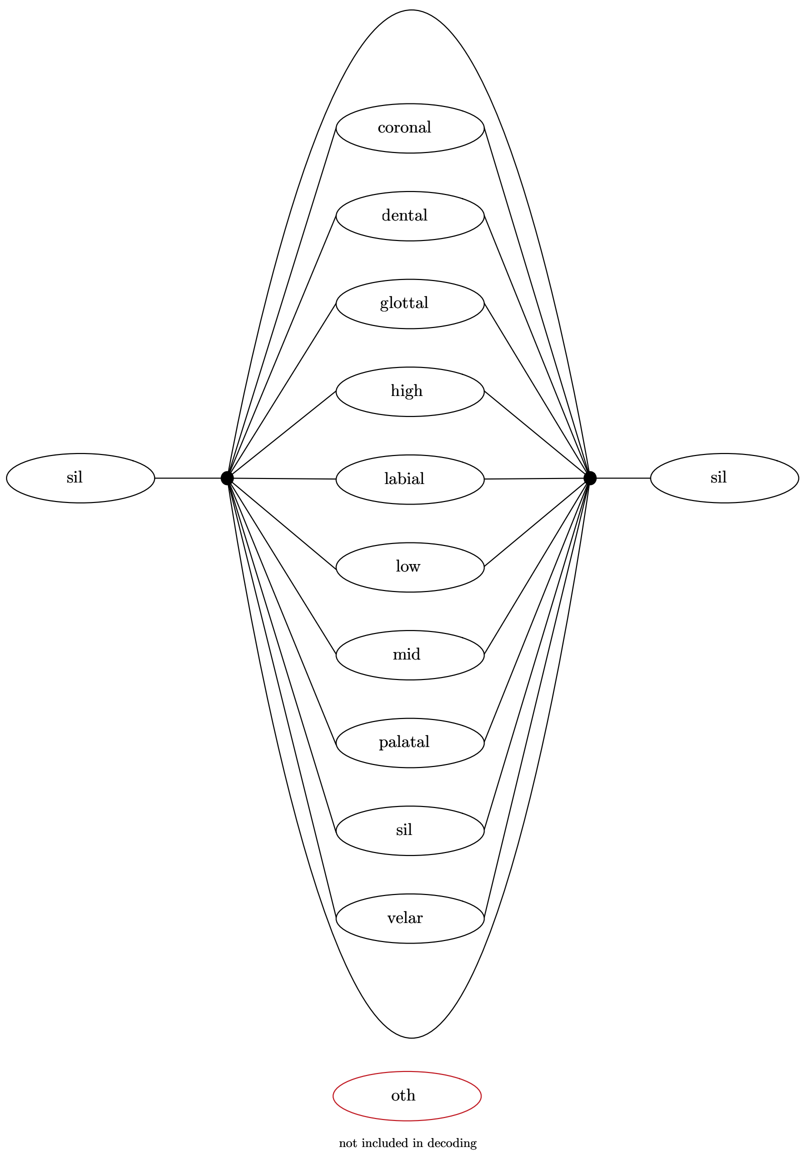


Figure 11.2: Grammar for CI

```
<Mean> 30
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
<Variance> 30
1e30 1e30 1.0 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 1e30 ...
    1e30 1e30 1e30
```

This is a "dummy" model, because we don't really use GMMs in our setup. The $\tilde{o}$ means "global option value". Following on the same line, the observation vector size is set to the number of posteriors. $\tilde{s}$ means "shared state distribution", and is followed by the state name, underscore and state number. It is numbered from 2 to 4, because states 1 and 5 are reserved start and end states, respectively. Underneath is the HMM definition of `coronal`:

```
~h "coronal"
<BeginHMM>
<NumStates> 5
<State> 2 ~s "coronal__2"
<State> 3 ~s "coronal__3"
<State> 4 ~s "coronal__4"
<TransP> 5
0 1 0 0 0
0 0.5 0.5 0 0
0 0 0.5 0.5 0
0 0 0 0.5 0.5
0 0 0 0 0
<EndHMM>
```

which includes the transition matrix. All states after the start state have 0.5 probability of either progressing to the next state, or going back on itself. These transition probabilities are not re-estimated, as in the conventional HMM training, but are rather left like this, as they have minimal effect on the decoding. The number of posteriors for all different decodings are listed below.

Table 11.8: Number of posteriors

CI 1-state	10
CI 3-state	30
RC 1-state	100
RC 3-state	300

Now we are finally ready to do decoding with HVite, with the command

```
HVite -l '*' -i output.mlf -C config_net -w wdnet -s 1.0 -p 0.0 -S ...
    htk_filelist.scp -H hmmproto dict hmmlist
```

11.6 Evaluating the frontend

The different training methods were then evaluated at the frame level, after decoding. The decoding gives us new label files, that we will compare with an original pfile. The reason for comparing it with a pfile rather than an original label file, is that the discontinuities have been taken into consideration here with the "oth" label. The information in the pfile is also what the ANN has to work with, and so it is logical to compare the results with that. The original pfile to be compared with is always a single state CI one, because what we really care about is the attribute recognition, not the states that we have used in between, for training the ANN. It might seem counter-intuitive to compare with a pfile that includes "oth" when it is not included in decoding, but to get a real and honest number about the performance, we need to take it into consideration. A few, or many frames may be labeled "oth", and those frames will never be recognized with our setup. They will get some different label. No doubt, this will impact the evaluation negatively. As was mentioned earlier, the first round of training was done without English data. This reduces the dataset quite drastically, and this effect must be taken into account. Less data probably entails a poorer ANN to use for recognition.

11.6.1 Evaluation method

Evaluation is done on frames, and not segments of multiple frames.Each frame has its own label and is 10 ms long. By default, HVite gives scoring information (log likelihoods) in the recognized label file, such as:

```
#!MLF!#
"*/p1_g01_f1_1_t-a0001.rec"
0 11600000 sil -95.436058
11600000 13000000 sil -17.501068
13000000 17900000 sil -43.445045
17900000 18500000 velar -10.734372
18500000 19400000 mid -12.244259
19400000 19900000 coronal -8.889365
19900000 20400000 mid -5.715060
20400000 21000000 dental -8.824461
21000000 23100000 mid -18.393522
23100000 23600000 dental -5.103466
23600000 23900000 dental -6.842369
23900000 25100000 low -18.461010
25100000 26100000 coronal -20.219107
26100000 26900000 dental -10.475648
26900000 27600000 mid -7.869104
27600000 29400000 coronal -27.718321
```

Above is an example of decoding of Norwegian training data. We see the start frame in the first column, end frame in the second column, attribute in the third column, and scoring information in the fourth column. The state information is suppressed by default, and we don't need it for evaluation either. For evaluation, we want to strip off all unneeded information. This can be done with the argument `-o S`. Below is an example of decoding of Norwegian test data:

```
#!MLF!#
"*/p1_g01_f1_4_x-a0001.rec"
0 3000000 sil
3000000 7900000 sil
7900000 12000000 sil
12000000 13600000 sil
13600000 14500000 sil
14500000 17100000 sil
```

```
17100000 17800000 sil
17800000 19100000 sil
19100000 20000000 sil
20000000 20400000 velar
20400000 20900000 mid
20900000 21200000 coronal
21200000 21600000 mid
21600000 22300000 dental
22300000 23800000 mid
23800000 24500000 dental
```

The measures of interest are True Positives, False Positives, True Negatives and False Negatives. From them, other measures can be calculated. From [9] we get:

$$Accuracy = \frac{TP + TN}{P + N} \tag{11.1}$$

$$Precision = \frac{TP}{TP + FP} \tag{11.2}$$

$$Recall = \frac{TP}{P} \tag{11.3}$$

$$F_{score} = \frac{2}{1/Precision + 1/recall} \tag{11.4}$$

The evaluation script is found in appendix H. It takes in a decoded mlf with the three-column form demonstrated above, and an original pfile to compare with. A dictionary is also supplied. It can be invoked as

```
    perl Eval_Decoding.pl cv.pfile decoded.mlf
```

It removes header and file information from the input mlf until we are left only with a sequence of frames and labels. If it encounters a frame progression bigger than 10 ms, it expands that frames into 10 ms frames, so that each and every start and stop is strictly incremented with $1 * 10^5$. It prints out this new, expanded mlf without header information for control purposes. The pfile labels are also printed. It already progresses strictly with 10 ms at a time, for each frame. For example, an mlf looking like

```
#!MLF!#
"*/GEcall-106-G.rec"
0 500000 sil
500000 1700000 sil
1700000 2100000 high
```

would look like this, after processing by the evaluation script:

```
0 100000 sil
100000 200000 sil
200000 300000 sil
300000 400000 sil
400000 500000 sil
500000 600000 sil
600000 700000 sil
700000 800000 sil
800000 900000 sil
900000 1000000 sil
1000000 1100000 sil
1100000 1200000 sil
```

```
1200000 1300000 sil
1300000 1400000 sil
1400000 1500000 sil
1500000 1600000 sil
1600000 1700000 sil
1700000 1800000 high
1800000 1900000 high
1900000 2000000 high
2000000 2100000 high
```

Then the time information is removed, and we are left with only the labels, with indices taken from the dictionary.

```
8
8
8
8
8
8
8
8
8
8
8
8
8
8
8
8
8
3
3
3
3
```

The last step before calculating scores, is to expand these labels into binary vectors, as though we were using a detector bank, rather than a multiclass recognizer. Only the first frame is shown below.

```
0
0
0
0
0
0
0
0
1
0
```

The same process is done for the pfile information. Now we can compare the two lists with the equations 11.1, 11.2, 11.3 and 11.4. The scores are calculated for each attribute separately, and then a total score.

All 34 networks were tested in this way. F-score is the harmonic mean of recall and precision, and gives us the best single number to compare networks with. Other results are found in the appendix. Numbers for a single state is first shown in a table, then a plot of the Fscore of the realignment development is shown for 3-state. Finally, a table for the best iteration is provided for easy comparison with single state networks. Let's start with the results from the RC networks.

11.7 RC trained without English

In this section we will see how RC networks perform, in 1-state and 3-state configurations. First, let's look at OGI's development set, which like the training set, does not include English. Then we will compare the results with TIMIT and the Norwegian NAFTA database.

11.7.1 OGI CV set

For RC, one-state is shown in the table below.

Table 11.9: OGI scores for RC 1-state

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	90.5	52.4	54.2	0.53
Dental	92.0	51.8	55.7	0.54
Glottal	99.4	58.1	54.1	0.56
High	94.0	61.5	61.2	0.61
Labial	95.8	61.6	64.9	0.63
Low	93.0	68.1	66.8	0.67
Mid	88.9	59.3	72.9	0.65
Palatal	98.2	58.7	50.5	0.54
Sil	91.0	84.1	88.1	0.86
Velar	97.1	68.0	55.3	0.61
Oth	nan	nan	nan	nan
Total	94.1	67.6	67.6	0.68

Now, let's look at the development in RC 3-state alignment F-score.
Looking at the total precision reveals that the best network is the unaligned network, which is unexpected and disappointing. Nevertheless, as expected, the tristate network outperforms the single state one.

Table 11.10: OGI scores for RC 3-state 0 alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	91.5	56.9	60.6	0.59
Dental	92.5	55.1	57.1	0.56
Glottal	99.4	62.9	49.6	0.55
High	94.2	63.4	61.1	0.62
Labial	95.9	63.1	64.5	0.64
Low	93.3	68.9	69.6	0.69
Mid	89.3	59.9	76.8	0.67
Palatal	98.2	57.9	51.6	0.55
Sil	91.2	84.7	87.7	0.86
Velar	97.3	72.7	55.9	0.63
Oth	nan	nan	nan	nan
Total	94.4	69.1	69.1	0.69

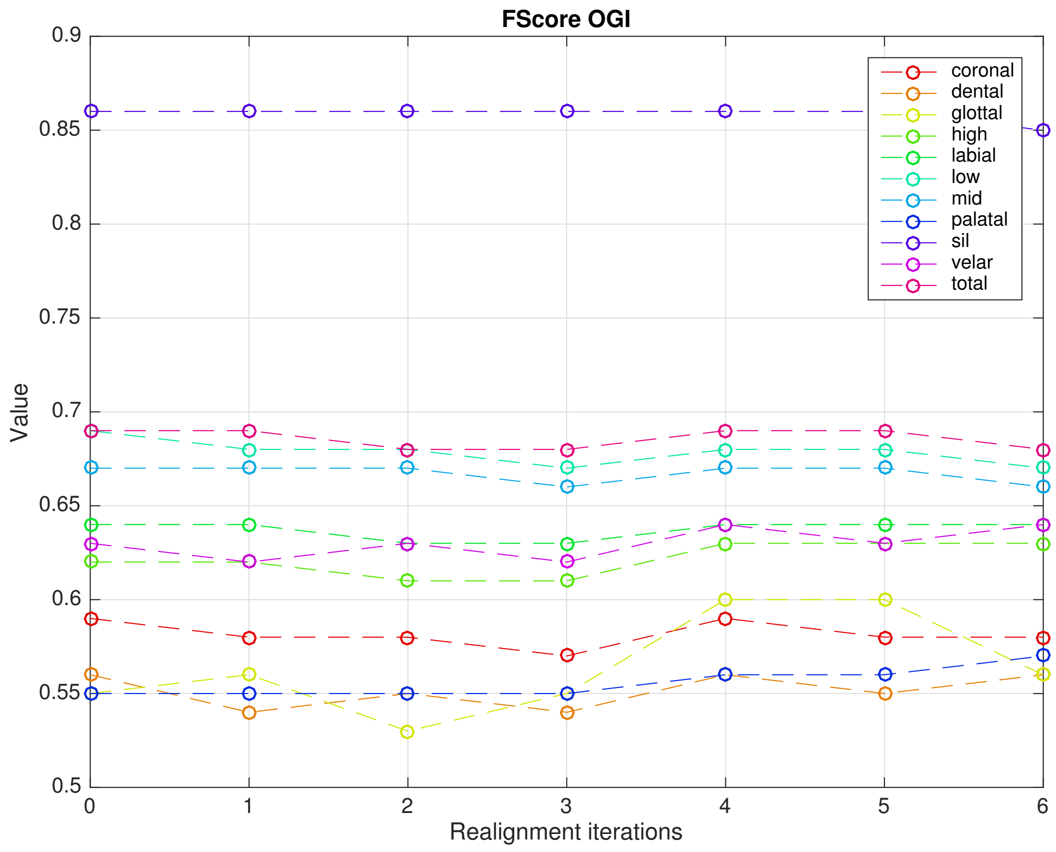


Figure 11.3: RC OGI FScore realignment

11.7.2 TIMIT test set

We see that the TIMIT scores reaffirm what we learned in last subsection. The unaligned RC network is better than all the realigned ones.

Table 11.11: TIMIT scores for RC 1-state

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	77.2	65.1	42.3	0.51
Dental	96.6	3.9	7.5	0.05
Glottal	99.2	82.9	15.7	0.26
High	88.9	57.7	32.4	0.42
Labial	91.2	58.5	34.3	0.43
Low	88.2	58.7	45.4	0.51
Mid	80.0	22.3	45.1	0.30
Palatal	97.0	60.2	56.6	0.58
Sil	83.7	45.4	92.0	0.61
Velar	95.2	56.6	58.4	0.58
Oth	nan	nan	nan	nan
Total	90.5	47.9	47.9	0.48

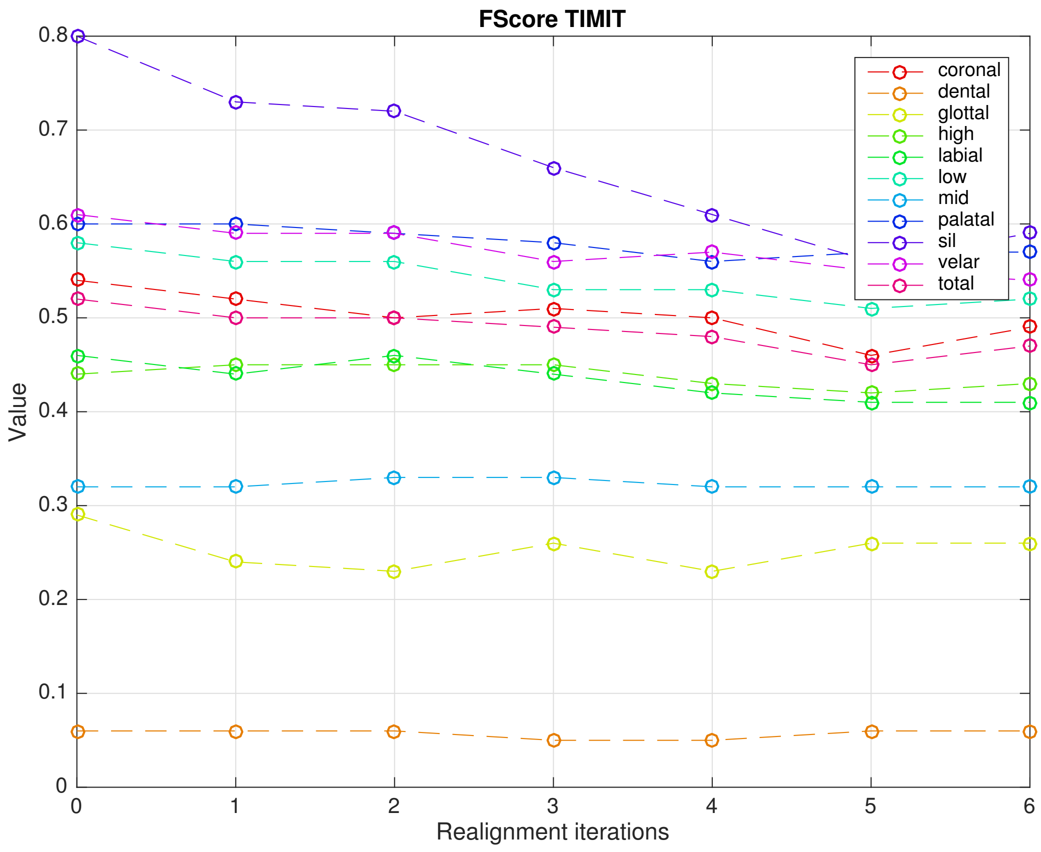


Figure 11.4: RC TIMIT FScore realignment

Table 11.12: TIMIT scores for RC 3-state 0 alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	77.6	64.7	46.2	0.54
Dental	96.7	4.6	8.5	0.06
Glottal	99.2	78.4	17.6	0.29
High	88.6	54.5	37.2	0.44
Labial	91.3	58.0	38.1	0.46
Low	89.1	60.7	55.8	0.58
Mid	77.4	22.4	57.0	0.32
Palatal	97.2	64.2	56.0	0.60
Sil	93.8	72.3	88.8	0.80
Velar	95.5	58.0	63.7	0.61
Oth	nan	nan	nan	nan
Total	91.3	52.3	52.3	0.52

11.7.3 NAFTA test set

Table 11.13: Norwegian scores for RC 1-state

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	88.7	26.1	49.8	0.34
Dental	83.6	41.4	18.9	0.26
Glottal	99.2	58.7	20.3	0.30
High	91.7	55.0	38.3	0.45
Labial	93.6	61.1	28.1	0.39
Low	94.1	60.6	46.6	0.53
Mid	85.9	42.4	72.3	0.53
Palatal	97.7	46.5	48.4	0.47
Sil	94.6	89.6	96.8	0.93
Velar	94.9	46.1	53.1	0.49
Oth	nan	nan	nan	nan
Total	93.1	62.1	62.1	0.62

Once again, the results from before are reaffirmed. The unaligned RC network performs better than all the realigned ones. Notice though, that the Norwegian results are far better than for TIMIT. Notice also that the tristate precision is only marginally higher than the single state precision.

11.7.4 Summary

We clearly see from all the test and CV sets that tristate RC systems always performs better than single state counterparts. The maximum OGI total precision at 69.1 % is far higher than for TIMIT at 52.3 % and for NAFTA at 62.9 %. Remember though, that the CV set isn't directly comparable to the test sets of TIMIT and NAFTA. It is primarily used as a indicator for when realignment no longer bears fruit.

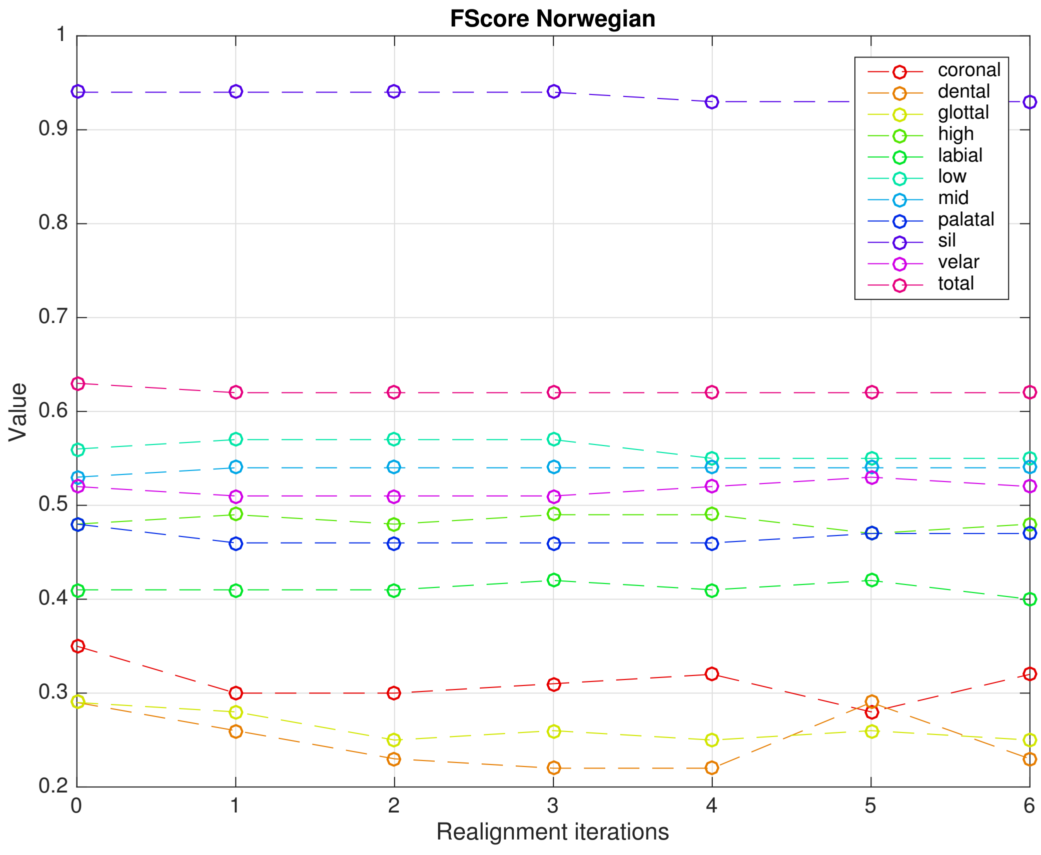


Figure 11.5: RC Norwegian FScore realignment

Table 11.14: Norwegian scores for RC 3-state 0 alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	88.7	26.4	50.9	0.35
Dental	83.2	40.7	22.2	0.29
Glottal	99.2	60.6	18.7	0.29
High	91.7	55.1	42.0	0.48
Labial	93.7	60.6	30.6	0.41
Low	94.2	60.3	51.9	0.56
Mid	86.1	42.7	71.4	0.53
Palatal	98.0	55.0	42.9	0.48
Sil	95.4	92.0	95.9	0.94
Velar	95.5	51.7	51.9	0.52
Oth	nan	nan	nan	nan
Total	93.3	62.9	62.9	0.63

11.8 CI trained without English

In this section, the same is done as for RC systems trained without English, only with CI attributes.

11.8.1 OGI CV set

Table 11.15: OGI scores for CI 1-state alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	90.5	52.4	51.1	0.52
Dental	91.8	50.8	49.3	0.50
Glottal	99.4	60.3	53.6	0.57
High	93.9	61.0	59.6	0.60
Labial	95.9	63.6	61.3	0.62
Low	92.9	66.4	69.1	0.68
Mid	89.1	59.7	73.4	0.66
Palatal	98.0	54.3	50.4	0.52
Sil	91.0	82.9	90.0	0.86
Velar	97.0	65.5	55.8	0.60
Oth	nan	nan	nan	nan
Total	94.1	67.3	67.3	0.67

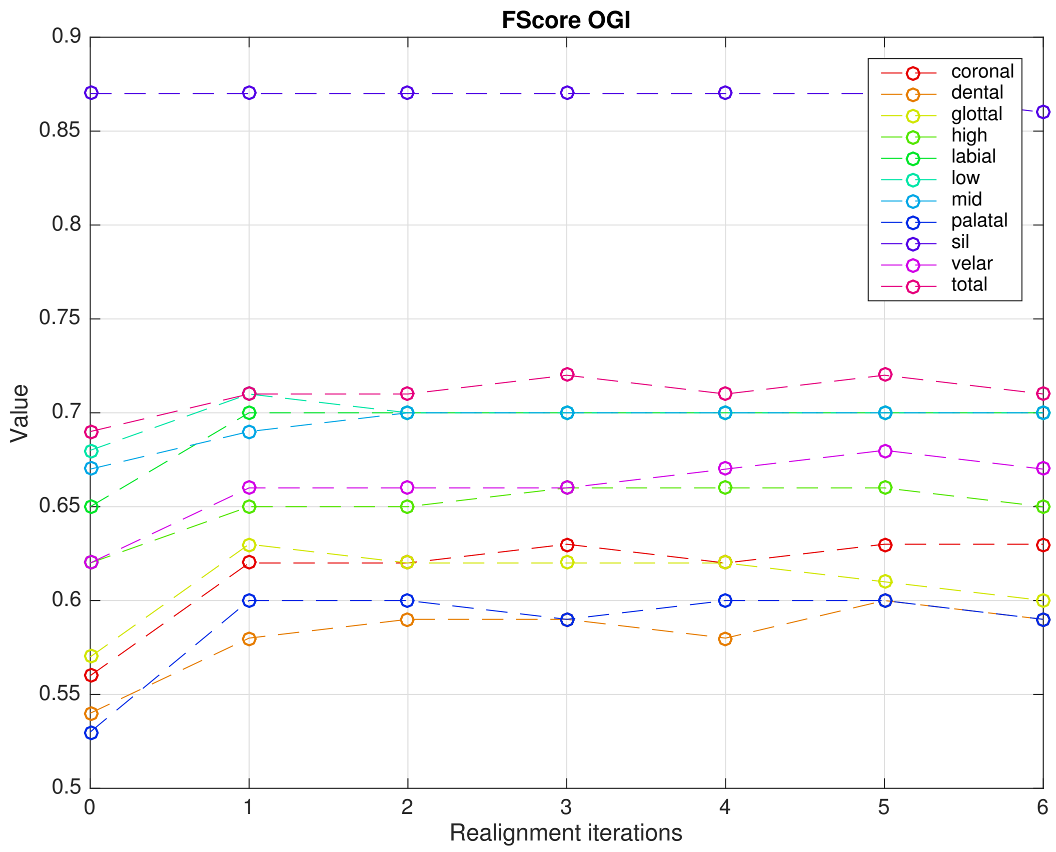


Figure 11.6: CI OGI FScore realignment

The third alignment gives the best total precision here.

Table 11.16: OGI scores for CI 3-state 3 alignment

Articulation	**Accuracy %**	**Precision %**	**Recall %**	**F-Score**
Coronal	92.6	63.2	62.2	0.63
Dental	93.5	62.6	55.8	0.59
Glottal	99.5	69.8	55.1	0.62
High	94.6	66.0	65.2	0.66
Labial	96.8	73.7	66.8	0.70
Low	93.3	67.8	73.4	0.70
Mid	90.4	63.3	77.9	0.70
Palatal	98.4	65.5	53.6	0.59
Sil	91.2	82.3	91.6	0.87
Velar	97.5	74.7	59.9	0.66
Oth	nan	nan	nan	nan
Total	94.8	71.6	71.6	0.72

11.8.2 TIMIT test set

Table 11.17: TIMIT scores for CI 1-state alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	77.0	65.4	39.7	0.49
Dental	95.7	4.5	12.4	0.07
Glottal	99.2	75.1	20.5	0.32
High	89.1	58.2	34.5	0.43
Labial	91.7	62.1	37.6	0.47
Low	88.2	56.7	54.9	0.56
Mid	77.2	21.9	55.2	0.31
Palatal	97.0	61.0	50.3	0.55
Sil	90.9	61.3	92.4	0.74
Velar	95.3	57.4	55.5	0.56
Oth	nan	nan	nan	nan
Total	90.9	49.7	49.7	0.50

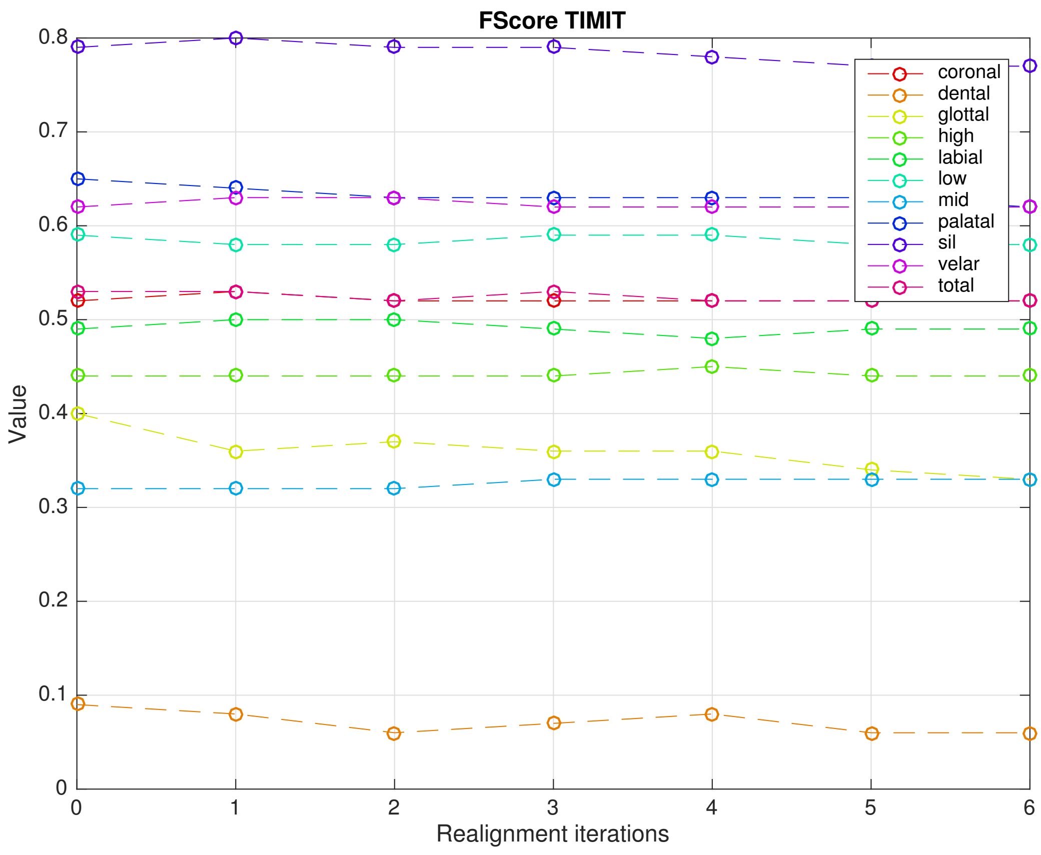


Figure 11.7: CI TIMIT FScore realignment

For TIMIT, the first alignment gives the best result. The improvement is small compared to the OGI CV set, though.

Table 11.18: TIMIT scores for CI 3-state 1 alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	78.4	69.5	42.6	0.53
Dental	96.1	6.0	14.4	0.08
Glottal	99.2	81.3	22.9	0.36
High	88.9	56.9	35.5	0.44
Labial	92.2	66.7	40.2	0.50
Low	88.3	56.8	60.3	0.58
Mid	76.6	21.8	57.5	0.32
Palatal	97.5	67.4	60.7	0.64
Sil	93.5	69.7	93.7	0.80
Velar	96.2	67.2	60.0	0.63
Oth	nan	nan	nan	nan
Total	91.4	52.7	52.7	0.53

11.8.3 NAFTA test set

Table 11.19: Norwegian scores for CI 1-state alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	87.8	25.4	55.0	0.35
Dental	83.6	41.9	20.6	0.28
Glottal	99.2	67.8	21.7	0.33
High	91.7	55.0	38.8	0.46
Labial	93.7	60.8	30.3	0.40
Low	94.0	57.9	51.8	0.55
Mid	86.4	43.3	71.5	0.54
Palatal	98.1	56.0	45.6	0.50
Sil	96.3	92.6	97.9	0.95
Velar	95.5	52.1	45.3	0.48
Oth	nan	nan	nan	nan
Total	93.3	63.1	63.1	0.63

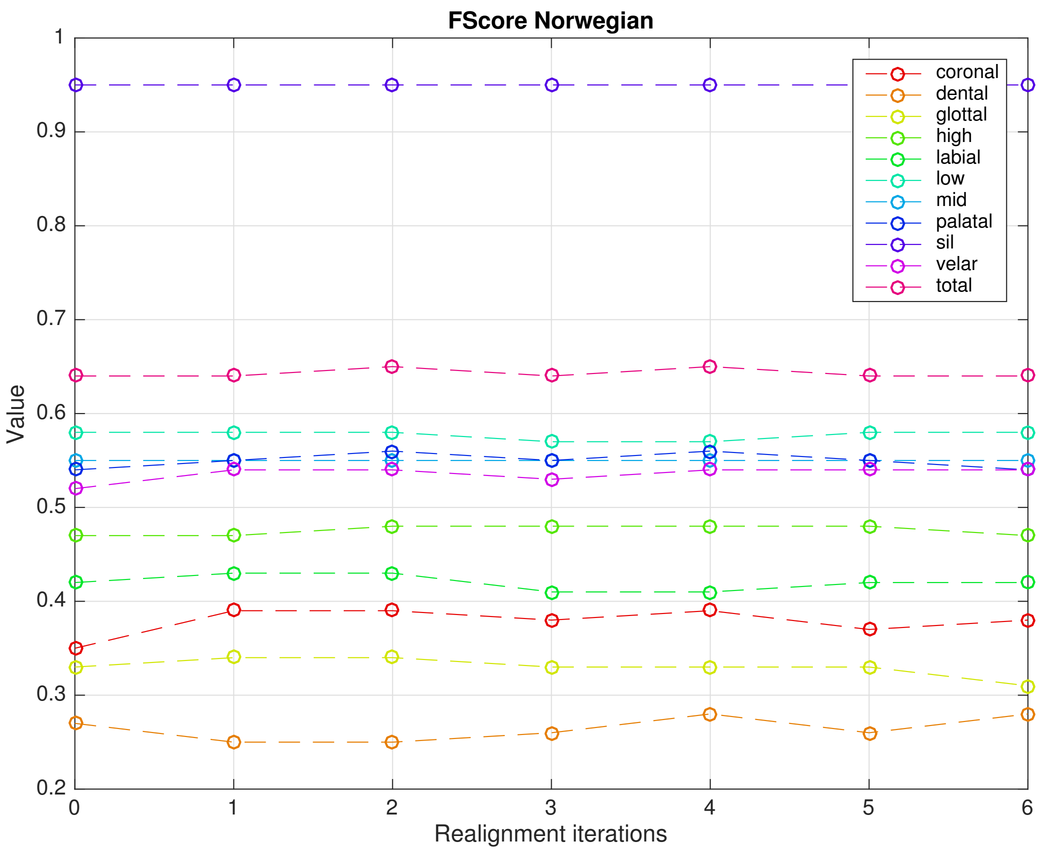


Figure 11.8: CI Norwegian FScore realignment

For Norwegian, the fourth alignment gives the best results. The difference between NAFTA and TIMIT is stark.

Table 11.20: Norwegian scores for CI 3-state 4 alignment

Articulation	**Accuracy %**	**Precision %**	**Recall %**	**F-Score**
Coronal	88.5	28.2	61.4	0.39
Dental	84.3	46.0	20.3	0.28
Glottal	99.3	72.7	21.0	0.33
High	92.0	58.0	40.5	0.48
Labial	94.2	71.7	29.0	0.41
Low	94.2	59.2	55.6	0.57
Mid	86.5	43.8	73.1	0.55
Palatal	98.2	60.4	51.6	0.56
Sil	95.9	90.9	98.9	0.95
Velar	96.1	60.2	48.9	0.54
Oth	nan	nan	nan	nan
Total	93.6	64.6	64.6	0.65

11.8.4 Summary

Again, as with RC, it is clear that tristate CI systems perform better than single state counterparts. For OGI, the best performing system has a maximum total precision at 71.6 %, which is an improvement of 2.5 % compared to the highest RC precision. TIMIT's maximum total precision lies at 52.7%, giving an improvement of 0.4% compared to RC. NAFTA's precision lies at 64.6%, with an improvement of 1.7 % compared to RC.

11.9 CI trained with English

Since the CI network proved most promising, it was chosen as the network to use going forward. A last test is done in which we see the OGI CV improvement over 3-state CI realignment realignments, as well as performance on NAFTA.

11.9.1 OGI CV set

In this section, the same is done as for RC systems trained without English, only with CI attributes with all languages in the training set. The third alignment gives the best results here. Notice

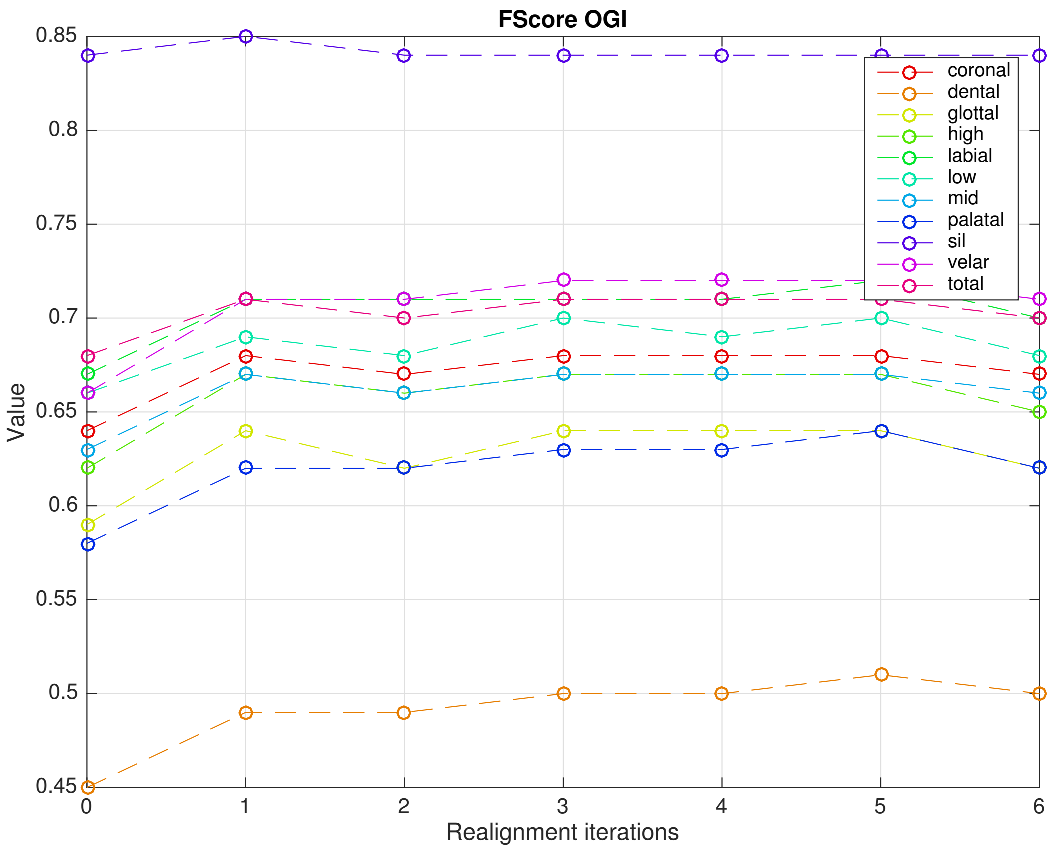


Figure 11.9: CI OGI FScore realignment

the small decrease in precision when including all training languages, and testing on the cv set with all languages, compared with the last section where English was not included.

Table 11.21: OGI scores for CI 3-state 3 alignment

Articulation	**Accuracy %**	**Precision %**	**Recall %**	**F-Score**
Coronal	89.7	63.7	72.1	0.68
Dental	95.3	64.0	40.8	0.50
Glottal	99.5	76.9	54.9	0.64
High	94.5	67.4	67.2	0.67
Labial	96.0	73.2	69.8	0.71
Low	93.3	66.4	73.1	0.70
Mid	90.3	62.7	72.1	0.67
Palatal	98.7	68.6	58.8	0.63
Sil	90.7	81.0	88.0	0.84
Velar	97.8	77.7	67.7	0.72
Oth	nan	nan	nan	nan
Total	94.7	70.9	70.9	0.71

11.9.2 NAFTA test set

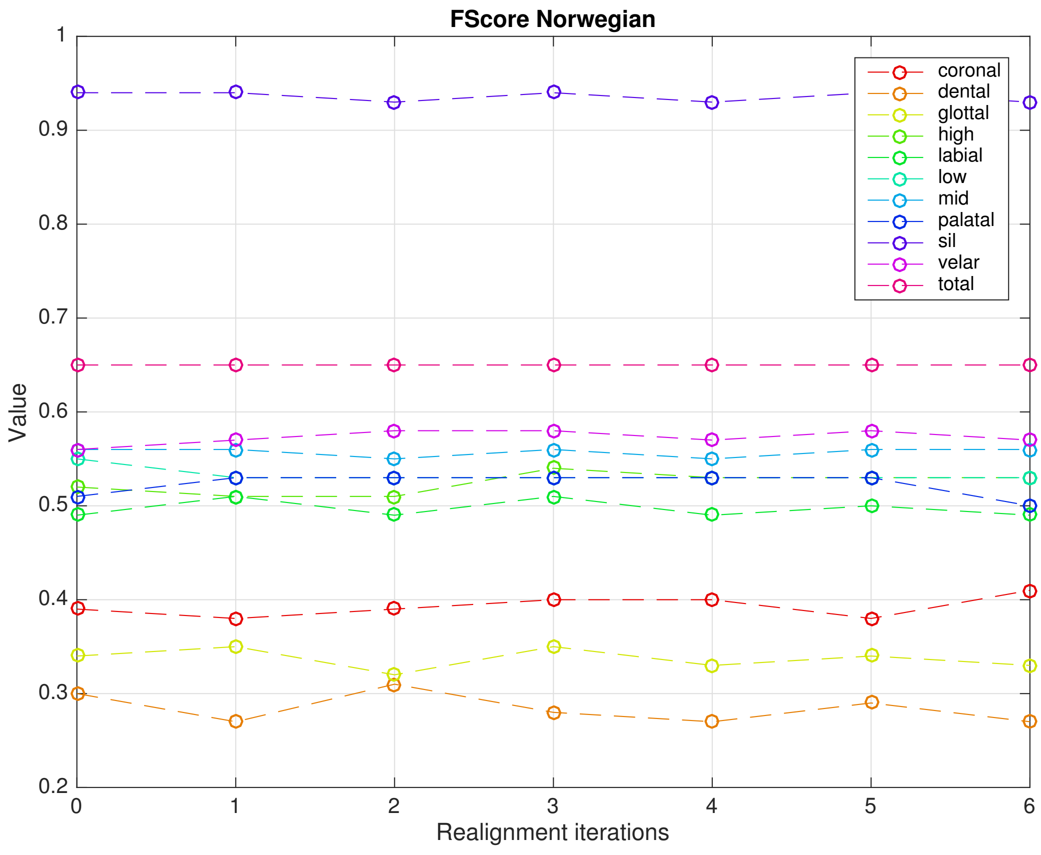


Figure 11.10: CI Norwegian FScore realignment

As for the OGI CV set, the third alignment gives best performance for Norwegian.

Table 11.22: Norwegian scores for CI 3-state 3 alignment

Articulation	Accuracy %	Precision %	Recall %	F-Score
Coronal	88.7	29.0	63.5	0.40
Dental	84.8	50.3	19.0	0.28
Glottal	99.3	82.6	21.8	0.35
High	92.1	56.6	50.9	0.54
Labial	94.5	69.2	39.8	0.51
Low	94.3	63.3	46.2	0.53
Mid	87.2	45.5	72.7	0.56
Palatal	98.4	69.5	42.8	0.53
Sil	95.1	89.1	98.7	0.94
Velar	96.5	65.2	51.6	0.58
Oth	nan	nan	nan	nan
Total	93.7	65.4	65.4	0.65

11.9.3 Summary

Notice a slight decrease in performance on the OGI CV set compared to the network not trained on English: with a total precision at 70.9%, we see a drop of 0.7%. The Norwegian total precision improves, however: $65.4\,\%$, giving an increase of $0.8\,\%$. The third CI alignment proves to be the model to use, going forward. All decoded attributes will henceforth be recognized through this network, and decoded with the tristate CI framework.

11.10 Discussion

In retrospect, the TIMIT and Norwegian test sets should have been partitioned to be roughly equal size. The Norwegian test set is over twice the size of TIMIT's. This partition requires some work so that each speaker, region and gender is represented equally. Still, it is doubtful that the results for RC would have been dramatically different, as all numbers point to it being inferior to CI.

Chapter 12

Experiments: Backend

12.1 Datasets

The datasets used in the backend are different from the ones used in the frontend. The difference lies mainly in that here, we use spontaneous speech, as opposed to speech read from planned out sentences. All training languages except for Norwegian are taken from the Callfriend database. The test utterances (apart from Norwegian) are from the NIST LID 2003 evaluation plan. Norwegian training and test data are from part 3 of the same database, NB tale/NAFTA. All data

Table 12.1: Durations in hours for training sets

Arabic	8.55
English	10.02
Farsi	9.13
French	9.75
German	9.14
Hindi	9.25
Japanese	9.51
Korean	8.22
Mandarin	8.28
Spanish	9.52
Tamil	7.47
Vietnamese	10.70
Norwegian	4.57
Total	114.11

in the backend are run through a VAD tool (provided by Marco Siniscalchi), that removes silence from the waveforms. The point of this is that silence skews the probability distribution of the attributes. In spontaneous speech recordings, silence is plentiful. Frames of silence are non-informative, and should be removed.

The VAD tool divides each file into a folder of chunks, where each folder do not exceed ≈ 30 seconds worth of silence-free speech. The folder naming scheme for Callfriend English training data is, for example:

```
en_4175_A_chunk
en_4175_B_chunk
```

where `A` and `B` refer to two speakers in a telephone conversation. For Norwegian, the database is originally sorted in 24 dialect groups, with 12 of them being with foreign accents. These are

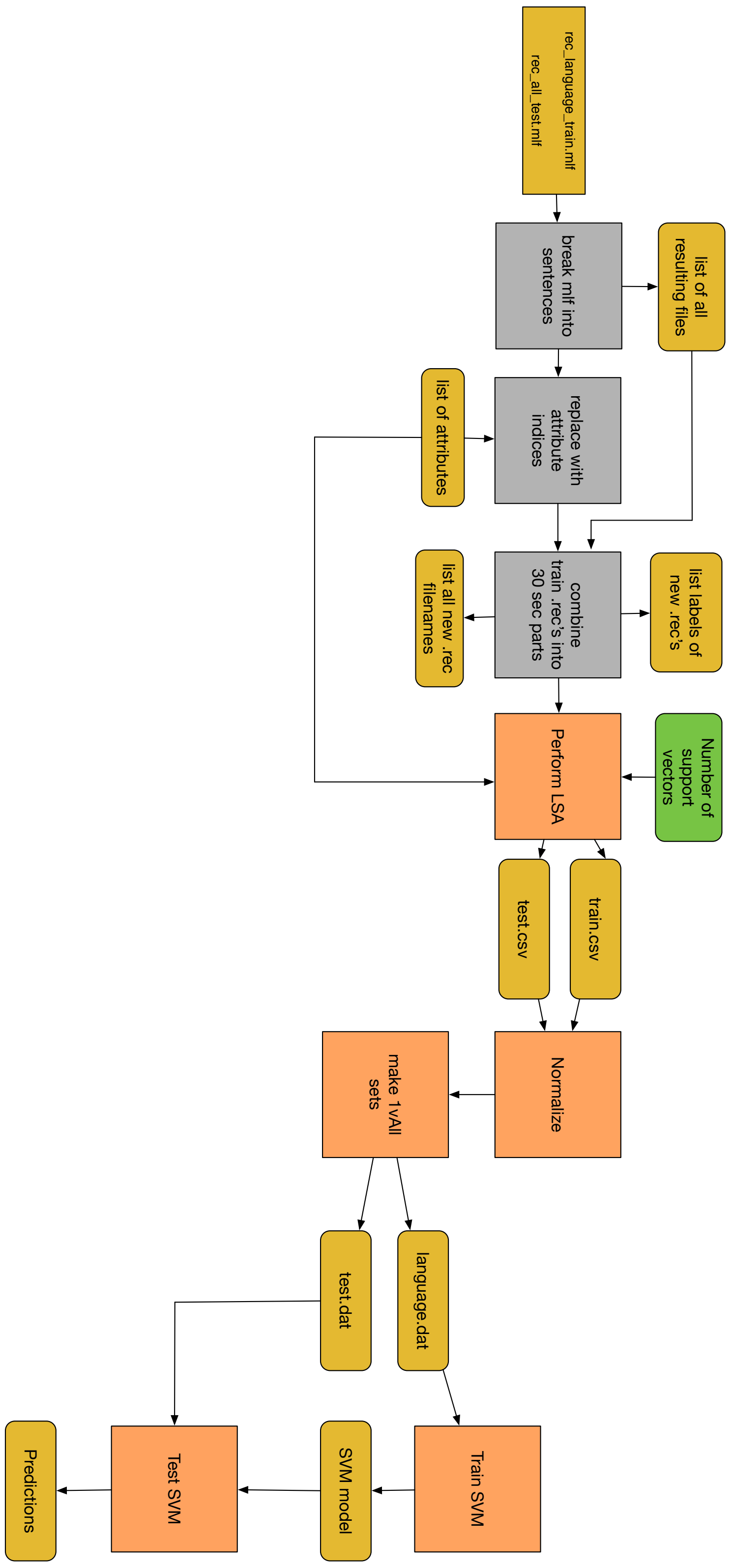


Figure 12.1: Backend flowchart

Table 12.2: Durations in hours for silence-removed test sets

Arabic	0.64
English	1.93
Farsi	0.63
French	0.64
German	0.65
Hindi	0.64
Japanese	1.28
Korean	0.64
Mandarin	0.64
Russian	0.64
Spanish	0.63
Tamil	0.63
Vietnamese	0.63
Norwegian	1.85
Total	12.07

excluded in this work, leaving us with 12 dialect groups. These are all resampled, run through VAD and combined into a single folder. Then a script, written by the author, sorts the NAFTA testing data into per-speaker folders of approximately 30 seconds each. There is some wiggle room here. The tolerance is set to 25-36 seconds, and some folders include a single file from a different speaker. Due to project time constraints, this simple scheme was chosen. Lastly, NAFTA testing data folders are renamed according to the LID naming scheme in Siniscalchi's setup (Russian data was removed, since it is intended to be a "surprise language", beyond the scope of this thesis). They have names such as `lid00001_chunk`. In all, there are 1443 folders of testing data, when Norwegian is included, whose total time roughly adds up to the numbers in table 12.2 (which are exact numbers). For each testing file, there is a label provided in a file called `rec_lid03e1_30.lab`.

12.2 Procedure

12.2.1 Preprocessing

Before LSA can be performed, some text manipulation must be done in advance. Siniscalchi's scripts do most of this. Siniscalchi's top level script is called `backendNIST.sh`, and is found in the appendix. `breakNIST.pl` breaks master label files into individual `.rec` files. Then, the attribute identities are replaced with a number from a dictionary indice, with `replacePhns.pl`. For use in training, the time markers in the `.rec`'s are used to divide them further, into files representing approximately 30 seconds worth of attribute labels, with the script `combineCALL.m`. Variants of these scripts were modified by the author to fit with the NAFTA training set naming scheme. A list of all these 30 second `.rec`'s for training and testing data are input to `quadlsaNIST.m`, which performs the LSA.

12.2.2 LSA

`quadlsaNIST.m` is implemented in accordance with chapter 7. Despite its name, it performs up to trigram LSA. Its inputs are: list of training files, list of testing files, number of phones/attributes and number of support vectors.

Since we have 10 attributes including "sil", the n-gram count is $10 + 100 + 1000 = 1110$. After combining training files into 30 seconds each, into 12525 we get a count vector C of dimension 1110×12525. Notice that this doesn't completely add up to the total number of hours in the table **??**, but this is because several of the `.rec` files slightly surpass the 30 second mark. It is clear, upon inspection, that C is quite sparse. Many of the bigrams, and especially the trigrams, never occur. This affirms the utility of the SVD, which we will soon use.

The script computes weighted count values, or term-document matrix W with dimensions 1110×12525 with eq. 7.3, and subsequently uses it with the prescribed number of support vectors in the command

```
[U,S,V] = svds(W,numSV);
```

Which is the time-consuming part. The number of support vectors are here equal to the singular values used in SVD. Thus we get matrices U with 1110×300, a diagonal matrix S with 300×300 and V with 12525×300. We keep the largest singular values and compute

$$B = S\,V^T \tag{12.1}$$

Giving us a down-projected "concept space" matrix B with 300×12525, which is written (transposed) to `train.csv` for use with the SVM. The same process is done for testing files, producing a new term weighted W, except we use the left singular matrix U to obtain a *pseudo document vector* in the reduced rank space (here written as in Siniscalchi's code):

$$Q = U^T\,W \tag{12.2}$$

Before being written to `test.csv`, Q is also transposed (this corresponds to the equations in chapter 7).

`test.csv` and `train.csv` are normalized for use in the SVM. The script mkSVM1vALL.pl reformats the 300 dimensional scores into `.dat` files, giving them a positive or negative label:

```
+1 1:0.6825 2:0.23966 3:-0.038572 4:0.087513 5:0.077157 6:-0.055949 ...
```

where the number of singular values following the label is 300 (equal to number of support vectors). Then the command in section 10.6.1 is invoked with the right parameters, and with term weighting 3. This choice might seem arbitrary. It had been iteratively determined by Marco Siniscalchi, and project time constraints made it impossible to experiment with both this and the number of support vectors. Thus, all experiments were done with these parameters.

Each language is then trained and tested (classified) as in section 10.6.2 on its own one-versus-all SVM, and output distances is kept for each language in a separate file with the `.pred` extension. We are now ready to use GMMs to get probability distributions for the target and anti-target classes for each language. This is implemented in Siniscalchi's `marcoTestGMM.m`, where the EM algorithm is implemented through other scripts, and the LLR (log-likelihood ratio test) is done.

Before we go there, a preliminary experiment should be mentioned. Part 1 of the NAFTA database was used initially, in the backend. It has a lot of overlap between sentences and speakers in training and testing, and the effect of this became apparent after running the training and testing scripts. SVMlight outputs some statistics after classification, that can be used as indicators before doing the subsequent GMM analysis. Norwegian scored:

```
Accuracy on test set: 99.32% (1602 correct, 11 incorrect, 1613 total)
Precision/recall on test set: 98.49%/98.20%
```

while, in contrast, Vietnamese scored

```
Accuracy on test set: 95.10% (1534 correct, 79 incorrect, 1613 total)
Precision/recall on test set: 50.43%/73.75%
```

Across the board, Norwegian scored *much* better than all the other languages. This is an obvious red flag for overtraining/overfitting. Near 100% precision with the flawed frontend is unrealistic. Even after removing most of the overlap (the a-marked sentences were removed), this outperformance persisted. This is why part 1 of NAFTA was exchanged for part 3, which is far more comparable to Callfriend (both are spontaneous speech, removing almost all overlap between sentences). The following results validate this assumption. Results for Norwegian part 3:

```
Accuracy on test set: 92.31% (1332 correct, 111 incorrect, 1443 total)
Precision/recall on test set: 79.46%/73.25%
```

To compare, here are results for Vietnamese in the new system:

```
Accuracy on test set: 93.97% (1356 correct, 87 incorrect, 1443 total)
Precision/recall on test set: 47.11%/71.25%
```

The number of Norwegian testing files was also increased, so this could explain the slight drop in Vietnamese performance. Notice that Norwegian still outperforms the rest. Part 3 was used from then on.

12.2.3 LLR

We are mainly interested in the results from the final LID stage. The verification is done in a null hypothesis fashion, while still using the precision/recall dichotomy from last chapter. The likelihood ratio test (LLR) can be expressed as

$$\Lambda = log\Big(\frac{L(\mathcal{C}_{language}|x)}{L(\mathcal{C}_{allothers}|x)}\Big) > \theta \tag{12.3}$$

Where θ is some threshold over which we accept the target language hypothesis, x is SVM distances and the C's represent some class or classes. The GMMs have the classified training data scores as input to the EM algorithm. With these GMMs, we can with the test data as input, obtain the likelihoods Λ. There is no obvious choice for this θ parameter; it depends on what performance we value. For this thesis, the FScore is used as a metric. It is the harmonic mean of precision and recall, and so a natural goal would be to maximize FScore. θ can be found iteratively. The parameter is to be shifted with small incremental values to maximize FScore. The FN, TN, FP and TP values are found through Siniscalchi's script `marcoTestGMM.m`, where the threshold is an input parameter. A scoring file with extension `.comp` is made for each language, with format:

```
norwegian yes correct
norwegian yes correct
norwegian no wrong
```

A script written by the author (`makescores.pl`) takes these files, converts them to binary (correct/incorrect becomes 1 or 0) and computes accuracy, precision, recall and FScore, as in the frontend chapter. First though, the maximum FScore is found with respect to the *entire* language set. It can be found by simply adding the lines

```
Acc=100*(TP+TN)/(TP+FP+TN+FN);
Prec=100*(TP)/(TP+FP);
Reca=100*(TP)/(TP+FN);
FScore=(2*TP)/((2*TP)+FP+FN);
disp(str)
disp(Acc)
disp(Prec)
disp(Reca)
disp(FScore)
```

to `marcoTestGMM.m` and simply iterating over θ.

12.3 Results

With the $\approx$ maximum FScore at $\theta = 2.4$, the scores for the languages are found in table 12.3.

Table 12.3: Test LID scores

Language	**Accuracy %**	**Precision %**	**Recall %**	**F-Score**
Arabic	95.0	55.3	52.5	0.54
English	89.5	75.6	54.2	0.63
Farsi	93.8	44.3	43.8	0.44
French	95.7	61.8	58.8	0.60
German	92.0	37.9	68.8	0.49
Hindi	94.5	50.0	52.5	0.51
Japanese	94.9	83.1	67.5	0.74
Korean	95.7	59.2	72.5	0.65
Mandarin	94.2	48.0	58.8	0.53
Spanish	93.7	44.8	58.8	0.51
Tamil	93.7	45.5	68.8	0.55
Vietnamese	94.9	53.6	65.0	0.59
Norwegian	92.7	78.0	79.0	0.79
For entire set	93.9	59.6	63.1	0.61

12.4 Discussion

We see from table 12.3 that Norwegian performs best, with FScore at 0.79. Runner up is Japanese at 0.74, then Korean at 0.65. Worst performer is Farsi at 0.44. Of course, Norwegian and Japanese has considerably longer testing sets than the others, so this could be a factor, in that the testing sets possibly become more representative of their respective languages. The Vietnamese and English training sets are the largest. Remember also that the frontend is trained on, among others, English. One would hence expect the OGI languages to perform best. This is not the case (with the exception of Japanese). Overall, the results aren't overwhelmingly good. The precision score for the entire set is 59.6%. This means that on average, a little over half of any random language in the set will be detected correctly. And logically, this means that the frontend needs work, which we already knew from chapter 11.

Chapter 13

Conclusion

In this project, a LRE (language recognition) system was implemented with various tools. It was based on an existing system, largely written by professor Marco Siniscalchi of Kore University of Enna. The aim was to test a working system with Norwegian language, and compare the performance to other languages.

The system is divided into two main parts: the frontend and the backend. The frontend "tokenizes" the audio data into a sequence of articulatory features. This part is called an UAR (universal attribute recogniser). These are then used in a VSM (vector space modelling) approach, where the spoken utterance is considered as a spoken document vector [24]. The "place" category of articulatory features were chosen as a starting point. A binary maximum margin classifier, SVM (support vector machine), separates the spoken documents which belong to a given language, and subsequently a target and anti-target probability distribution is made for each language.

The frontend was trained with six languages, and alternately with context independent (CI) and right context (RC) place attributes. A deep neural network functioned as a discriminative block, after which an HMM did decoding, in a connectionist approach. Both one- and tristate HMM (Hidden Markov Models) were implemented with each of these contexts. In the tristate configuration, Viterbi realignment was done six times. Different configurations were performance-tested at the frame level. The RC configuration proved to be inferior to the CI counterpart. This was unexpected, because a similar approach implemented by Siniscalchi et al. [24] (although tested at the segment level) delivered better results for RC, than CI. What was expected, however, was that the tristate configuration always outperformed the single state configuration. Even at the best performance level (total FScore of 0.65% for Norwegian), it is clear that the frontend is not satisfactory. The frontend clearly needs improvement.

The backend was tested with 13 languages (including Norwegian). Since the frontend was the limiting factor here, we couldn't expect very impressive results. The best performer was Norwegian, with FScore at 0.79. However, the average over the entire test set at 0.61 indicates that the system needs improvement.

Chapter 14

Future work

The first goal of the project was to do LID on Norwegian data. As the title of this thesis suggests, the ultimate goal is to extend it to recognition of Norwegian dialects. This is a natural next step. Some improvements should be made first, though. The backend results, as well as the frontend results, indicate that the frontend is not satisfactory for attribute tokenization. Several mitigating measures can be done here:

- Increase training data, and possibly training languages
- Custom tune the RBM and ANN parameters
- Partition all testing data to have roughly equal size, taking speaker and sentence variation into consideration

As was mentioned, the "oth" attribute shouldn't be modelled with three states, but rather a single one. HTKbook [2] (p. 34) describes a context-free "short pause" model that can be used here. The effect of including "oth" in decoding is unclear, and could also be investigated. Expanding the attribute to tristates must also be avoided, in using the ICSI tools. The "sil" attribute is another potential candidate for single state representation. Another experiment that actually is already prepared for, is finding the effect of including priors in the decoding. The script `prior_pfile.pl` finds frame priors in a pfile, and can be used for this purpose.

The current backend setup for language recognition could then be used with dialects, where each dialect takes the place of a language. Again, care must be taken to avoid overlap between testing and training data.

Lastly, foreign accents in Norwegian could be included in the LID system. iVectors with Finnish data have been used with success for this purpose. Kaldi already has some functionality for iVectors, and the extent of the functionality, as well as required extensions to it, could be investigated.

Bibliography

[1] T. Joachims, Making large-Scale SVM Learning Practical. Advances in Kernel Methods - Support Vector Learning, B. Schölkopf and C. Burges and A. Smola (ed.), MIT-Press, 1999 `http://svmlight.joachims.org`.

[2] The HTK Book v 3.4. Steve Young, Gunnar Evermann, Mark Gales, Thomas Hain, Dan Kershaw, Xunying (Andrew) Liu, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, Phil Woodland. . pages 1–368, June 2016.

[3] Yaser S Abu-Mostafa. *Learning from Data*. 2012.

[4] Jerome R Bellegarda. Latent Semantic Mapping. *IEEE Signal Processing Magazine*, pages 70–80, September 2005.

[5] Ian Goodfellow Yoshua Bengio and Aaron Courville. Deep Learning. 2016.

[6] Y Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, 2009.

[7] Christopher M Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, November 1995.

[8] Christopher M Bishop. *Pattern Recognition and Machine Learning*. Springer Verlag, August 2006.

[9] Tom Fawcett. An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8):861–874, June 2006.

[10] Asja Fischer and Christian Igel. An Introduction to Restricted Boltzmann Machines. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[11] Mark Gales and Steve Young. The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends® in Signal Processing*, 1(3):195–304, 2007.

[12] Geoffrey Hinton. CSC321 - Lecture notes - Introduction to Neural Networks and Machine Learning. 2014.

[13] Geoffrey E Hinton. A Practical Guide to Training Restricted Boltzmann Machines. Springer Berlin Heidelberg, 2012.

[14] Xuedong Huang, Alejandro Acero, and Hsiao-Wuen Hon. *Spoken Language Processing*. A Guide to Theory, Algorithm, and System Development. Prentice Hall, 2001.

[15] Yan Huang, Dong Yu, Chaojun Liu, and Yifan Gong. A Comparative Analytic Study on the Gaussian Mixture and Context Dependent Deep Neural Network Hidden Markov Models. *Proceedings of the 15th Annual Conference of the International Speech Communication Association, INTERSPEECH 2014*, 98052(September), 2014.

[16] Daniel Jurafsky and James H Martin. *Speech and Language Processing*. 2009.

[17] Yann LeCun, Leon Bottou, Genevieve B Orr, and Klaus Robert Müller. Efficient BackProp. *Neural Networks: Tricks of the Trade*, 1524:9–50, 1998.

[18] Yann LeCun, Sumit Chopra, Raia Hadsell, Marc Aurelio Ranzato, and Fu Jie Huang. A Tutorial on Energy-Based Learning. *MIT Press*, pages 1–59, September 2006.

[19] Abdel-rahman Mohamed, George E Dahl, Vincent Vanhoucke, Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Navdeep Jaitly, Andrew Senior, Patrick Nguyen, Tara N Sainath, Tara Sainath, and Brian Kingsbury. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[20] Grégoire Montavon, Genevieve B Orr, and Klaus Robert Müller, editors. *Neural Networks: Tricks of the Trade*, volume 7700 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[21] N Morgan and H Bourlard. Continuous speech recognition. *IEEE Signal Processing Magazine*, 12(3):24–42, May 1995.

[22] Katharina Morik, Peter Brockhausen, and Thorsten Joachims. Combining Statistical Learning with a Knowledge-Based Approach - A Case Study in Intensive Care Monitoring. In *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann Publishers Inc., June 1999.

[23] Eric Roberts. Neural Networks - History `http://cs.stanford.edu/people/eroberts/courses/soco/projects/neural-networks/History/history1.html`.

[24] Sabato Marco Siniscalchi, Jeremy Reed, Torbjørn Svendsen, and Chin-Hui Lee. Universal attribute characterization of spoken languages for automatic spoken language recognition. *Computer Speech & Language*, 27(1):209–227, January 2013.

[25] Mehdi Soufifar. Subspace Modeling of Discrete Features for Language Recognition Mehdi Soufifar. pages 1–163, August 2014.

[26] Øystein Staven. Deep neural networks. Technical report, January 2016.

[27] Theano Development Team. Restricted Boltzmann Machines tutorial. `http://deeplearning.net/tutorial/rbm.html`, 2013.

[28] Karel Vesely and Daniel Povey. Kaldi ASR `http://kaldi-asr.org`.

[29] D Randall Wilson and Tony R Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, December 2003.

[30] Fatemeh Zahedi. An Introduction to Neural Networks and a Comparison with Artificial Intelligence and Expert Systems. *Interfaces*, 21(2):25–38, 1991.

*

Appendix A

Scores

A.1 Trained without english

A.1.1 Norwegian test set

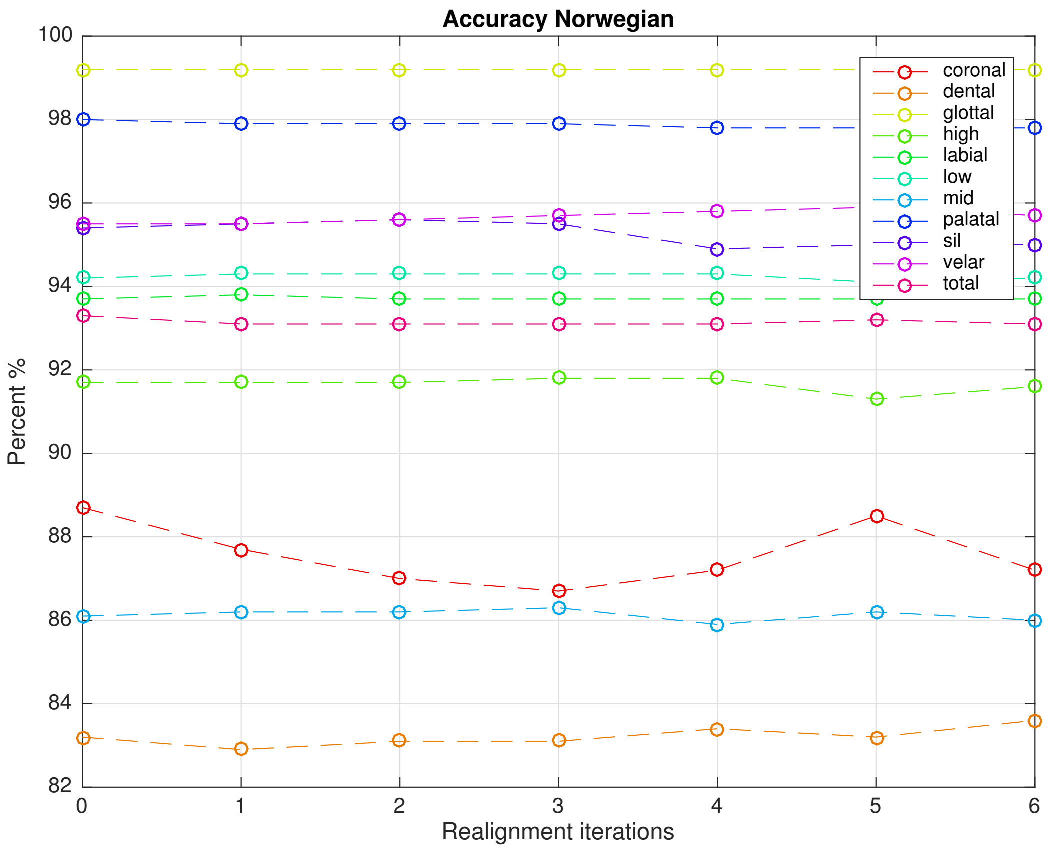


Figure A.1: RC Norwegian Accuracy realignment

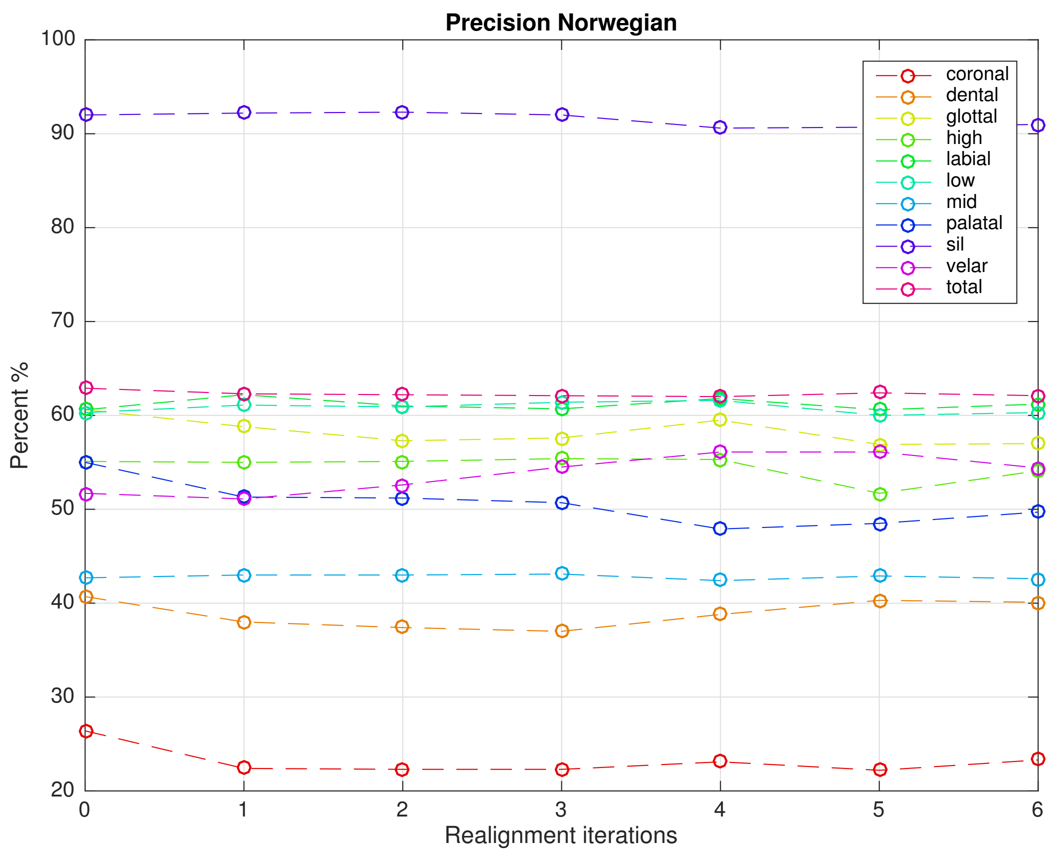


Figure A.2: RC Norwegian Precision realignment

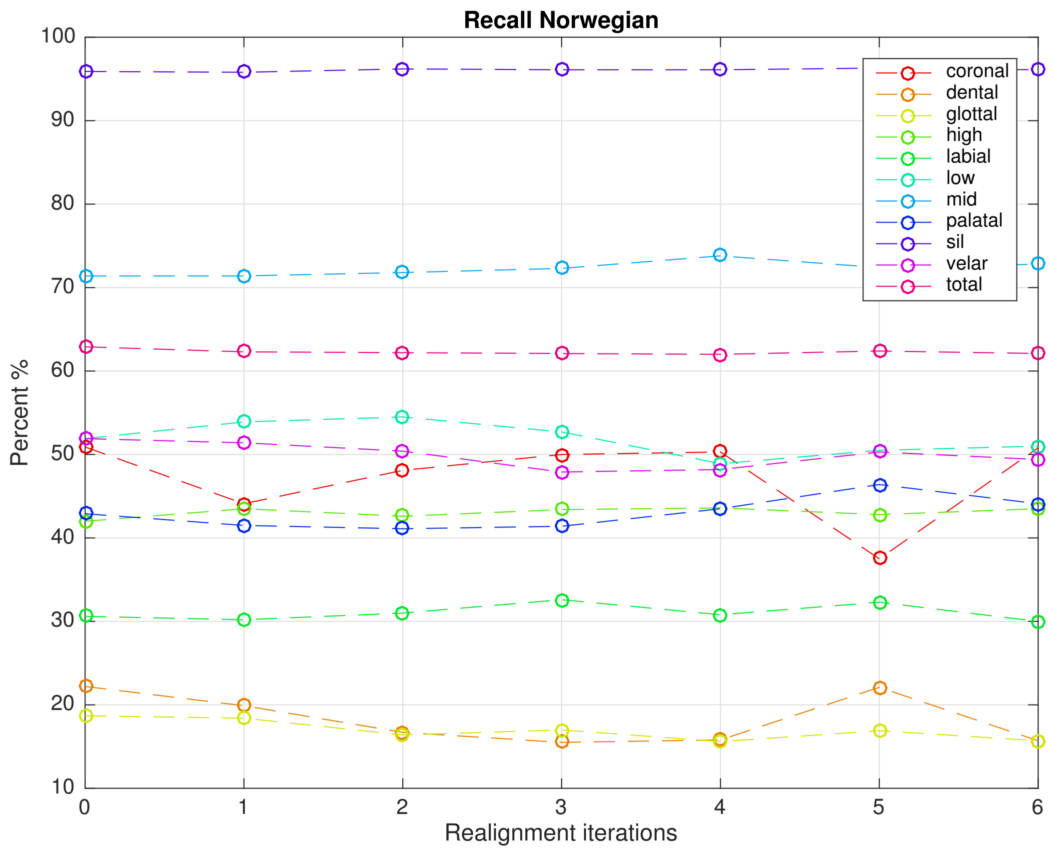


Figure A.3: RC Norwegian Recall realignment

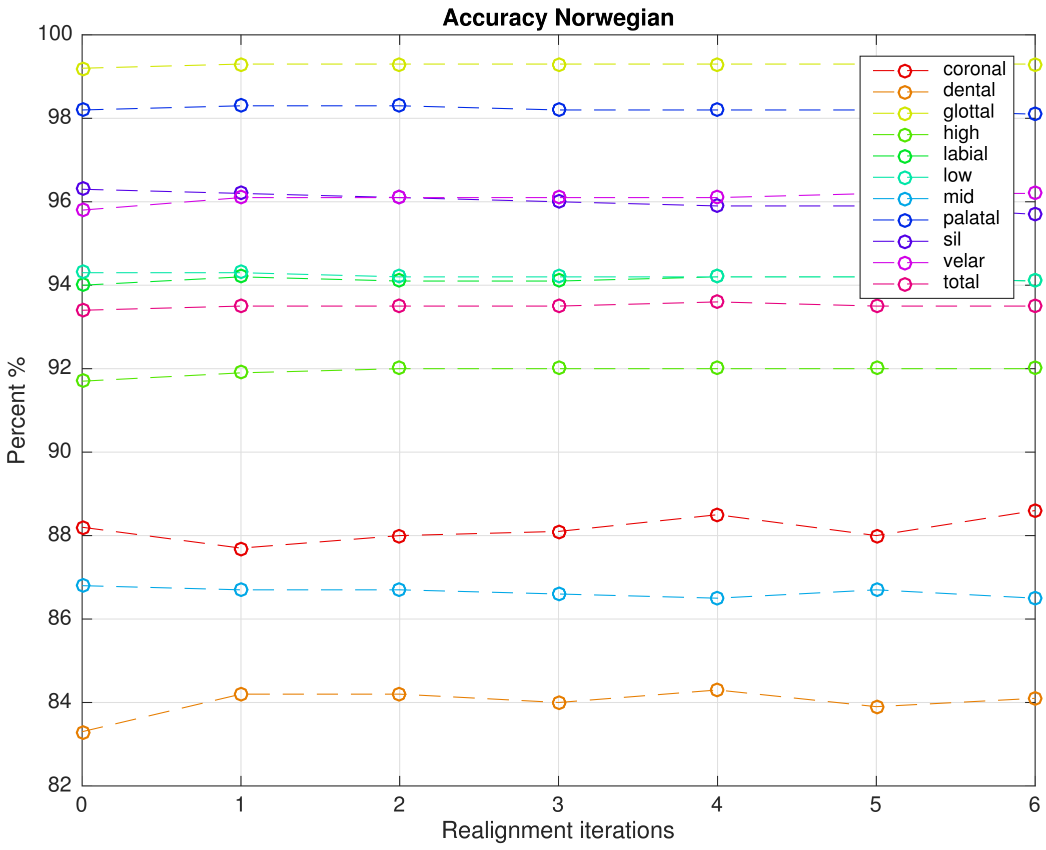


Figure A.4: CI Norwegian Accuracy realignment

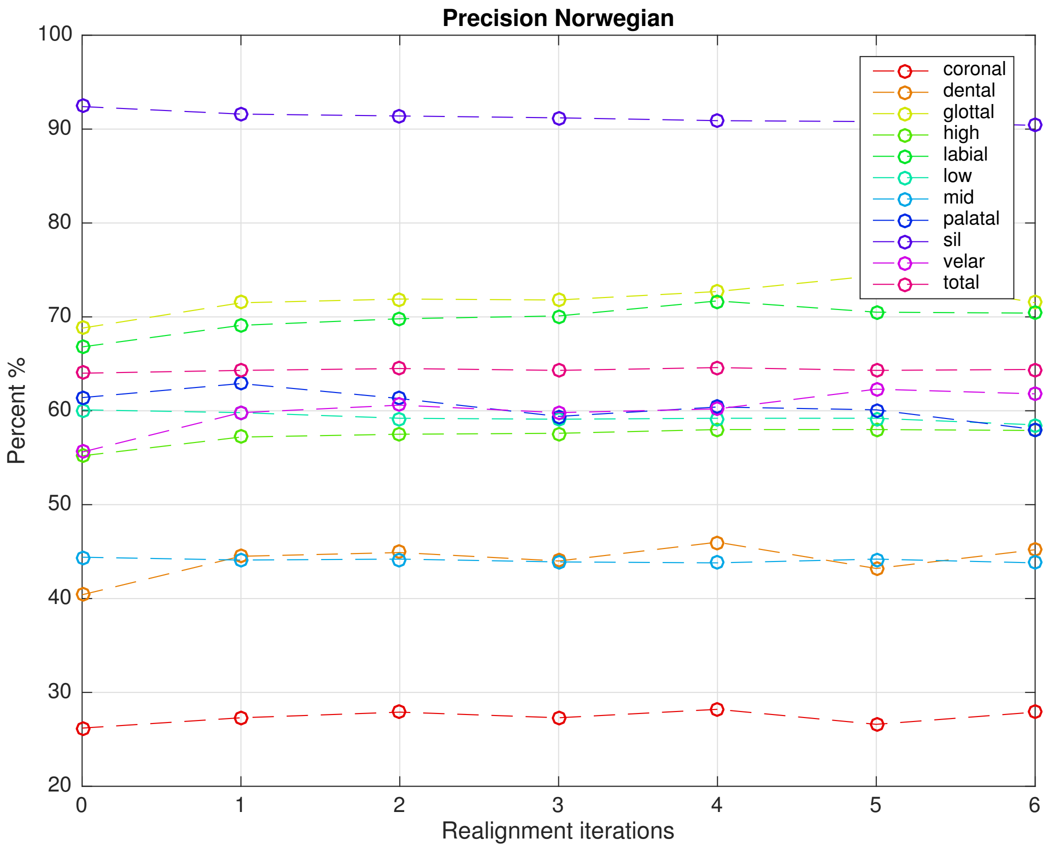


Figure A.5: CI Norwegian Precision realignment

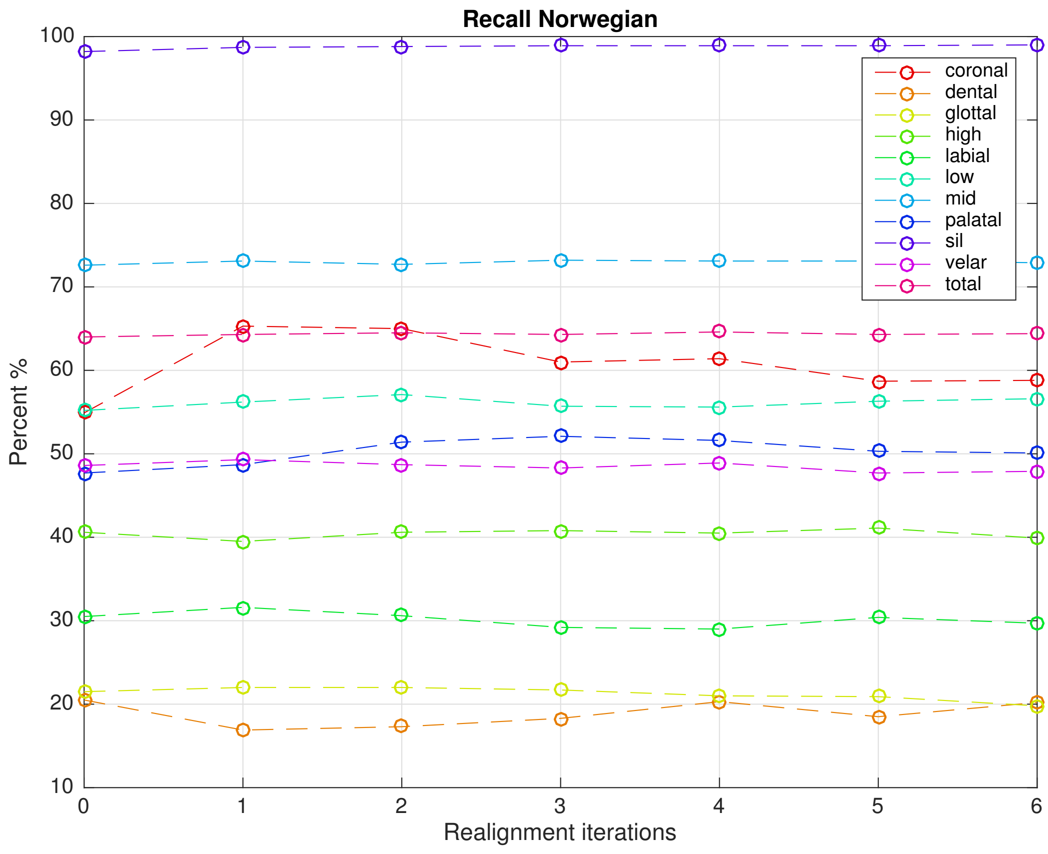


Figure A.6: CI Norwegian Recall realignment

A.1.2 TIMIT test set

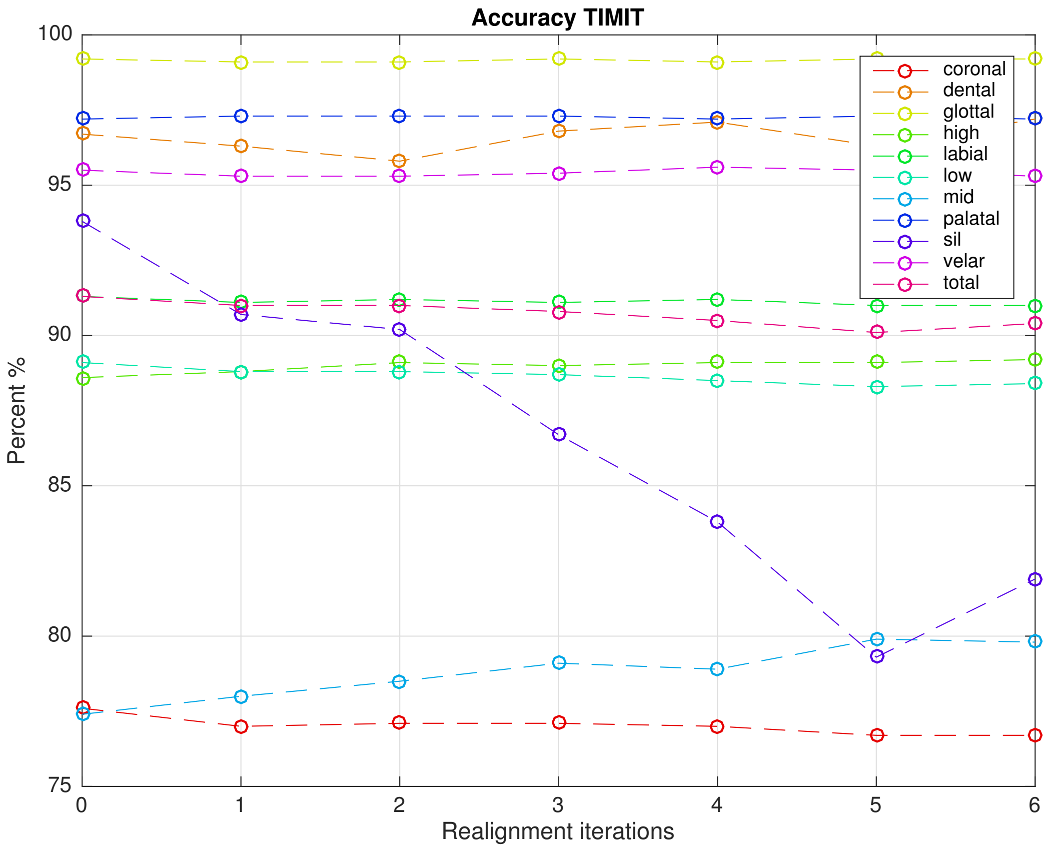


Figure A.7: RC TIMIT Accuracy realignment

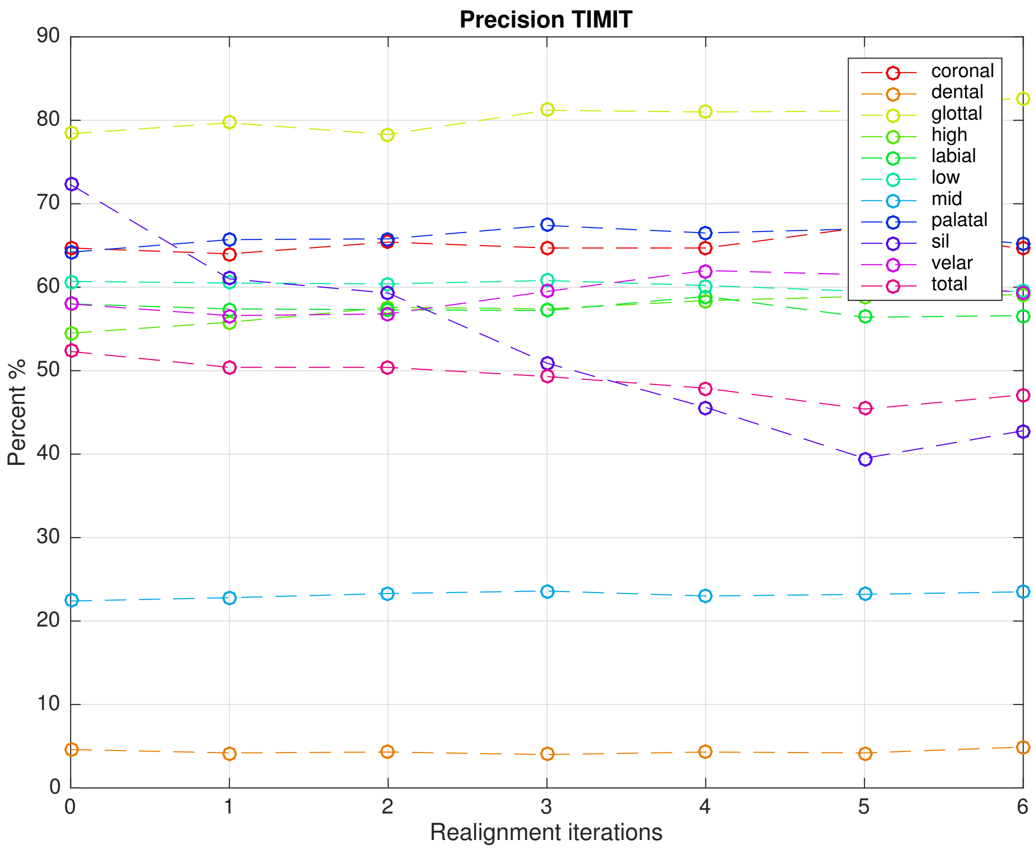


Figure A.8: RC TIMIT Precision realignment

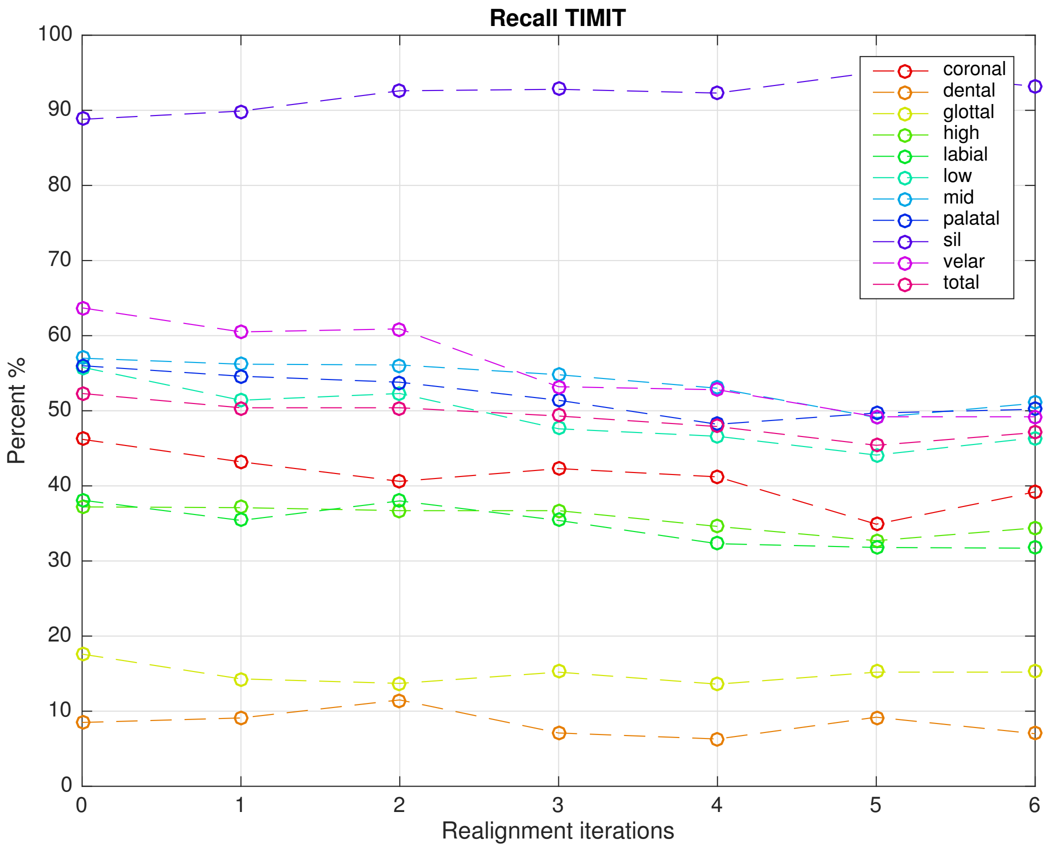


Figure A.9: RC TIMIT Recall realignment

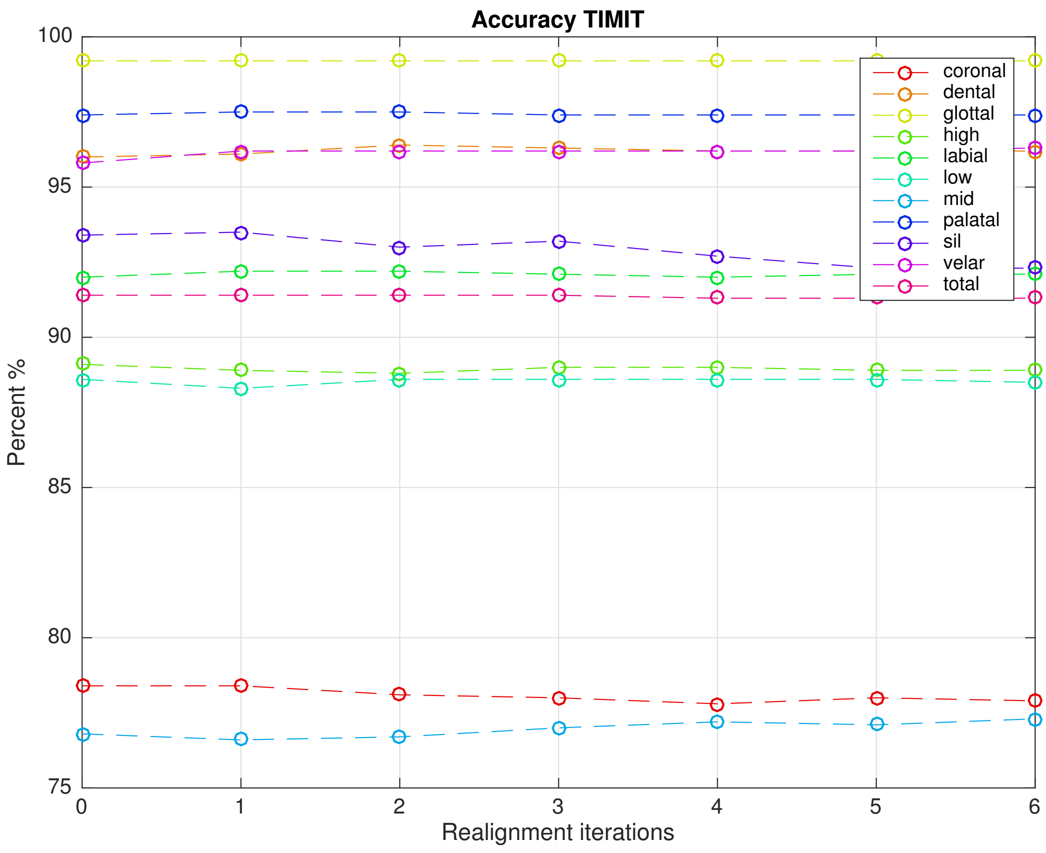


Figure A.10: CI TIMIT Accuracy realignment

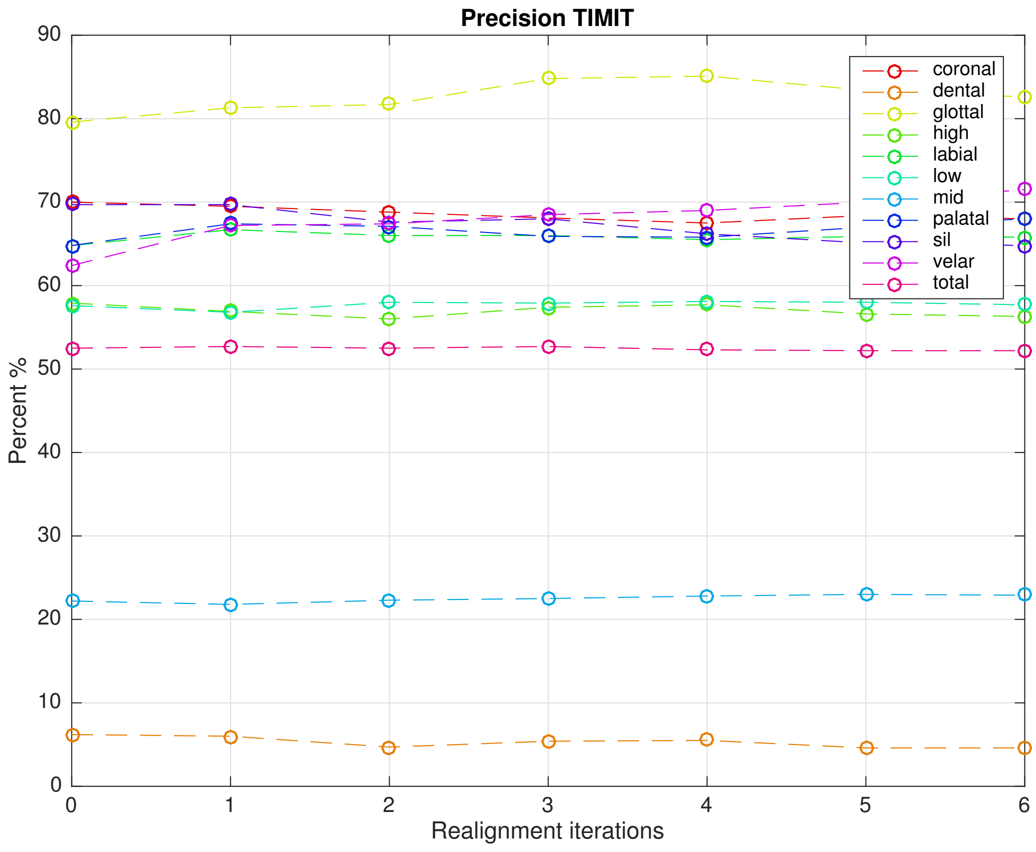


Figure A.11: CI TIMIT Precision realignment

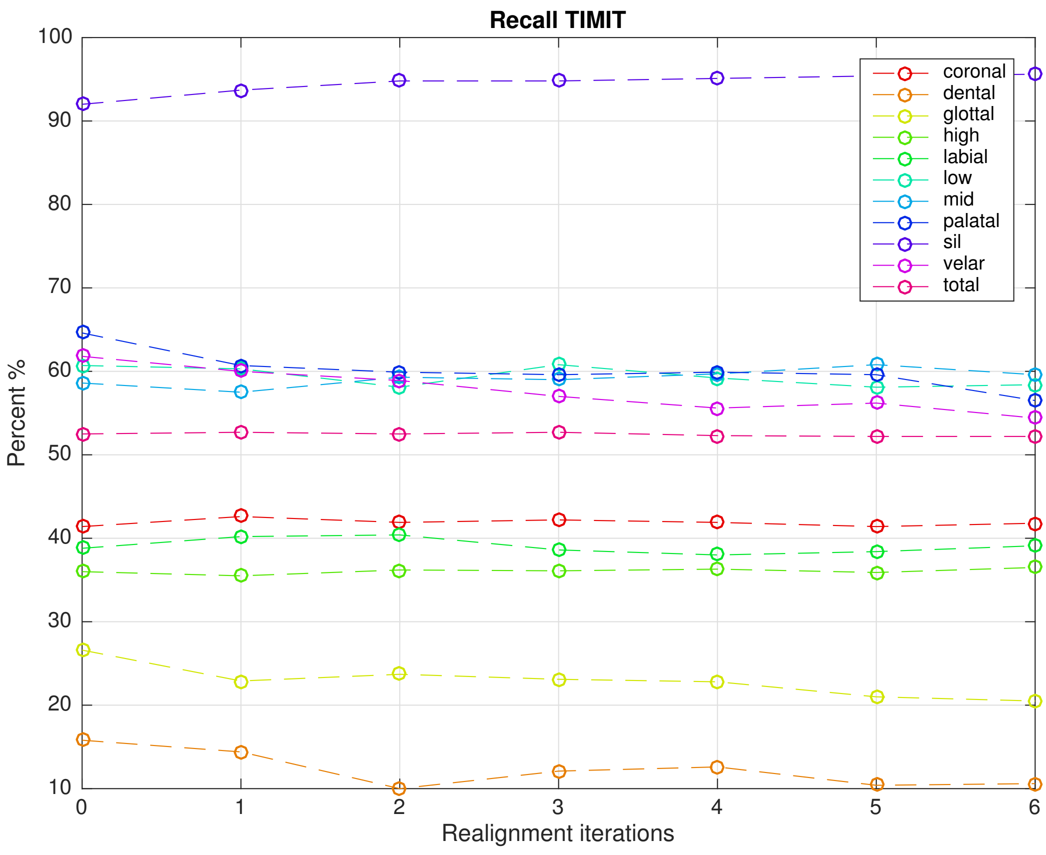


Figure A.12: CI TIMIT Recall realignment

A.1.3 OGI development set

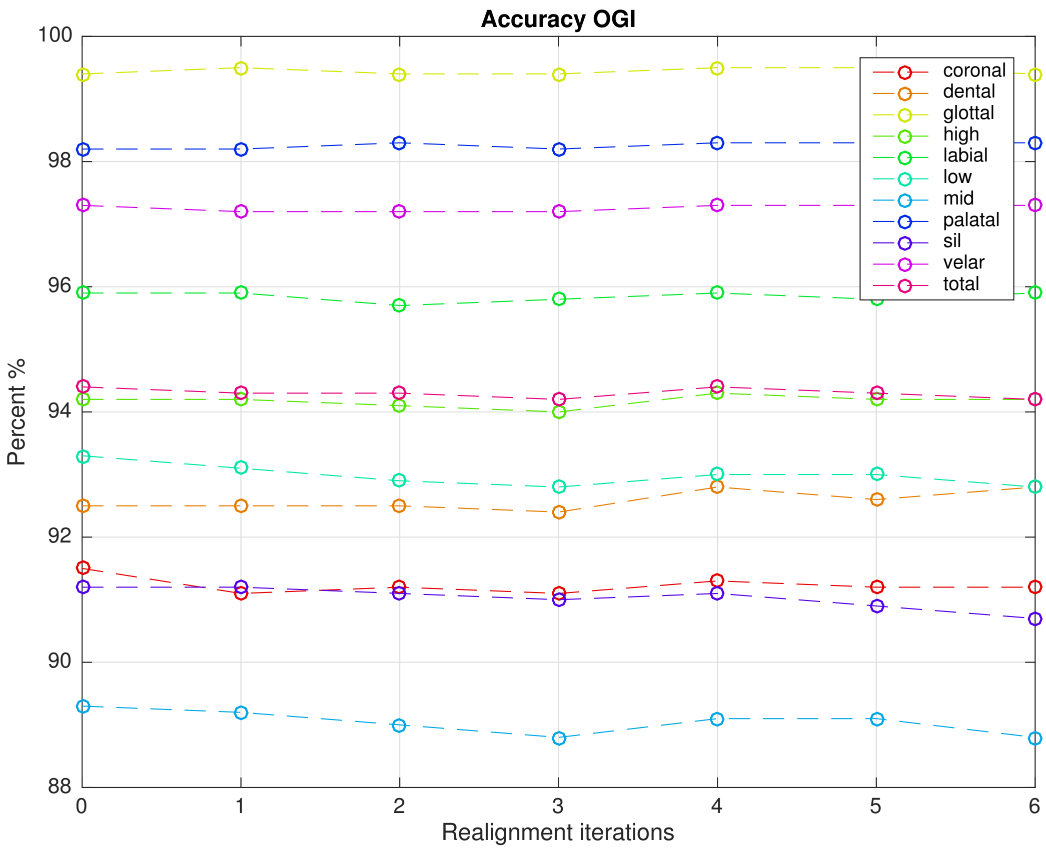


Figure A.13: RC OGI Accuracy realignment

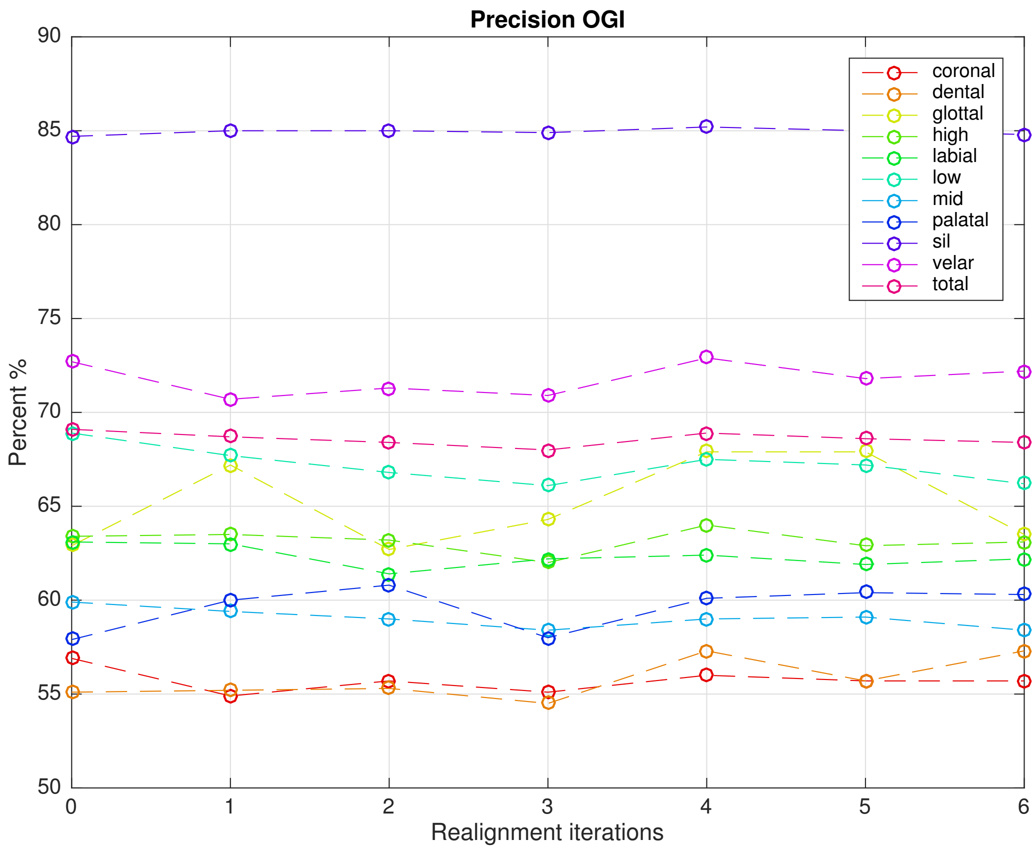


Figure A.14: RC OGI Precision realignment

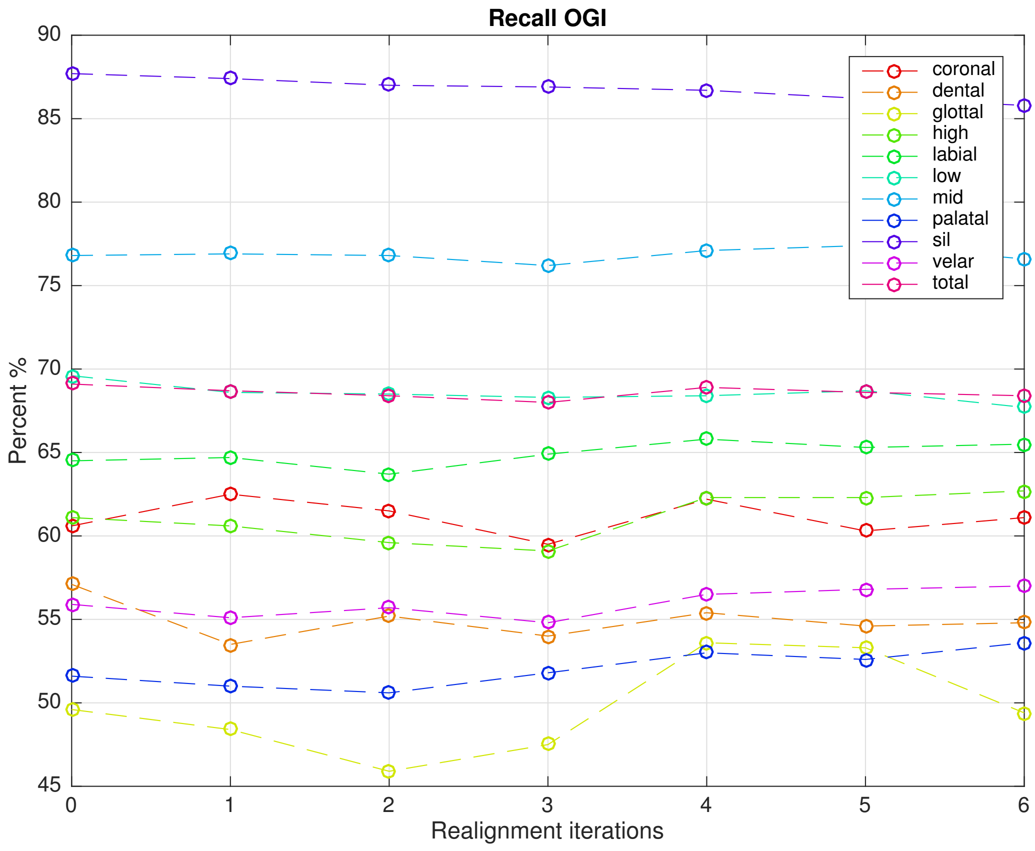


Figure A.15: RC OGI Recall realignment

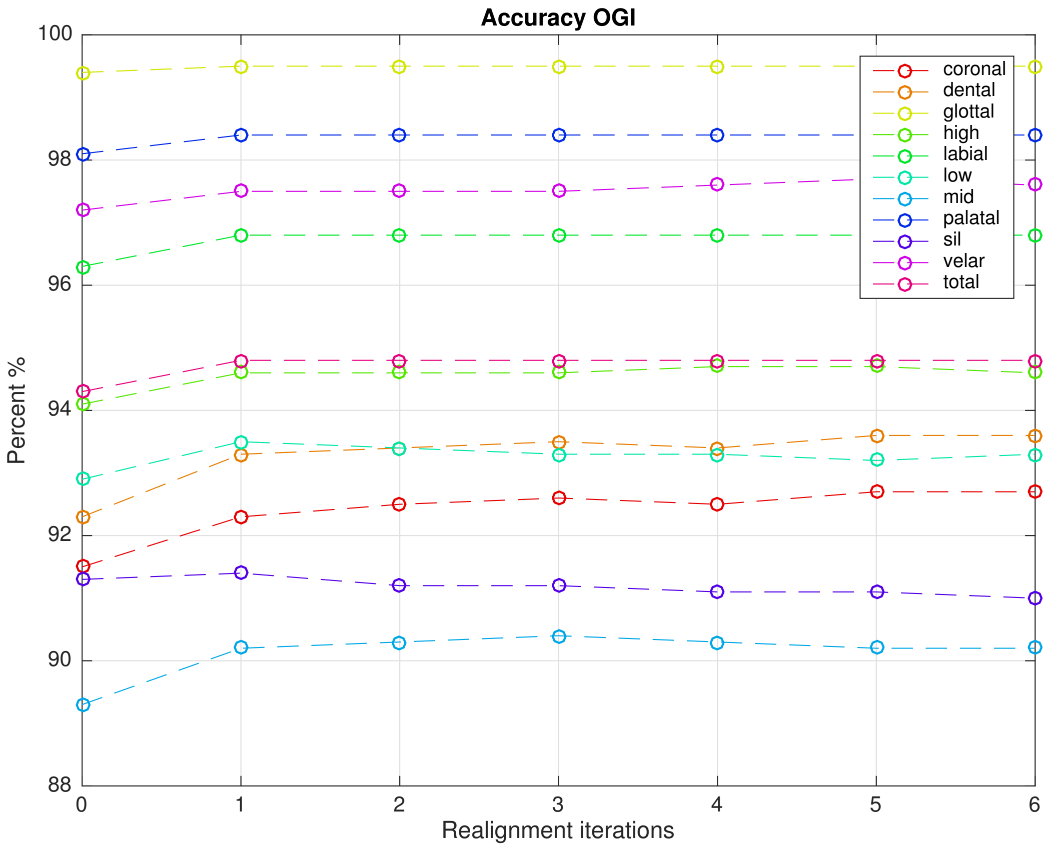


Figure A.16: CI OGI Accuracy realignment

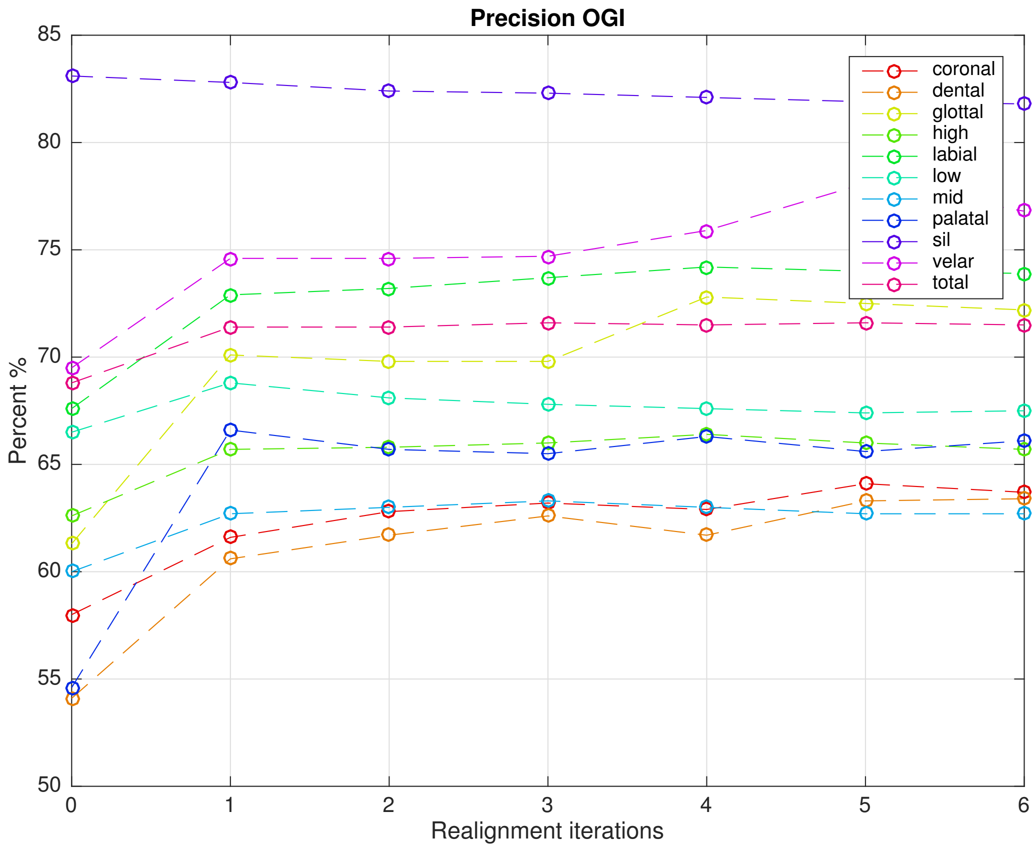


Figure A.17: CI OGI Precision realignment

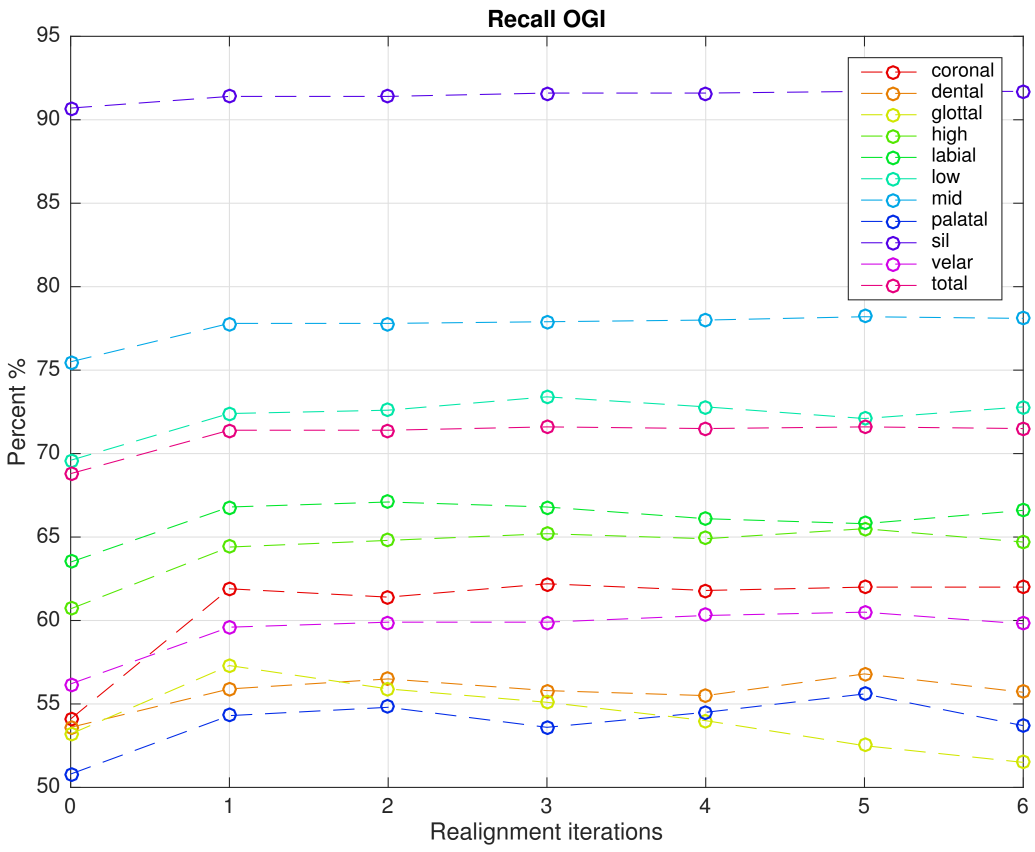


Figure A.18: CI OGI Recall realignment

A.2 Trained with english

A.2.1 Norwegian test set

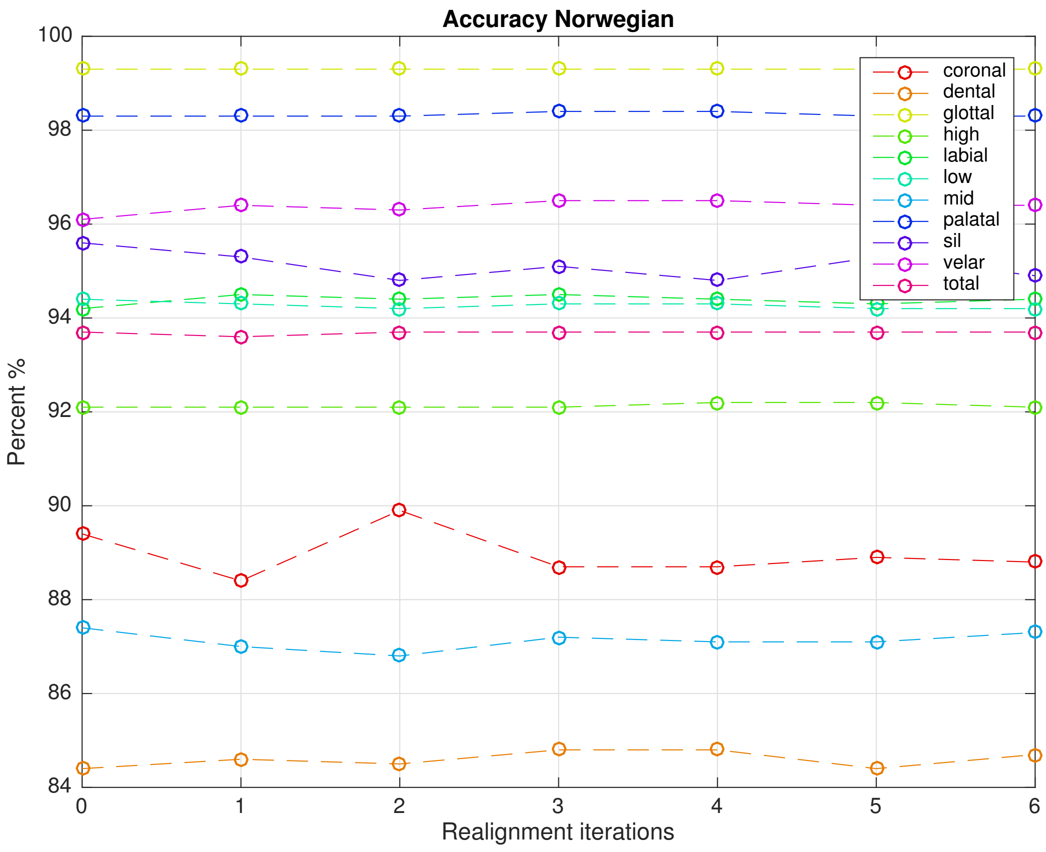


Figure A.19: CI Norwegian Accuracy realignment

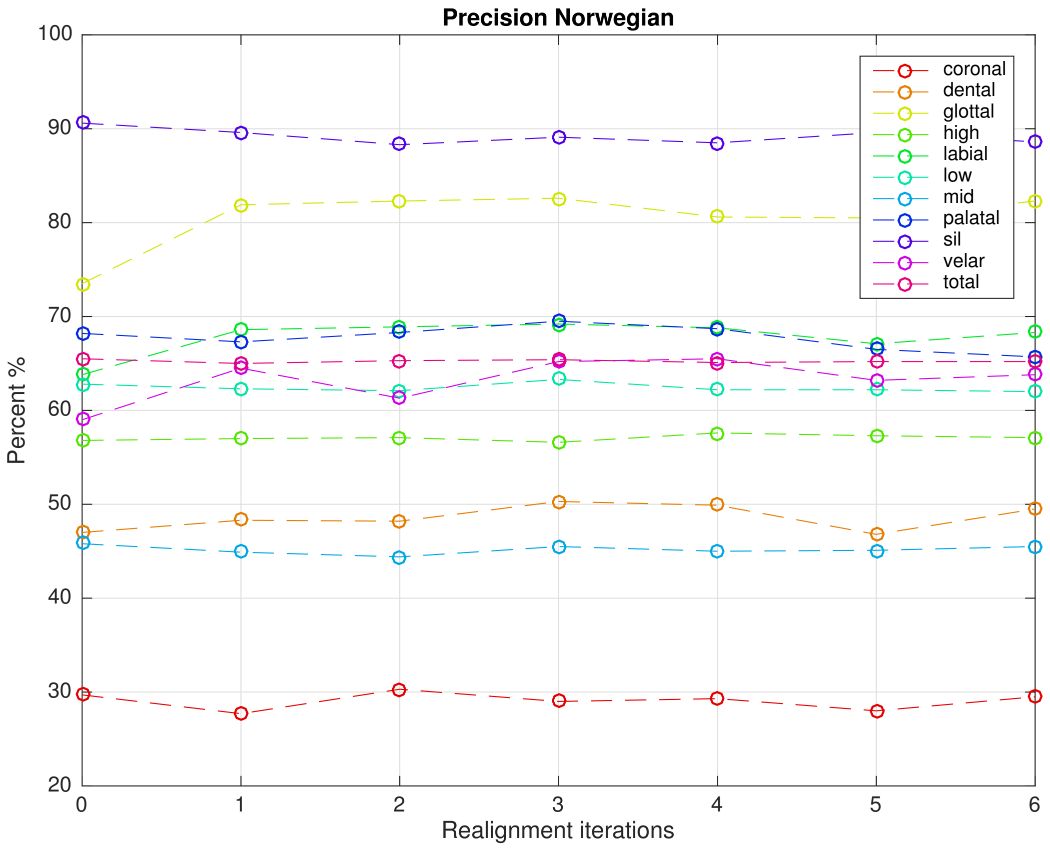


Figure A.20: CI Norwegian Precision realignment

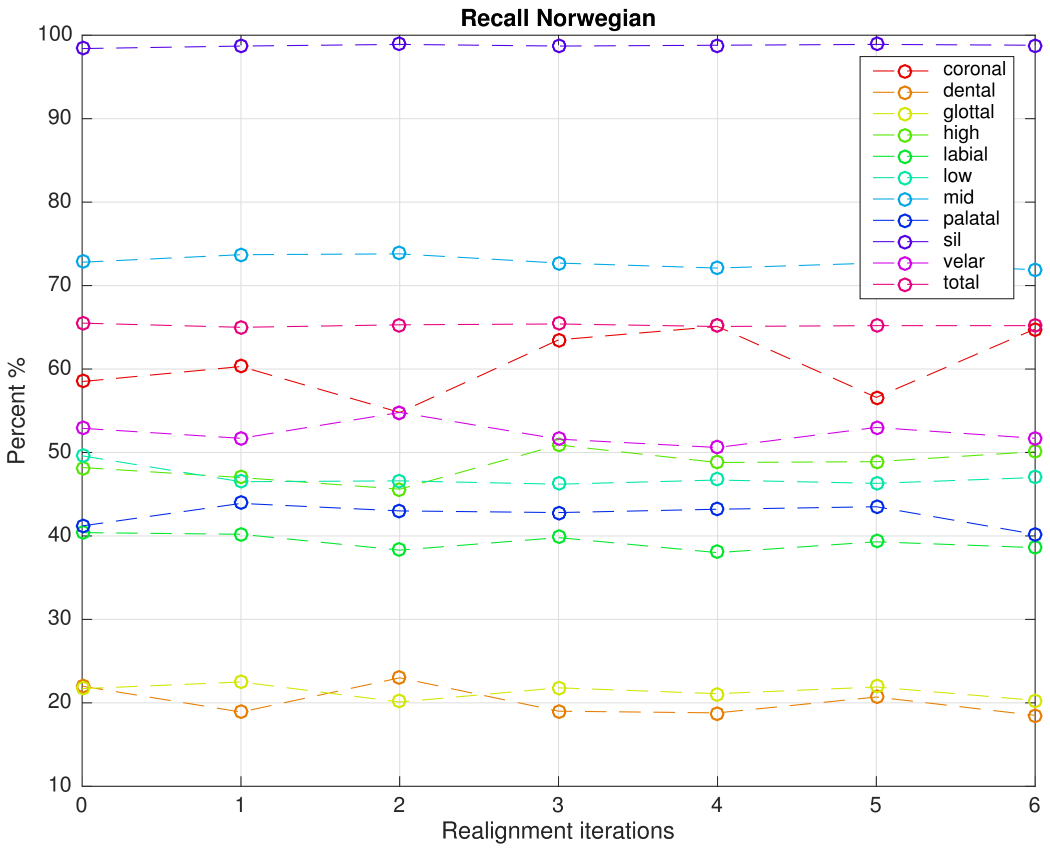


Figure A.21: CI Norwegian Recall realignment

A.2.2 OGI development set

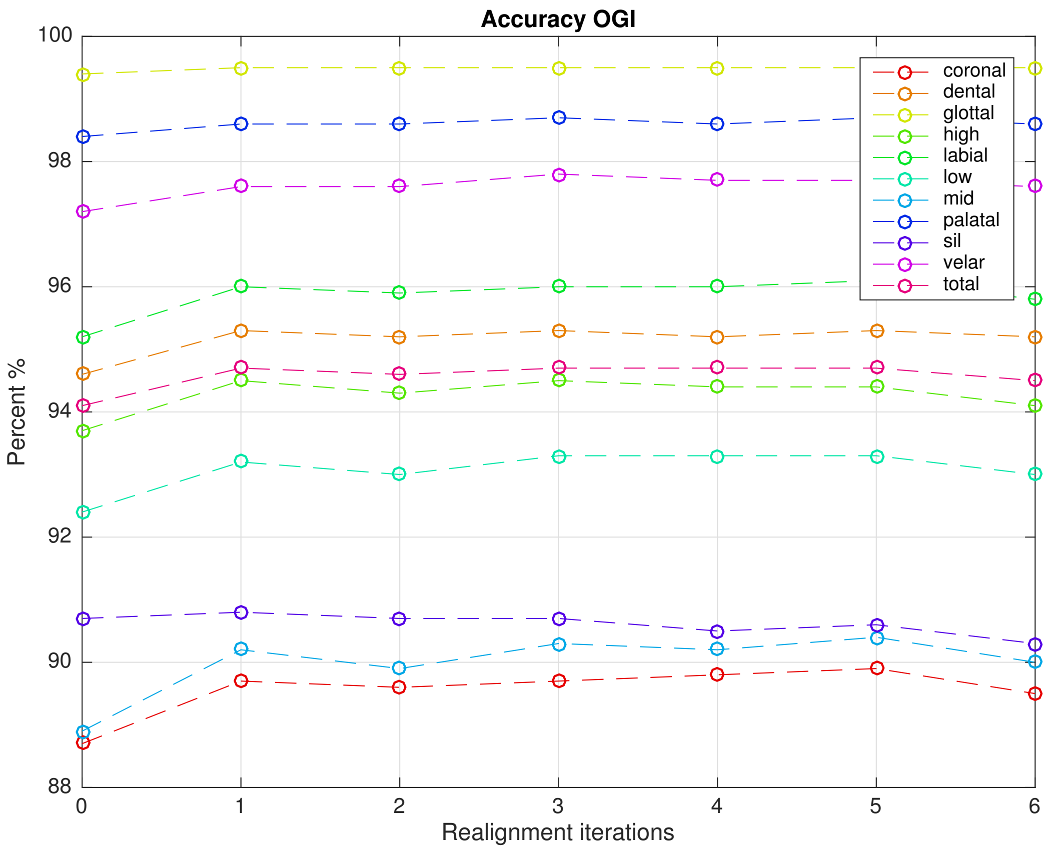


Figure A.22: CI OGI Accuracy realignment

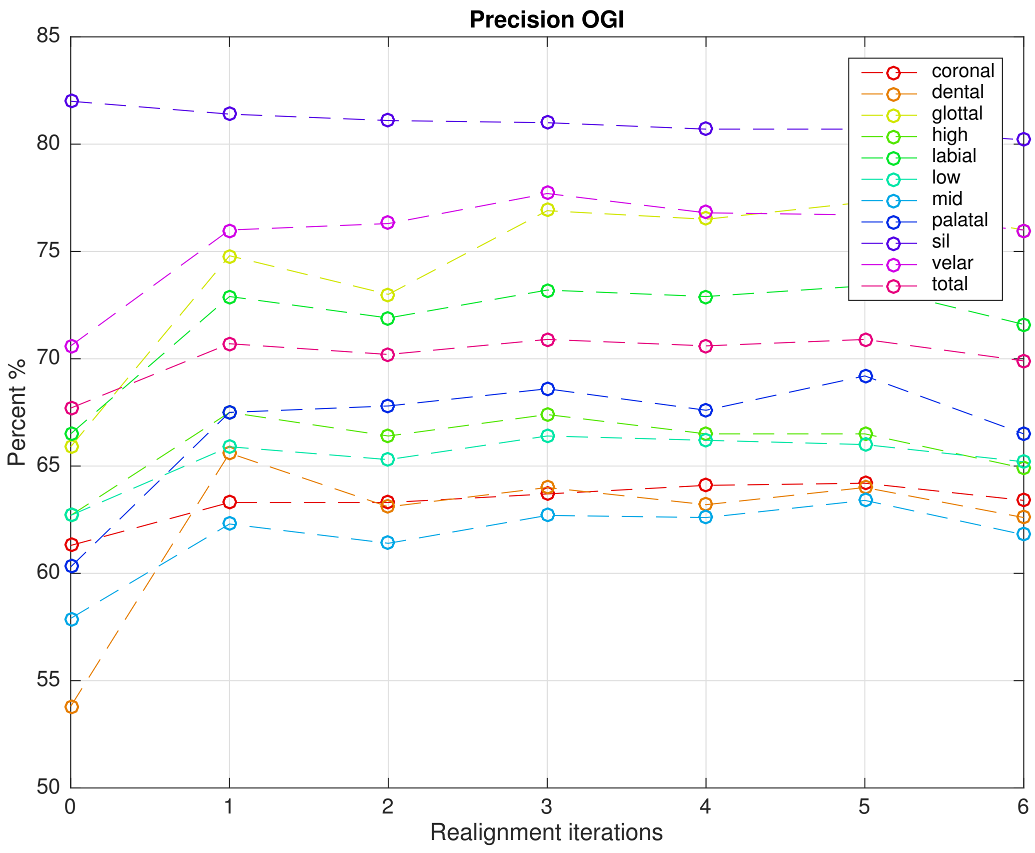


Figure A.23: CI OGI Precision realignment

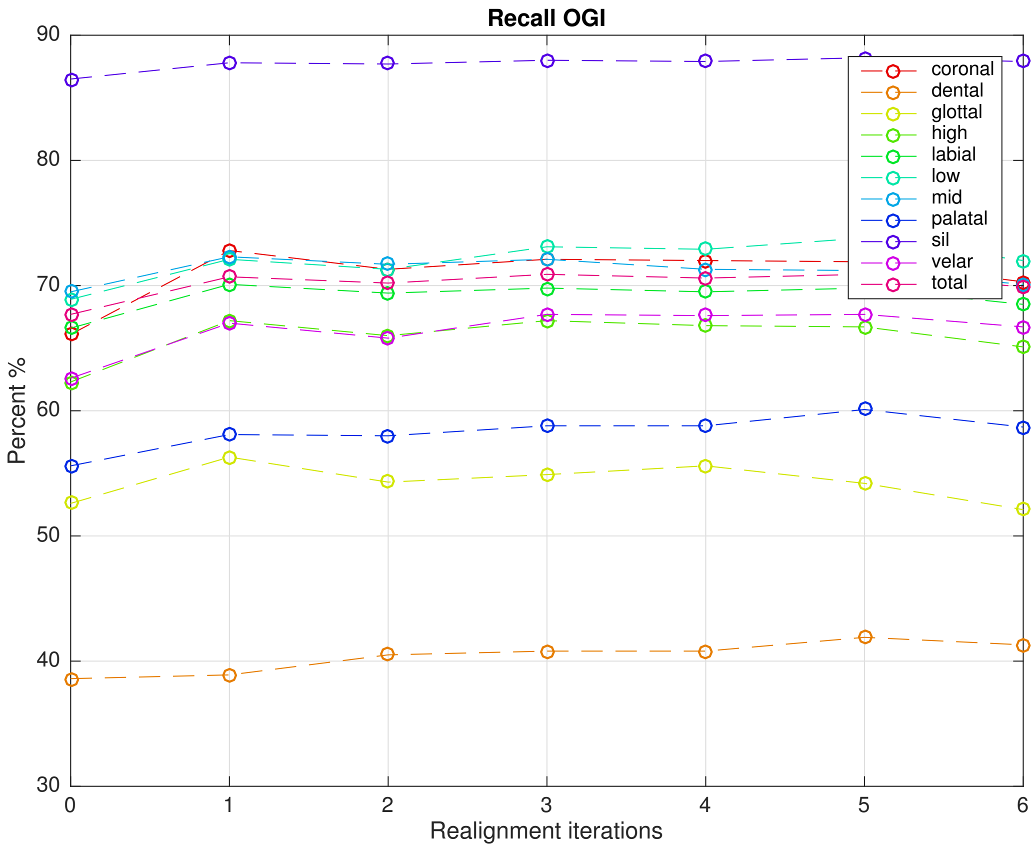


Figure A.24: CI OGI Recall realignment

Appendix B

LED script

```
#!/bin/bash
#Script to make RC, TC, LC and CI master label files from phone mlfs in ...
    batch.
#--Ãystein Staven, NTNU---
lans="eng spa ger hin jap man"
contexts="TC LC RC"
for lan in $lans; do

HLEd -n dicts/non_compacted/dict_CI_place.dict -l '*' -i  ...
    attMLFs/non_compacted/CI/refCI_$lan".mlf"  ...
    LEDS/non_compacted/$lan"_place.led" phoneMLFs/$lan".mlf"

for context in $contexts; do
HLEd -n dicts/non_compacted/dict_$context"_place_"$lan".dict" -l '*' -i ...
    attMLFs/non_compacted/$context/ref$context"_"$lan".mlf" ...
    LEDS/non_compacted/$context".led" ...
    attMLFs/non_compacted/CI/refCI_$lan".mlf"
sort dicts/non_compacted/dict_$context"_place_"$lan".dict" -o ...
    dicts/non_compacted/dict_$context"_place_"$lan".dict"
done
done

contexts="CI TC LC RC"
for context in $contexts; do
cd "/home/studenter/oyststa/LRE/create_place/EYSTEIN/ \
attMLFs/non_compacted/$context"
i=0
for lan in $lans; do
if [ "$i" = "0" ]; then
        cat ref$context"_"$lan".mlf" > combined_$context".mlf"
else
    tail -n +2 ref$context"_"$lan".mlf" >> combined_$context".mlf"
fi
    i=1
done
done

cd /home/studenter/oyststa/LRE/create_place/EYSTEIN
contexts="TC LC RC"
for context in $contexts; do
#finally, merge all dicts into one, for each context except CI.

for lan in $lans; do
    cat dicts/non_compacted/dict_$context"_place_"$lan".dict" >> temp1
done
```

```

sort -u temp1 > dicts/non_compacted/dict_$context"_place.dict"
rm temp1
# for lan in $lans; do
# rm dicts/non_compacted/dict_$context"_place_"$lan".dict"
# done
done
```

Appendix C

Combine MLFs

```
#!/bin/bash
#Script to combine mlf of different languages into one.
#--Ãystein Staven, NTNU--
context=CI
lans="eng ger hin jap man spa"
i=0
    for lan in $lans; do
if [ "$i" = "0" ]; then
        cat ref$context"_"$lan".mlf" > combined_$context".mlf"
else
    tail -n +2 ref$context"_"$lan".mlf" >> combined_$context".mlf"
fi
    i=1
done
```

Appendix D

Resample and filter

```
## !/bin/bash
##
## first resample to 8 kHz:
##RESAMPLING####

rm -r /prosjekt/studenter/oyststa/NAFTA
find /talebase/data/speech_raw/NAFTA/Speech/sennheiser -not -path ...
    '*/\.*'  -type f > naftafiles.scp

cd /talebase/data/speech_raw/NAFTA/Speech/sennheiser

mkdir /prosjekt/studenter/oyststa/NAFTA && mkdir ...
    /prosjekt/studenter/oyststa/NAFTA/sennheiser
find . -type d | cpio -pd /prosjekt/studenter/oyststa/NAFTA/sennheiser

cd ~/VAD_NTNU

fileset=`cat naftafiles.scp`
prefix="/talebase/data/speech_raw/NAFTA/Speech/sennheiser/"
suffix=".wav"
for file in `echo $fileset`; do
    foo=${file#$prefix}
    foo=${foo%$suffix}
    sox $file /prosjekt/studenter/oyststa/NAFTA/sennheiser/$foo.raw ...
        rate -s -a 8k
done
rm naftafiles.scp


##Now for the bandpass
####BANDPASS######

for file in `find /prosjekt/studenter/oyststa/NAFTA/sennheiser/ -not ...
    -path '*/\.*'  -type f`; do
    sox -r 8000 -e signed -b 16 -c 1 $file ...
        /prosjekt/studenter/oyststa/temp.raw  sinc 250-3450
    mv /prosjekt/studenter/oyststa/temp.raw $file
done
```

Appendix E

Find durations

```
#!/bin/bash
#Script to find duration of a set of files combined.
#--Ãystein Staven, NTNU
list=$1
files=`cat $list`
totdur=`echo "0" | bc`
for file in $files; do
durtemp=`sox -r 8000 -e signed -b 16 $file -n stat 2>&1 | sed -n ...
    's#^Length (seconds):[^0-9]*\([0-9.]*\)$#\1#p'`
durtemp=`echo $durtemp | bc`
totdur=$(echo "scale=2;$totdur+$durtemp" | bc)
done
echo "Duration of all files combined, in seconds:"
echo $totdur
echo "Durations of all files combined, in hours:"
totdur=$(echo "scale=2;$totdur/60/60" | bc)
echo $totdur
```

Appendix F

Feat. extract and ANN train

```
#!/bin/bash

context=$1
compacted=$2
tristates=$3
do_kalditrain=$4

#test input arguments
if [ \( "$compacted" = "yes"  \) -o \( "$compacted" = "no" \) ] &&  \
[ \( "$context" = "CI"  \) -o \( "$context" = "LC" \) -o \( "$context" ...
    = "RC" \) \
-o \( "$context" = "TC" \) ] && \
[ \( "$tristates" = "yes"  \) -o \( "$tristates" = "no" \) ] && \
[ \( "$do_kalditrain" = "yes"  \) -o \( "$do_kalditrain" = "no" \) ]; then
echo "---Starting process....."
else
printf "Usage: ./masterplace.sh context='CI|LC|RC|TC' ...
    compacted='yes|no' \
tristates='yes|no' do_kalditrain='yes|no' \n"
exit 1
fi

if [ "$compacted" = "yes" ]; then
    com_path="compacted"
else
    com_path="non_compacted"
fi

rm -r out tmp

dict="/home/studenter/oyststa/LRE/create_place/EYSTEIN/dicts/ \
"$com_path"/dict_"$context"_place.dict"
mlf="/home/studenter/oyststa/LRE/create_place/EYSTEIN/attMLFs \
    /"$com_path"/"$context"/combined_"$context".mlf"

./ogi_place.sh $mlf $dict > ogi_log.txt
if grep -Fxq  ogi_log.txt -e 'ERROR'; then
    printf "Errors found in ogi_place. See ogi_log.txt \n"
    exit 1
else
    printf "No errors in ogi_place. Continuing. \n"
fi

if [ "$tristates" = "yes" ]; then
    states=5;
    printf "Tristates chosen. Changing target number and HMM models \n"
    wc_temp=`wc "dicts/"$com_path"/dict_"$context"_place.dict"`
    numlabels=$(echo ${wc_temp[0]}|cut -d' ' -f1)
    numlabels=$(((numlabels+1)*3))
else
    states=3;
    printf "Single state chosen. Changing target number and HMM models \n"
    wc_temp=`wc "dicts/"$com_path"/dict_"$context"_place.dict"`
    numlabels=$(echo ${wc_temp[0]}|cut -d' ' -f1)
    numlabels=$((numlabels+1))
fi

###PFILE2KALDI####

cp -r /home/studenter/oyststa/LRE/create_place/EYSTEIN/out \
/home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/
cp -r /home/studenter/oyststa/LRE/create_place/EYSTEIN/tmp \
/home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/

cd /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi

if [ "$tristates" = "yes" ]; then
    ./eyscript_tristate.sh
else
    ./eyscript.sh
fi

##KALDI##
if [ "$do_kalditrain" = "yes" ]; then
#copying results to kaldi-trunk
cp -r ...
    /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/kaldiformat/ark \
/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/

cd /home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/
rm -r data-fbank
mkdir data-fbank
cd data-fbank
mkdir develop lang train test
cd /home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/exp
rm -r place_cd_ali_dev place_cd_ali_test place_cd_ali_train \
place_pretrain-dbn-dnn place_pretrain-dbn
mkdir place_cd_ali_dev place_cd_ali_test place_cd_ali_train
cd ..
. cmd.sh
. path.sh

copy-feats ark:ark/train.ark.txt ark, scp:/home/studenter/ \
oyststa/kaldi-trunk/egs/ogi/s5/data-fbank/train/train.ark,\
/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/data-fbank \
/train/feats.scp
copy-feats ark:ark/cv.ark.txt \
ark,scp:/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/ \
data-fbank/develop/cv.ark,\
/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/data-fbank/ \
develop/feats.scp
copy-feats ark:ark/test.ark.txt \
ark,scp:/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/ \
data-fbank/test/test.ark, \
/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/ \
data-fbank/test/feats.scp

cd exp
cd place_cd_ali_train
gzip -c ../../ark/ali_train_labs.ark > ali.1.gz
cd ../place_cd_ali_test
gzip -c ../../ark/ali_test_labs.ark > ali.1.gz
cd ../place_cd_ali_dev
gzip -c ../../ark/ali_cv_labs.ark > ali.1.gz
cd /home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/
dir=exp/place_pretrain-dbn
#pretraining
$cuda_cmd $dir/log/pretrain_dbn.log steps/nnet/pretrain_dbn.sh \
--rbm-iter 3 data-fbank/train/ $dir > pre.log
#fine tuning
dir=exp/place_pretrain-dbn-dnn
ali_train=exp/place_cd_ali_train
ali_cv=exp/place_cd_ali_dev
feature_transform=exp/place_pretrain-dbn/final.feature_transform

dbn=exp/place_pretrain-dbn/6.dbn

$cuda_cmd $dir/log/train_nnet.log steps/nnet/train.sh \
--feature-transform $feature_transform --dbn $dbn --hid-layers 0 \
--learn-rate 0.008 data-fbank/train data-fbank/develop \
data-fbank/lang $ali_train $ali_cv $dir > train.log

#mkdir /home/studenter/oyststa/LRE/backend/deepnetwork/
#place/$context"_"$states"_"$com_path
mkdir /home/studenter/oyststa/LRE/backend/deepnetwork \
/place/CI_5_non_compacted_0
cp /home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/exp/ \
place_pretrain-dbn-dnn/final.nnet \
/home/studenter/oyststa/LRE/backend/deepnetwork/place/ \
CI_5_non_compacted_0/final.nnet
cp /home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/exp/ \
place_pretrain-dbn-dnn/final.feature_transform \
/home/studenter/oyststa/LRE/backend/deepnetwork/place/ \
CI_5_non_compacted_0/final.feature_transform
tar -cvf /prosjekt/studenter/oyststa/EXPs/ \
${context}_${states}_${com_path}_0.tar.gz \
/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/exp
else
    printf "Kaldi training not chosen. Moving on \n"
fi
###BACKEND###

#Now, go to the backend and run the script "run_all.sh" with your ...
    preferred languages.
# Also change strbut2htk argument in SVite.sh

#cleanup
#rm /home/studenter/oyststa/LRE/create_place/EYSTEIN/ogi_log.txt
#rm /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/tempvar
echo "---ALL DONE--- "
```

Appendix G

Realignment

```
# #!/bin/bash
#
# #
# #test input arguments
# #check if files exists and if iterations is a number
# if [[ $iterations =~ ^-?[0-9]+$ ]]; then
#   echo "---Starting process.....";
# else
#   printf "Usage: ./realign_train.sh <num of iterations>\n"
#   exit 1
# fi

#First make the .ark file from the very first training, to subsequently ...
    do realignment

####NEW - KALDI FIRST FORWARD PASS

cd /home/studenter/oyststa/LRE/create_place/EYSTEIN
./masterplace.sh RC no yes yes

cd /home/studenter/oyststa/LRE/backend/tristates/scoring/OGI/scripts
    echo "Forwarding Phase 0"
/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
--use-gpu=yes ...
    --feature-transform=/home/studenter/oyststa/LRE/backend/deepnetwork/ \
place/RC_5_non_compacted_0/final.feature_transform ...
    /home/studenter/oyststa/LRE/ \
backend/deepnetwork/place/RC_5_non_compacted_0/final.nnet \
ark:"/home/studenter/oyststa/kaldi-trunk/egs \
/ogi/s5/data-fbank/train/train.ark"\
 ark:"/home/studenter/oyststa/LRE/backend/tristates \
 /scoring/OGI/arks/OGI_train_out.ark"

/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
--use-gpu=yes ...
    --feature-transform=/home/studenter/oyststa/LRE/backend/deepnetwork/ \
place/RC_5_non_compacted_0/final.feature_transform ...
    /home/studenter/oyststa/LRE/ \
    backend/deepnetwork/place/RC_5_non_compacted_0/final.nnet \
ark:"/home/studenter/oyststa/kaldi-trunk/egs/ \
    ogi/s5/data-fbank/develop/cv.ark"\
ark:"/home/studenter/oyststa/LRE/backend/ \
    tristates/scoring/OGI/arks/OGI_cv_out.ark"
```

```

/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
--use-gpu=yes ...
    --feature-transform=/home/studenter/oyststa/LRE/backend/deepnetwork\
/place/RC_5_non_compacted_0/final.feature_transform ...
    /home/studenter/oyststa/LRE/\
backend/deepnetwork/place/RC_5_non_compacted_0/final.nnet \
ark:"/home/studenter/oyststa/kaldi-trunk/egs/ \
ogi/s5/data-fbank/test/test.ark" \
ark:"/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/arks/OGI_test_out.ark"
# # ###END NEW PART

#loop over iterations
for i in {1..6}; do
    j=$[i-1]
if [ $i -eq 1 ]; then
cp /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/ \
out/timit_0-16k_23mel_1st/ \
pfiles/train.pfile ...
    /home/studenter/oyststa/LRE/backend/tristates/scoring/OGI/ \
originalpfiles/train.pfile
cp /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/ \
out/timit_0-16k_23mel_1st/ \
pfiles/test.pfile /home/studenter/oyststa/LRE/backend/ \
tristates/scoring/OGI/originalpfiles/test.pfile
cp /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi/ \
out/timit_0-16k_23mel_1st/ \
pfiles/cv.pfile /home/studenter/oyststa/LRE/backend/ \
tristates/scoring/OGI/originalpfiles/cv.pfile
else
cp "/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/realigned/train${j}.pfile" \
/home/studenter/oyststa/LRE/backend/tristates/scoring \
/OGI/originalpfiles/train.pfile
cp "/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/realigned/test${j}.pfile" \
/home/studenter/oyststa/LRE/backend/tristates/ /
scoring/OGI/originalpfiles/test.pfile
cp "/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/realigned/cv${j}.pfile" \
/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/originalpfiles/cv.pfile


echo "Forwarding phase $j"
####NEW - KALDI J'TH FORWARD PASS
/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
--use-gpu=yes --feature-transform= \
"/home/studenter/oyststa/LRE/backend/deepnetwork \
/place/RC_5_non_compacted_${j} \
/final.feature_transform" "/home/studenter/oyststa/LRE \
/backend/deepnetwork/place/RC_5_non_compacted_${j}/final.nnet" \
ark:"/home/studenter/oyststa/kaldi-trunk/egs/ogi/s5/ \
data-fbank/train/train.ark" \
ark:"/home/studenter/oyststa/LRE/backend/tristates/ \
scoring/OGI/arks/OGI_train_out.ark"

/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
 --use-gpu=yes --feature-transform= \
 "/home/studenter/oyststa/LRE/backend/deepnetwork \
 /place/RC_5_non_compacted_${j}/ \
 final.feature_transform" "/home/studenter/oyststa/LRE \
 /backend/deepnetwork/place/RC_5_non_compacted_${j}/final.nnet" \
ark:"/home/studenter/oyststa/kaldi-trunk/egs/ogi/ \
 s5/data-fbank/develop/cv.ark"\
  ark:"/home/studenter/oyststa/LRE/backend/ \
  tristates/scoring/OGI/arks/OGI_cv_out.ark"

/home/studenter/oyststa/kaldi-trunk/src/nnetbin/nnet-forward ...
    --apply-log=true \
 --use-gpu=yes --feature-transform="/home/studenter/oyststa \
 /LRE/backend/deepnetwork\
 /place/RC_5_non_compacted_${j}/final.feature_transform" \
  "/home/studenter/oyststa/\
 LRE/backend/deepnetwork/place/RC_5_non_compacted_${j}/final.nnet" \
ark:"/home/studenter/oyststa/kaldi-trunk/egs/ \
ogi/s5/data-fbank/test/test.ark" \
 ark:"/home/studenter/oyststa/LRE/backend/tristates/ \
 scoring/OGI/arks/OGI_test_out.ark"
###END NEW PART
fi

for set in "train" "cv" "test"; do
cd /home/studenter/oyststa/LRE/backend/tristates/scoring/OGI/scripts
perl prior_pfile.pl "../originalpfiles/${set}.pfile" ../dict/dict \
../priors/"priors_reali${j}_${set}" #find priors
perl post2pfile.pl "../arks/OGI_${set}_out.ark" \
"../posteriori/post_reali${i}_${set}.pfile" #make new posteriorfile
pfile_realign -i "../originalpfiles/${set}.pfile" -o \
 "../realigned/${set}${i}.pfile" -p ...
     "../posteriori/post_reali${i}_${set}.pfile" \
 -v=true -l #realign
done

#now let's do the training
cd /home/studenter/oyststa/LRE/Pfile2Kaldi/create_kaldi
./eyscript_realign.sh $i
cd /home/studenter/oyststa/LRE/backend/tristates/scoring/OGI/scripts

./kaldtrain.sh $i

done

#last priors for the i'th iteration
for set in "train" "cv" "test"; do
cd /home/studenter/oyststa/LRE/backend/tristates/scoring/OGI/scripts
perl prior_pfile.pl "../realigned/${set}${i}.pfile" ../dict/dict \
"../priors/priors_reali${i}_${set}" #find priors
done

echo "All iterations done!"
```

Appendix H

Evaluate decoding

```
#!/usr/bin/perl
##Script to compare a prior pfile with a decoded mlf
##
##Ãystein Staven, NTNU, 7 june 2016
#
#Requires an input mlf made with arguments <-o S>

use List::MoreUtils qw(firstidx);
#use strict;
#use warnings;
###########################################
##  Check input arguments
###########################################
# $num_args = $#ARGV+1;
# if ($num_args != 2) {
#     print "\nUsage: prior_ark.pl <input pfile> <output ark> ...
    <priorlist>\n";
#     exit;
# }

######################
##  Input arguments
######################
$input_pfile=$ARGV[0];
$dict="dict_CI_place.dict";
$dictlatex="dictlatex";
#$list="cv.scp";
$input_mlf=$ARGV[1];
#############################
##  Get some info about pfile
##############################
@pfile_info=split(/[\n,\s]+/,`pfile_info -i $input_pfile`);
#make original labs
system("pfile_print -i $input_pfile -o temppfile.txt -q");
$pfile_sentences=$pfile_info[1];
$pfile_frames=$pfile_info[3];
#print "@info\n";
#print "$pfile_sentences\n";
#print "$pfile_frames\n";

$numatts=`cat $dict | wc -l`;
###################
##  Initialize
######################
my @starts_array=();
my @stops_array=();
my @phns_array=();
#$time_shift=10*(1e-3); #frame shift constant
$time_shift=1e5;
#my $dir = './tmpdecode'; #temporary folder for recs
#mkdir $dir unless -d $dir;
@attributes=();
    @attributes = do {
        open $fh, "<", $dict
            or die "could not open $dict: $!";
        <$fh>;
    };

close($fh);
@attributeslatex=();
    @attributeslatex = do {
        open $fh, "<", $dictlatex
            or die "could not open $dictlatex: $!";
        <$fh>;
    };

close($fh);
#####################################
##  Import MLF and make new continous mlf, then print to two new files
######################################
$mlf_sentences=`grep $input_mlf -e '.rec' | wc -l`;

if ($pfile_sentences != $mlf_sentences){
    die("MLF and pfile has different number of sentences\n");
}
open(FOUT1,">newref.txt") || die("Cannot open newref.txt \n");
open(FOUT2,">reclabs.txt") || die("Cannot open reclabs.txt \n");
open(FIN,"$input_mlf") || die("Cannot open MLF file\n");

    while($line = <FIN>) {
        chomp $line;
        next if ($line =~ m/MLF/);  #skip if it's the MLF header
        next if ($line =~ m/^\./);  #skip if it's the end of sentence
        if ($line =~ m/.rec/) { #this means a new file
        for($el=0;$el<=$#phns_array;$el++) {
            print FOUT1 "$starts_array[$el] $stops_array[$el] ...
                $phns_array[$el]\n";
            $idx = firstidx {/$phns_array[$el]/} @attributes;
            print FOUT2 "$idx\n";
        }
        @starts_array=();
        @stops_array=();
        @phns_array=();
        next;
        }
        ($start, $stop, $phn) = split /\s+/, $line, 3;
        $diff=$stop-$start;
        #section that takes care of time jumps more than time_shift:
        if($diff == $time_shift){
        push @starts_array, $start;
        push @stops_array, $stop;
        push @phns_array, $phn; }
        else {
            $counter=0;
            #push @starts_array, $start;
            $newtime=$start;
            while($counter != $diff){
                push @starts_array, $newtime;
                $newtime=$newtime+$time_shift;
                push @stops_array, $newtime;
                push @phns_array, $phn;
                $counter=$counter+$time_shift;
        }

    }
}
#Printing the final file
for($el=0;$el<=$#phns_array;$el++) {
    print FOUT1 "$starts_array[$el] $stops_array[$el] $phns_array[$el]\n";
    $idx = firstidx {/$phns_array[$el]/} @attributes;
    print FOUT2 "$idx\n";
}
close(FIN);
close(FOUT);

open(FIN,"temppfile.txt") || die("Cannot open temppfile.txt\n");
open(FOUT,">lablabs.txt") || die("Cannot open lablabs.txt\n");
while($line = <FIN>) {
    chomp $line;
    @columns= split / /, $line;
    print FOUT "$columns[50]\n";
}
close(FIN);
close(FOUT);

unlink("temppfile.txt");
#unlink("newref.txt");

####################################
##  Make binary recognizer vectors
####################################
open(FIN1,"reclabs.txt") || die("Cannot open reclabs.txt");
open(FIN2,"lablabs.txt") || die("Cannot open lablabs.txt");
open(FOUT1,">binreclabs.txt") || die("Cannot open binreclabs.txt");
open(FOUT2,">binlablabs.txt") || die("Cannot open binlablabs.txt");

while($line = <FIN1>) {
    for($el=0;$el<=$numatts-1;$el++) {
        if ($el == $line)
                {print FOUT1 "1\n";}
            else{print FOUT1 "0\n";}
    }
}

while($line = <FIN2>) {
    for($el=0;$el<=$numatts-1;$el++) {
        if ($el == $line)
                {print FOUT2 "1\n";}
            else{print FOUT2 "0\n";}
    }
}

close(FIN1); close(FIN2);
close(FOUT1); close(FOUT2);

##########################################
##  Now for the evaluation for each class
##########################################
open(LATEX,">latex.txt") || die("Cannot open latex.txt");
open(MATLAB,">matlab.txt") || die("Cannot open matlab.txt");
@lablabs=(); #import truth labels
```

```
    @lablabs = do {
        open $fh, "<", "binlablabs.txt"
            or die "could not open binlablabs.txt: $!";
        <$fh>;
    };
close($fh);
@reclabs=(); #import recognized labels
    @reclabs = do {
        open $fh, "<", "binreclabs.txt"
            or die "could not open binreclabs.txt: $!";
        <$fh>;
    };
close($fh);

for($class=0;$class<=$numatts-1;$class++) {
     $TP=0; #true positives
     $FP=0; #false positives
     $TN=0; #true negatives
     $FN=0; #false negatives
$globalcount=0;
$counter=0;
foreach my $line (@reclabs) {
    #print "counter: $counter\n";
    #now check for TP,FP,TN,FN
if ($counter==$class)
{
    if ($line==1 and $lablabs[$globalcount]==1) {$TP++;}
    if ($line==1 and $lablabs[$globalcount]==0) {$FP++;}
    if ($line==0 and $lablabs[$globalcount]==0) {$TN++;}
    if ($line==0 and $lablabs[$globalcount]==1) {$FN++;}
}
    $counter++;
    $globalcount++;
    if ($counter == $numatts){$counter=0;}
}
#print "$class: TP: $TP FP: $FP TN: $TN FN $FN\n";
if ($TN == 0 || $TP == 0) {$Acc = $Prec = $Reca = $FScore = "nAn";}
else{
$Acc=100*($TP+$TN)/($TP+$FP+$TN+$FN);
$Prec=100*($TP)/($TP+$FP);
$Reca=100*($TP)/($TP+$FN);
$FScore=(2*$TP)/((2*$TP)+$FP+$FN);
printf MATLAB ("%5.1f %5.1f %5.1f %5.2f \n",$Acc,$Prec,$Reca,$FScore);
}
printf("$attributeslatex[$class] Acc=%5.2f\%, Prec=%5.2f\%, ...
    Reca=%5.2f\%, FScore=%5.2f\n",$Acc,$Prec,$Reca,$FScore);
# print latex formatted table
printf LATEX ("%s %s",$attributeslatex[$class],"\&");
printf LATEX ("%5.1f \& %5.1f \& %5.1f \& %5.2f ...
    \\\\\n",$Acc,$Prec,$Reca,$FScore);
}

##################################################
##  Now for the evaluation for each frame regardless of class
##################################################
$TP=0; #true positives
$FP=0; #false positives
$TN=0; #true negatives
$FN=0; #false negatives
$globalcount=0;
$counter=0;
foreach my $line (@reclabs) {
    #print "counter: $counter\n";
    #now check for TP,FP,TN,FN
    if ($line==1 and $lablabs[$globalcount]==1) {$TP++;}
    if ($line==1 and $lablabs[$globalcount]==0) {$FP++;}
    if ($line==0 and $lablabs[$globalcount]==0) {$TN++;}
    if ($line==0 and $lablabs[$globalcount]==1) {$FN++;}

    $globalcount++;
}
#print "Total: TP: $TP FP: $FP TN: $TN FN $FN\n";
if ($TN == 0 || $TP == 0) {$Acc = $Prec = $Reca = $FScore = "nAn";}
else{
$Acc=100*($TP+$TN)/($TP+$FP+$TN+$FN);
$Prec=100*($TP)/($TP+$FP);
$Reca=100*($TP)/($TP+$FN);
$FScore=(2*$TP)/((2*$TP)+$FP+$FN);
printf MATLAB ("%5.1f %5.1f %5.1f %5.2f \n",$Acc,$Prec,$Reca,$FScore);
}
printf("\nTotal: Acc=%5.2f\%, Prec=%5.2f\%, Reca=%5.2f\%, ...
    FScore=%5.2f\n",$Acc,$Prec,$Reca,$FScore);

# print latex formatted table
printf LATEX ("Total \&");
printf LATEX ("%5.1f \& %5.1f \& %5.1f \& %5.2f ...
    \\\\\n",$Acc,$Prec,$Reca,$FScore);

close(LATEX);
close(MATLAB);
```

Appendix I

Backend top script (Siniscalchi)

```
#!/bin/bash
#
#$ -cwd
#$ -j y
#$ -S /bin/bash
#
###THIS ASSUMES THAT USENGNSD IS ENGLISH AND SPANISHNCD IS SPANISH.  ...
    THERE IS NO TAWAIN,USENGSD,SPANISHCD

#trainSets="rec_canadian_train rec_farsi_train rec_german_train"
####UNCOMMENT FOLLOWING TWO LINES FOR FULL EVALUATON
trainSets="rec_arabic_train rec_english_train rec_farsi_train ...
    rec_french_train rec_german_train rec_hindi_train rec_japanese_train ...
    rec_korean_train rec_mandarin_train rec_spanish_train ...
    rec_tamil_train rec_vietnamese_train"
trainlans=("arabic" "english" "farsi" "french" "german" "hindi" ...
    "japanese" "korean" "mandarin" "spanish" "tamil" "vietnamese" ...
    "norwegian")
#trainSets="rec_arabic_train"
#trainlans=("arabic")

####COMMENT THE FOLLOWING LINE FOR FULL EVALUATION
#trainlans=("canadian" "farsi" "german") #changed from arabic and ...
    german to canadian and farsi -Ãžystein

testSet="rec_lid03e1_30"

numSV=$1
weight=$2
rm -rf nist
mkdir nist

for set in `echo $trainSets`
  do
    echo $set
done

echo "special treatment for Norwegian"
for set in `echo rec_norwegian_train`; do
    mkdir nist/$set
    perl breakNIST.pl $set.mlf place
    ls nist/$set > nist/curSet.scp
    perl replaceNAFTAPhns.pl place $set
    mkdir nist/temp30
    matlab -nodisplay -nodesktop -nojvm -r ...
        "combineNAFTA('nist/curSet.scp','$set')"
    rm -rf nist/$set
    mv nist/temp30 nist/$set
    perl listFiles.pl "nist/$set" >> nist/all.scp
done

for set in `echo $trainSets`
  do
  echo $set
  mkdir nist/$set
  perl breakNIST.pl $set.mlf place
  perl listCALL.pl $set
  perl replacePhns.pl place $set
  mkdir nist/temp30
  matlab -nodisplay -nodesktop -nojvm -r ...
      "combineCALL('nist/setCount.txt','$set')"
  rm -rf nist/$set
  mv nist/temp30 nist/$set
  perl listFiles.pl "nist/$set" >> nist/all.scp
done

perl mkLabels.pl "nist/all.scp" "nist/train.lab"

echo "Done making training files"
echo "Making testing file"
mkdir nist/$testSet
perl breakNIST.pl $testSet.mlf
perl listNIST.pl $testSet

perl replacePhns.pl place $testSet
mkdir nist/temp30
matlab -nodisplay -nodesktop -nojvm -r ...
    "combineNIST('nist/setCount.txt','$testSet')"
rm -rf nist/$testSet
mv nist/temp30 nist/$testSet
perl listFiles.pl "nist/$testSet" > nist/test.scp
perl mkTestLabs.pl "$testSet"
echo "Done making testing files"

numPhns=`wc -l phn_place | awk '{ print $1 }'`
echo "Num svs = $numSV"
matlab -nodisplay -nodesktop -nojvm -r ...
    "quadlsaNIST('nist/all.scp','nist/test.scp',$numPhns,$numSV)"
#matlab -r "quintlsa('nist/all.scp','nist/test.scp',$numPhns,$numSV)" ...
    #was commented
matlab -nodisplay -nodesktop -nojvm -r "normVects('nist')"
##mv nist/train.csv nist/trainNorm.csv #was commented
##mv nist/test.csv nist/testNorm.csv    #was commented

for ((pos=0;pos<${#trainlans[@]};pos++))
  do
  set=${trainlans[$pos]}
  echo "$set"
  perl mkSVM1vAll.pl nist/trainNorm.csv nist/train.lab ${set} ...
      nist/${set}.dat
  perl mkSVM1vAll.pl nist/testNorm.csv nist/test.lab $set nist/test.dat
  svm_learn -j ${weight} nist/${set}.dat nist/theModel
  svm_classify nist/test.dat nist/theModel nist/${set}.pred
  svm_classify nist/${set}.dat nist/theModel nist/${set}.predT
#  perl svm2scores.pl nist $set nist/test.lab nist/${set}.pred
done > NISTsvmlog.txt
```

Appendix J

LSA (Siniscalchi)

```matlab
function quadlsaNIST(trainList,testList,numWords,numSV)
% trainList=('nist/all.scp');
% testList=('nist/test.scp');
% numWords=10;
% numSV=300;
% numSV = numSV
fileList = textread(trainList,'%s');
C = zeros(numWords+numWords*numWords+numWords^3,length(fileList));
numDocs = length(fileList);
for fileNum = 1:numDocs
  str = sprintf('%s',fileList{fileNum});
  x = dlmread(str);
  xsize = size(x);
%  x = x(:,3);
  for n = 1:numWords
    C(n,fileNum) = length(find(x == n));
  end

  temp = zeros(numWords,numWords);
  for n = 2:length(x)
    temp(x(n),x(n-1)) = temp(x(n),x(n-1)) + 1;
  end
  C((numWords+1):(numWords+numWords^2),fileNum) = temp(:);

  temp = zeros(numWords,numWords,numWords);
  for n = 3:length(x)
    temp(x(n),x(n-1),x(n-2)) = temp(x(n),x(n-1),x(n-2)) + 1;
  end
  C((numWords+numWords^2+1):end,fileNum) = ...
      temp(:);

end
dlmwrite('nist/Cquad.txt',C,' '); %<---Remove this after done debugging

W = zeros(size(C));
n = sum(C);
t = sum(C');
epsilon = zeros(1,numWords+numWords*numWords+numWords^3);

for ii = 1:length(t)
  if (t(ii) > 0)
    temp = C(ii,:)./t(ii);
    temp = temp.*log(temp);
    I = find(C(ii,:) == 0);
    temp(I) = 0;
    epsilon(ii) = -1/log(numDocs) * sum(temp);
  end
end

dlmwrite('nist/wordEntropies.txt',epsilon,'\n'); %<---Remove this after ...
    done
                                          %debugging

for ii = 1:length(t)
  for jj = 1:numDocs
    W(ii,jj) = (1-epsilon(ii))*C(ii,jj)/n(jj);
  end
end
dlmwrite('nist/W.txt',W,' ')
clear C
[U,S,V] = svds(W,numSV);
clear W
B = S*V';
%B = V';
dlmwrite('nist/train.csv',B','delimiter',',','precision','%.10f')
clear S
clear V
fileList = textread(testList,'%s');
C = zeros(numWords+numWords*numWords+numWords^3,length(fileList));
numDocs = length(fileList);
for fileNum = 1:numDocs
  str = sprintf('%s',fileList{fileNum});
  x = dlmread(str);
%  x = x(:,3);
  for n = 1:numWords
    C(n,fileNum) = length(find(x == n));
  end
  temp = zeros(numWords,numWords);
  for n = 2:length(x)
    temp(x(n),x(n-1)) = temp(x(n),x(n-1)) + 1;
  end
  C((numWords+1):(numWords+numWords^2),fileNum) = temp(:);
  temp = zeros(numWords,numWords,numWords);
  for n = 3:length(x)
    temp(x(n),x(n-1),x(n-2)) = temp(x(n),x(n-1),x(n-2)) + 1;
  end
  C((numWords+numWords^2+1):end,fileNum) = ...
      temp(:);

end
dlmwrite('nist/Cquadtest.txt',C,' '); %<---Remove this after done debugging
Q = zeros(size(C));
n = sum(C);

for ii = 1:length(t)
  for jj = 1:numDocs
    Q(ii,jj) = (1-epsilon(ii))*C(ii,jj)/n(jj);
  end
end

dlmwrite('nist/Q.txt',Q,' ')
Q = U'*Q;
dlmwrite('nist/test.csv',Q','delimiter',',','precision','%.10f')
```

```



quit
```

Appendix K

Find priors

```
#!/bin/perl
#This script calls on pfile_print to find prior probabilities ...
    associated with each class
#-----------Ãystein Staven, NTNU 22 May 2016-----------------------------

#############################
##  Declarations
#############################
use Math::BigFloat;
Math::BigFloat->precision(-8);
############################
##  Check input arguments
#############################
$num_args = $#ARGV+1;
if ($num_args != 3) {
    print "\nUsage: prior_pfile.pl <input pfile> <dict> <output file>\n";
    exit;
}

####################
##  Input arguments
####################
$pfile=$ARGV[0];
$dict=$ARGV[1];
$output=$ARGV[2];
#######################################
##  Get info about number of states/attributes and make a txt version ...
    of pfile
######################################
$num_attr=`cat $dict | wc -l`;

system("pfile_print -i $pfile -o temppfile.txt"); #TODO:uncomment

$pfileinfo=`pfile_info -i $pfile`;
$num_frames = (split / /, $pfileinfo)[2];
#print "$num_frames\n";
##############################
## Initialize all count_array elements to zero. Needed later.####
################################
my @labellist=();
my @logprior=();
##################################
##  Load entire pfile into array
##################################
```

```perl
open(TEXTFILE,"temppfile.txt") or die "ERROR: can not open ...
    temppfile.txt\n";
while($line = <TEXTFILE>) {
    chomp $line;
    #now starts count process for each label
    $curlabel=(split / /, $line)[50];
    #increase at appropriate index
    $labellist[$curlabel]=$labellist[$curlabel] + 1;
}
close(TEXTFILE);
############################################
##  Now find log priors and print to output
############################################
open(OUTFILE, ">$output") or die "ERROR: can not open $output\n";
for($el=0;$el<=$num_attr-1;$el++) {

 if ($labellist[$el]  eq "")
  { $logprior[$el] = -1000; }
 else
 { $logprior[$el] = log($labellist[$el]) - log($num_frames); }
printf OUTFILE "%.5f\n", $logprior[$el];
}
close (OUTFILE);
#########################################
##Check if all probabilities sum to one.
#########################################
open $fh, "<", "$output" or die "Can't open '$output'\n";
my $probsum=Math::BigFloat->new(0.0);
while($line = <$fh>)
{
 chomp $line;
 $probsum=$probsum+exp($line);
}
 print "\n-------------------Sum of all probabilities is: ...
     $probsum------------------------\n";

 close $fh;
 unlink("temppfile.txt");
```

Appendix L

OGI LEDs

English:

```
RE labial b f m p v w
RE coronal d dx en l n s t z er r
RE palatal y jh ch zh sh
RE low aa ae aw ay oy
RE dental dh th
RE mid ah eh ey ow
RE high ih iy uh uw
RE velar g k ng
RE glottal hh
RE sil pau
ME sil sil sil
```

German:

```
RE dental f v
RE palatal y jh ch sh
RE glottal hh
RE labial p b f m w
RE coronal cx d dx l n r s t ts x z
RE low A: a aa ae aw
RE mid ah uh oo ee eh oa ea
RE high iy ih uh uw ihw  ai ia
RE velar k g ng
RE sil pau
ME sil sil sil
```

Hindi:

```
RE palatal y sh jhh jh chh ch
RE dental dt dd ddh dth n
RE glottal hh
RE labial p b bh f m w
RE coronal d dx dh t th s z l r rd
RE low a aa ae aw ai
RE mid ah eh ow e
RE high iy ih uw uh
RE velar k kh g ng
RE sil dtcl ddcl pau
ME sil sil sil
```

Japanese:

```
RE palatal y zh sh jh ch
RE dental d dz t ts s z n
RE glottal hh
RE labial p b f m w
RE coronal  rrr
RE low a ay
RE mid ah ey ow
RE high iy uw
RE velar k g ng
RE sil pau
ME sil sil sil
```

Mandarin:

```
RE palatal  sh ch ts c
RE dental tH
RE glottal hh
RE labial p ph f w m v
RE coronal l n s t r shr tsr tsh
RE low aa ae ao aw ai a
RE mid eh ah ax oe ox er ow ey
RE high iyw iy uw ih u
RE velar k kh ng
RE sil tscl tsrcl chcl pau
ME sil sil sil
```

Spanish:

```
RE palatal ly y jh sh ch
RE dental t d th dh s n l
RE glottal hh
RE labial p b bx f m w
RE coronal hs r
RE low aa ae ay
RE mid ey eh ow ah
RE high iy ih uw uh y
RE velar k g gx hx ng
RE sil pau
ME sil sil sil
```

Appendix M

Norwegian phone mapping

```
2 high
2: high
2y high
2} mid
3: low
4 dental
4_= dental
? glottal
@ mid
@U mid
A low
A: low
Ai low
A}   low
C palatal
D dental
I high
J palatal
J\ palatal
J_= palatal
L palatal
L_= palatal
O mid
O: mid
Oy mid
R coronal
R_= coronal
S palatal
T dental
U high
V low
Z coronal
aU low
b labial
c palatal
c__C palatal
d dental
d__Z palatal
d` dental
e mid
e: mid
eI mid
f labial
h glottal
i high
```

```
i: high
j palatal
l dental
l_= dental
l` palatal
l`_= dental
m labial
m_= labial
n dental
n_= dental
n` palatal
n`_= palatal
p labial
r\ glottal
r\_= glottal
r` dental
s coronal
s_= coronal
t dental
t__S palatal
t` dental
t`_= dental
u high
u: high
ui high
v labial
v_= labial
w labial
x glottal
y high
y: high
z coronal
{ low
{: low
{i low
} high
}: high
}i high
```