Adaptive Boosting for Automatic Speech Recognition

by

Kham Nguyen

Submitted to the Electrical and Computer Engineering Department,
Northeastern University
in partial fulfillment of the requirements for the degree of

Doctor Of Philosophy (PhD)

at the

NORTHEASTERN UNIVERSITY

October 2015

Approved for Dissertation Requirements of the Doctor of Philosophy Degree

Dr. Gilead Tadmor .....
Dissertation Advisor
Professor of Electrical and Computer Engineering

Dr. John Makhoul .....
Dissertation Secondary Advisor
Chief Scientist at Raytheon BBN Technologies

Dr. Jennifer Dy .....
Member, Thesis Committee
Professor of Electrical and Computer Engineering

Spyros Matsoukas .....
Member, Thesis Committee
Principle Scientist at Amazon.com

Dr. Sheila Hemami .....
Department Chair
Professor of Electrical and Computer Engineering

Dr. Sara Wadia-Fascetti .....
Associate Dean of Graduate School
Professor at School of Engineering

Abstract

The atomic units of most automatic speech recognition (ASR) systems are the phonemes. However, the most widely used features in ASR are *perceptual linear prediction* (PLP) and *mel-frequency cepstral coefficients* (MFCC), which do not carry the phoneme information explicitly. The discriminative features with phoneme information have been shown more powerful for ASR accuracy. The process of generating the discriminative features relies on training classifiers to transform the original features to a new probabilistic features.

One of most commonly used techniques for measuring the probabilities in continuous distributions is Gaussian mixture models (GMM). In this work, the GMM-based classifier is used to convert each acoustic feature vector to a posterior probability vector given all classes. Furthermore, an adaptive boosting (AdaBoost) algorithm is applied to combine the classifiers to enhance the performance.

The training of GMM-based AdaBoost classifiers requires very expensive computation. To make it feasible for very large vocabulary speech recognition systems with thousands of hours of training data, we have implemented a hierarchical AdaBoost to split the whole training to multiple parallel processes. The speed up reduced the training data time from about more 100 days to within a week.

The AdaBoost features were then used successfully to combine with spectral feature for ASR. Compared to the baseline of the standard features, the AdaBoost system reduced the word-error-rate (WER) by 2%. Moreover, the AdaBoost system also contributed consistent gains on the system combination even compared with a very strong baseline.

Acknowledgments

First of all, I would like to thank my advisor Dr. John Makhoul for giving me an invaluable opportunity to study and work with the experts in the the field of speech and language processing at BBN technologies. I am so thankful for Dr. Makhoul's kind support and advisement in preparation for my career.

I thank Professor Gilead Tadmor, my academic advisor, for accepting me into the school of engineering and offering valuable advice and counseling me as an international student in the U.S.

I would like to thank Professor Jennifer Dy for her help in writing the thesis proposal. I have acquired great knowledge of machine learning and pattern classification thanks to her class.

I am grateful to Spyros Matsoukas, who always gave feedback on my work at our weekly meetings at BBN. He spent a great deal of time giving me suggestions on my dissertation writing and preparing for my presentations.

I also would like to thank other BBNers with whom I have had the good fortune to work with. They are Tim Ng, Bing Zhang, Stavros Tsakalidis, Richard Schwartz, Eric Wu, Ivan Bulyko, Bing Xiang, and Moonyoung Kang. It was a pleasure to work with and learn from them all.

I also take this opportunity to thank my current colleagues at Nuance Communications: Peter Skala, Ming Yang, Woosung Kim, Scott McClure, and Ernest Pusateri. I have learned more of ASR from real life application side since joining the team.

I would like to thank the anonymous reviewers for their careful reading of the manuscript and their many insightful comments and suggestions to our paper of "Adaptive Boosting for ASR"[53] at ICASSP 2012.

Finally, many thanks to my family, who are always there for me. Especially, I would like to thank my uncle Long Nguyen, my mentor through my school years. He has always been there to help me get through the tough times I faced. Without his guidance, my dream of studying and working in the US would not be possible.

Contents

List of Figures

List of Tables

List of Acronyms

AdaBoost	Adaptive Boosting
AM	acoustic model
ASR	automated speech recognition
BC	broadcast conversation
BIC	Bayesian information criterion
BN	broadcast news
CMLLR	constrained maximum likelihood linear regression
CNC	confusion network combination
EM	expectation maximization
FFT	fast Fourier transform
GMM	Gaussian mixture model
HMM	hidden Markov model
LDA	linear discriminant analysis
LM	language model
LPC	linear predictive coding
MBR	Minimum Bayes' risk
MFCC	mel frequency cepstral coefficients
ML	maximum likelihood
MLLR	maximum likelihood linear regression
MLLT	maximum likelihood linear transform
MLP	multi-layer perceptron
MMI	maximum mutual information
MPE	minimum phoneme error

MPFE	minimum phone-frame error
NX	non-crossword (within word)
PCA	principal component analysis
PDF	probability density function
PLP	perceptual linear prediction
RDLT	region-dependent linear transform
RDT	region-dependent transform
ROVER	recognizer output voting error reduction
SAT	speaker adaptive training
SCTM	state-cluster tied mixture
SD	speaker dependent
SI	speaker independent
STM	state-tied mixture
STT	speech-to-text
WER	word error rate
XW	crossword

Chapter 1

Preface

Statistical automatic speech recognition (ASR) applications have been an interest and challenging research topic over the past decades. The U.S. Department of Defense Advanced Research Projects Agency (DARPA) program has highlighted the importance of this research direction through funding projects awarded to private industry and public institutions such as BBN Technologies, Cambridge University, CMU, IBM, LIMSI, and SRI. All ASR related DARPA projects, moreover, are required to undergo evaluation assessments performed by the National Institute of Standards and Technology (NIST).

Thanks to the technical developments over the years through the DARPA programs, the first successful ASR commercial application termed Dragon was developed in the early 1990s and was used for both personal documents and in the medical industry and legal services to improve productivity.

ASR applications have gained increased attention in recent years with the rising popularity of personal assistants for use in devices such as smart phones or watches (Apple Siri, Google Now, Samsung Svoice, Microsoft Cortana, or Amazon Echo, etc.) In addition, ASR applications from Nuance have also been deployed as virtual assistants for customer services in the banking and airline industries.

ASR applications convert speech signals to text. Like many other machine learning algorithms, an automatic speech recognition system needs to learn from training data. The standard training technique based on maximum likelihood (ML) criterion

attempts to fit the models to the training data without taking into account the recognition error rates. In contrast, discriminative training techniques try to minimize the error rates during the training procedure, which helps them outperform the standard ML training under most testing conditions. There are three main discriminative techniques that focus on models, transforms, and features.

The first technique on discriminatively trained model space includes model parameter estimation based on maximum mutual information (MMI) [2],[84], minimum phone error (MPE) [64], and other relevant criteria. These training approaches optimize the parameters to reduce the training error using the confusable hypotheses to teach the system to differentiate the correct choice from the close ones.

The second technique to train discriminately trained transforms includes frame minimum phone error (fMPE) [63] and region dependent transform (RDT) [89]. The general idea of this technique is to estimate linear feature transforms on discriminative criteria. The original features are transformed based on certain conditions. The transformed features are then used as inputs for the ASR systems.

The third technique trains discriminatively features. The traditional acoustic features such as perceptual linear prediction (PLP) and mel frequency cepstral coefficients (MFCC) widely used in most ASR systems; however they do not carry the phoneme discriminative information explicitly. The approach, to date, that has been used effectively to train those discriminative features is based on multilayer perceptron (MLP) [14]. The MLP builds a single neural network to estimate the posterior phoneme probability at frame level. The layers of the network are then connected by links with weights. The parameters of the network are updated adaptively based on the inputs and outputs flowing through the network. For very large vocabulary automatic speech recognition systems, training a single network with thousands of hours of acoustic data concurrently could take long time to complete. The discriminative features are used to replace or combine with the standard features for training.

In this work, we explore the use of adaptive boosting (AdaBoost) [22] for Gaussian

mixture classifiers to train discriminatively posterior phoneme probabilistic features. With thousands of hours of audio, the entire training process is really computational expensive. A parallel approach has been used to make the training feasible.

The rest of the thesis is organized as follows. In Chapter 2, an overview of automatic speech recognition is presented. Chapter 3 discusses the usage and limitation of standard features in ASR, and proposes the discriminative features as the replacement. Chapter 4 introduces to AdaBoost, and Chapter 5 continues with AdaBoost to combine Gaussian mixture classifiers. AdaBoost training experiments results are shown in Chapter 6. Chapter 7 presents the usage of AdaBoost features in ASR. Chapter 8 then concluses the thesis and provides suggestions for future AdaBoost studies in ASR.

Chapter 2

Automatic Speech Recognition

2.1 Introduction

Automatic Speech Recognition (ASR) [67] is a process of converting an acoustic signal into a sequence of words. The recognized words can be the outcome for applications such as command and control, data entry, and document preparation. They can also serve as input to further linguistic processing, for example, speech understanding or machine translation.

Speech recognition systems generally assume that the speech signal is a realization of some message encoded as a sequence of one or more symbols. The continuous speech waveform is first converted to a sequence of equally spaced discrete parameter vectors. The typical parametric representations commonly used are smoothed spectra or linear prediction coefficients, plus various other representations that are derived from these. Given the sequence of speech observation vectors $O = o_1, o_2, ..., o_n$, the goal of the decoder is to find the best word sequence W^* that *maximizes a posterior* (MAP) the probability:

$$W^* = arg \max_i \{P(W_i|O)\} \tag{2.1}$$

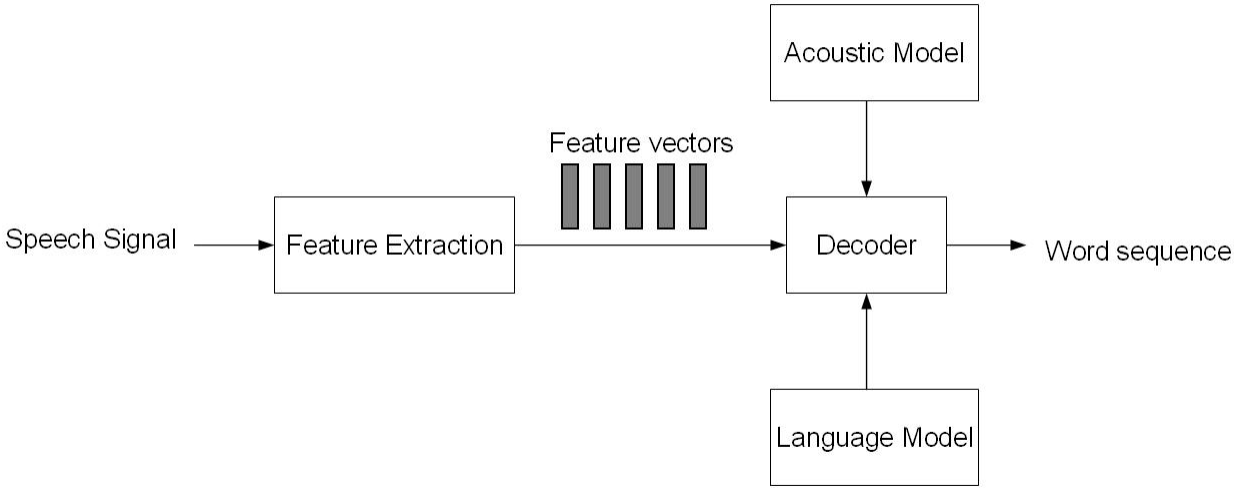

Figure 2-1: Architecture of an ASR System: Feature extraction, AM, and LM

where W_i is the i^{th} word sequence.

This probability is not able to compute directly, but using Bayes Rule it gives

$$P(W_i|O) = \frac{P(O|W_i)P(W_i)}{P(O)}. \tag{2.2}$$

Because the acoustic signal is common regardless of which word sequence is chosen, Eq. (2.1) is equivalent to the following:

$$W^* = arg\max_i\{P(O|W_i)P(W_i)\}. \tag{2.3}$$

The term $P(O|W_i)$ is generally referred to as the *acoustic model* (AM). Similarly, $P(W_i)$ is referred to as the *language model* (LM).

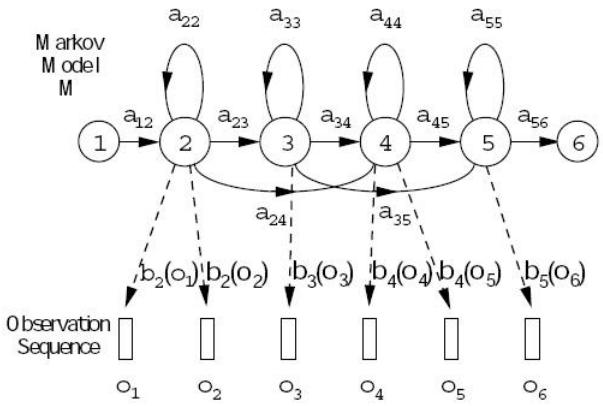

Figure 2-2: A sample of HMM Framework

2.2 Acoustic Model

The most widely used *acoustic model* is based on the *hidden Markov model* (HMM) [67]. In an HMM-based speech recognition system, it is assumed that the sequence of observed speech vectors corresponding to each word is generated by a Markov model. A Markov model is a finite state machine, which changes its current state to another one for every time unit. At each time t that a state j is entered, a speech vector o_t is generated with the probability density $b_j(o_t)$. Furthermore, the transition from state i to state j is also probabilistic and is governed by the discrete probability a_{ij}. For example, in Figure 2-2, the joint probability that O is generated by model λ moving through the state sequence S is calculated simply as the product of the transition and output probabilities:

$$P(O, X|\lambda) = a_{12}b_2(o_1)a_{22}b_2(o_2)a_{23}b_3(o_3)\cdots \tag{2.4}$$

However, in practice, only the observation sequence O is known and the underlying state sequence X is hidden, which is why it is called a hidden Markov model (HMM). Given that X is unknown, the required likelihood is computed by summing over all

possible state sequences $X = x_1, x_2, x_3, \cdots x_T$, that is:

$$P(O|\lambda) = \sum_{X} a_{x_0 x_1} \prod_{t=1}^{T} b_{x_t}(o_t) a_{x_t x_{t+1}} \tag{2.5}$$

where x_0 is constrained to be the model entry state and x_{T+1} is constrained to be the model exit state. The observation probability $b_j(o_t)$ for an observation o_t at state j is modeled by Gaussian mixtures as

$$b_j(o_t) = \sum_{k} m_{jk} \mathcal{N}(o_t|\mu_{jk}, \Sigma_{jk}), \tag{2.6}$$

for the k_{th} component of the state j^{th}. m_{jk} is mixture weight, and $\mathcal{N}(o_t|\mu_{jk}, \Sigma_{jk})$ is a Gaussian density with means μ_{jk}, and co-variance matrix Σ_{jk}:

$$\mathcal{N}(o_t) = \frac{1}{\sqrt{(2\pi)^n |\Sigma_{jk}|}} e^{-\frac{1}{2}(o_t - \mu_{jk})^T {\Sigma_{jk}}^{-1} (o_t - \mu_{jk})}. \tag{2.7}$$

An HMM-based acoustic model is trained and evaluated using three basic HMM problems [67]. The first problem computes the probability of the observation sequence given the model parameters. The second problem finds the "optimal" state sequence associated with the given observation sequence. The third problem trains the model parameters used in the first and second problems.

2.2.1 The First HMM Problem - Probability Evaluation

Given the observation sequence $O = \{o_1, o_2, o_3..\}$ and an HMM model $\lambda = \{A, B, \pi\}$, the likelihood is estimated for all possible state sequences using a form:

$$P(O|\lambda) = \sum_{\forall S} P(O|S, \lambda) P(S|\lambda). \tag{2.8}$$

For a particular state sequence $S = \{s_1, s_2, s_3, ..\}$, the emission probability is

$$P(O|S, \lambda) = \prod_{t=1}^{T} P(o_t|s_t, \lambda) = \prod_{t=1}^{T} b_{s_t}(o_t). \tag{2.9}$$

And the transition probability is computed as

$$P(S|\lambda) = \pi_{s_1} a_{s_1 s_2} a_{s_2 s_3} .. a_{s_T s_T}. \tag{2.10}$$

Merging the above results, we have:

$$P(O|\lambda) = \sum_{s_1, s_2, .., s_T} \pi_{s_1} b_{s_1}(o_1) a_{s_1 s_2} b_{s_2}(o_2) .. a_{s_2 s_3} b_{s_T}(o_T). \tag{2.11}$$

Computing this form of likelihood directly is not feasible, because there are N^T possible state sequences, where N is the number of states in an HMM and T is the number of observations. The forward and backward procedures are used for the likelihood evaluation. By using the cumulative variable to keep track of the observation sequence up to time t and with the current state of i

$$\alpha_t(i) = P(o_1, o_2, .., o_t, s_t = i|\lambda). \tag{2.12}$$

Based on this, the computational complexity is significantly reduced - from N^T to just N^2T by using the forward procedure.

Forward Procedure	
1. Initialization:	$\alpha_1(i) = \pi_i b_i(o_1)$
2. Induction:	$\alpha_{t+1}(j) = [\sum_{i=1}^{N} \alpha_t(i) a_{ij}] b_j(o_{t+1})$
3. Termination:	$P(O\|\lambda) = \sum_{i=1}^{N} \alpha_T(i)$

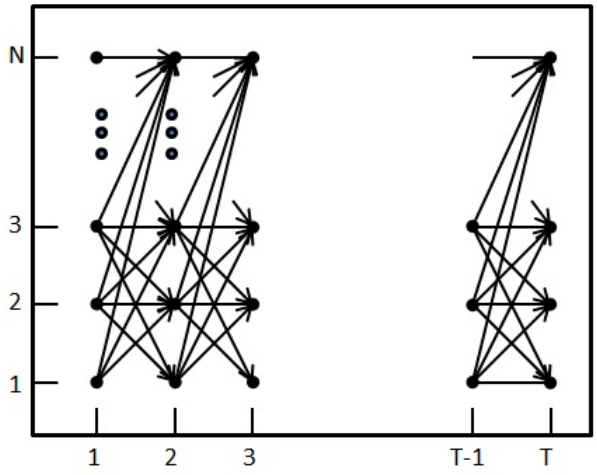

Figure 2-3: Lattices (Reproduced from Rabiner, 1989)

2.2.2 Second HMM Problem - Decoding

The second problem of HMM is to find the optimal sequence of states, which could be either the sequence of most likely states at each time t or the states with highest overall likelihood $P(O|S,\lambda)$. The first solution to choose the sequence of most individual states has a problem of "zero" likelihood when one of the transition probabilities a_{ij} is not available or equal to 0. The second solution is more widely used and called the Viterbi algorithm. Instead of finding the only most likely states, the Viterbi algorithm keeps all the "best" paths, which end at all N possible states at any time t. One variable is used to keep those scores

$$\delta_t(i) = \max_{q_1 q_2 .. q_{t-1}} P(q_1 q_2 .. q_t = S_i, o_1 o_2 .. o_t|\lambda). \tag{2.13}$$

The sequences will be extended by induction:

$$\delta_{t+1}(j) = \max_j [\delta_t(i) a_{ij}] b_j(o_{t+1}). \tag{2.14}$$

To trace back the optimal sequence, the algorithm keeps track of the state that maximizes $\delta_t(i)$ at each time t by using an array

$$\Psi_{t+1}(j) = \arg \max_{1 \leq i \leq N} [\delta_t(i) a_{ij}]. \tag{2.15}$$

2.2.3 Third HMM Problem - Parameter Estimation

The third HMM problem is the most challenging because it requires determining a method for adjusting the model parameters a_{ij} and $b_{x_t}(o_t)$ in order to satisfy an optimization criterion. There is no known way to find the model parameter set that maximizes the probability of the observation sequence in a closed form. However, we can choose a_{ij} and $b_{x_t}(o_t)$ so that the likelihood $P(O|\lambda)$ is maximized by using an iterative procedure such as the Baum-Welch method.

To describe the procedure for re-estimation (iterative update and improvement) of HMM parameters, we first define $\xi_t(i,j)$, the probability of being in state i at time t, and the state j at time $t+1$, given the model and the observation sequence:

$$\begin{aligned}
\xi_t(i,j) &= P(q_t = i, q_{t+1} = j | O, \lambda) && (2.16)\\
&= \frac{P(q_t = i, q_{t+1} = j, O|\lambda)}{P(O|\lambda)} && (2.17)\\
&= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{P(O|\lambda)} && (2.18)\\
&= \frac{\alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} b_j(o_{t+1}) \beta_{t+1}(j)} && (2.19)
\end{aligned}$$

where the probability of the partial observation sequence $o_1, o_2, ..., o_t$ (until time t), and state i at time t given the model λ is

$$\alpha_t(i) = P(o_1, o_2, ..., o_t, q_t = i | \lambda). \tag{2.20}$$

In a similar way,

$$\beta_t(i) = P(o_{t+1}, o_{t+2}, ..., o_T | q_t = i, \lambda). \tag{2.21}$$

Let $\gamma_t(i)$ be the probability of being in state i at time t, given the entire observation sequence and the model. Hence, we can relate $\gamma_t(i)$ and $\xi_t(i,j)$ by summing over j:

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j). \tag{2.22}$$

If we sum $\gamma_t(i)$ over the time index t, we get a quantity that can be interpreted as the expected (over time) number of times that state i is visited, or equivalently, the expected number of transitions made from state i (if we exclude the time slot $t = T$ from the summation). Similarly, the summation of $\xi_t(i,j)$ over t (from $t = 1$ to $t = T - 1$) can be interpreted as the expected number of transitions from state i to state j. From the model λ, we can re-estimate the parameters for λ' by

$$\overline{a_{ij}} = \frac{\text{expected number of transitions from state } i \text{ to } j}{\text{expected number of transitions from state } i} \tag{2.23}$$

$$= \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}, \tag{2.24}$$

$$\overline{b_j(v_k)} = \frac{\text{expected number of times in state } j \text{ and observing symbol } v_k}{\text{expected number of times in state } j} \tag{2.25}$$

$$= \frac{\sum_{t=1, s.t. o_t = v_k}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \gamma_t(i)}. \tag{2.26}$$

If we define the current model as $\lambda = (a_{ij}, b_j(o_t))$ and use that to compute the $\overline{a_{ij}}$ and $\overline{b_j(v_k)}$, and we define the re-estimated model as $\overline{\lambda} = (\overline{a_{ij}}, \overline{b_j(o_t)})$, the model $\overline{\lambda}$ is more likely than model λ in the sense that $P(O|\overline{\lambda}) > P(O|\lambda)$. That is, we have found a new model $\overline{\lambda}$ from which the more likely observation sequence has been produced.

By re-estimation, we can improve the probability of O being observed from the model until the convergence condition is reached.

2.3 Acoustic Model Discriminative Training

Compared to the maximum likelihood (ML) approaches, such as the Baum-Welch algorithm stated earlier, the discriminative approaches such as maximum mutual information (MMI) or minimum phoneme error (MPE) lead to better recognition results. Instead of trying to fit the data best, the discriminative approaches learn to separate the correct answers from the confusable ones. To train the discriminative models, a set of confusable hypotheses (lattices) are generated by an ML-based acoustic model and a weak language model.

2.3.1 Maximum Mutual Information Estimation

Maximum mutual information (MMI) estimation was proposed by Bahl [2]. The objective function of MMI is to maximize the mutual information between the observation hypothesis and the reference:

$$I(O, W_{ref}) = \frac{p(O, W_{ref})}{p(O)p(W_{ref})}. \tag{2.27}$$

The probabilities $p(W_{ref})$ do not affect the acoustic model optimization, so the MMI training goal is to estimate the parameters that maximize the overall conditional likelihood given the observations:

$$\begin{aligned} \lambda^*_{MMI} &= arg\max_{\lambda} \sum_{r=1}^{R} \frac{p_\lambda(O_r, W_r^{ref})}{p_\lambda(O_r)} \\ &= arg\max_{\lambda} \frac{p_\lambda(O_r|W_r^{ref})p(W_r^{ref})}{\sum_{W_i} p_\lambda(O|W_i)p(W_i)}. \end{aligned} \tag{2.28}$$

The numerator is the likelihood of the observations O_r given the correct word sequence W_r^{ref}. And, the denominator is the total likelihood of the observations given all possible word sequences W_i. Maximizing the objective function makes the correct model sequence likely and all other confusable sequences unlikely.

2.3.2 Minimum Phoneme Error Estimation

Another discriminative model training approach is minimum phoneme error (MPE) estimation [63]. The MPE objective function is to maximize the average phoneme accuracy of the hypothesis given the training reference,

$$\lambda_{MPE} = arg \max_{\lambda} \sum_{r=1}^{R} \frac{\sum_{W_i} p(O|W_i, \lambda) p(W_i) \epsilon(W_i, W_{ref})}{\sum_{W_i} p(O|W_i, \lambda) p(W_i)} \tag{2.29}$$

where $\epsilon(W_h, W_{ref})$ is a measure of the phoneme accuracy in each word string hypothesis W_i. Given the times of the phoneme boundaries, each phoneme in W_i is matched against the time in the reference W_{ref}, and $\epsilon(W_h, W_{ref})$ is a percentage overlap between these two word sequences.

2.4 Acoustic Model Adaptation

The mismatch in ASR system occurs during testing when there are new speakers or speakers with too little data used in training. Since most acoustic models are trained statistically, the mismatch will cause the recognition performance to degrade. Adaptation techniques have been used to update the models and reduce recognition error rate. The most widely used approach is called unsupervised adaptation where the hypotheses from an initial decoding pass are used to update models and closely match them to the test speakers. The updated models are then used to re-decode the test data. Even though the first pass hypotheses contain errors, the unsupervised adaptation decoding usually yields to a better performance than those in the first

pass. Because the adaptation uses the data to update the existing models, it does not require significant amounts of data.

2.4.1 Maximum A Posteriori Adaptation

Maximum a posteriori (MAP) adaptation [39] is a Bayesian based approach. Even if there is limited data that include the recognition errors in the first pass, MAP adaptation can still reduce the mismatch effectively. And based on speaker data during testing, the new adapted models are updated based on speaker independent (SI) model parameters. In this case, the SI models are used as a prior probability distribution over the model parameters. Instead of re-training the models on maximum likelihood (ML) method to maximize $p(X|\lambda)$, MAP adaptation maximizes a posteriori with respect to the speaker data

$$p(\lambda|X) \propto p(X|\lambda)p_0(\lambda). \tag{2.30}$$

Since the adapted models rely on the prior models, MAP adaptation does not require a large amount of speaker data. It uses a parameter τ to control how much it should trust the prior model and the speaker data.

$$\lambda^* = \arg\max_{\lambda} p(X|\lambda)^{\tau} p_0(\lambda) \tag{2.31}$$

Assume that the SI mean is μ_{i0} and the mixture weight is m_i, then the adapted mean $\hat{\mu_i}$ is estimated by

$$\hat{\mu_i} = \frac{\tau\mu_{i0} + \sum_n m_i x_n}{\tau + \sum_n m_i} \tag{2.32}$$

When τ is small, the effect of the SI model is smaller, and the speaker-specific observations $\{x_n\}$ dominate in the adapted mean. If the number of observations of the new speaker increases, the MAP adaptation becomes closed to the ML training on the new data because the new data then dominate over the old mean. In practice,

only the means are adjusted in the adaptation process, while the variances are kept unchanged. A typical value of τ is between 2 and 20. The main drawback of MAP adaptation is that the Gaussian components are updated locally. With a limited amount of speaker data a small number of components are updated, while the rest of them remain as the SI models.

2.4.2 Maximum Likelihood Linear Regression Adaptation

The Maximum Likelihood Linear Regression (MLLR) adaptation trains the transformation to map the model parameters (means and/or variances) so that the likelihood of the speaker adaptation data is maximized. Instead of adapting the model locally like in MAP, the MLLR adaptation converts all Gaussian components. Given the SI model $\lambda = \{\mu, \Sigma\}$, the MLLR adaptation transforms have the following forms:

$$\hat{\mu_i} = G\mu_i + b, \tag{2.33}$$

$$\hat{\Sigma_i} = H\Sigma_i H^T. \tag{2.34}$$

There are two versions of MLLR: unconstrained and constrained, where the second one, CMLLR, has $H = G$. The objective function of MLLR is to maximize the likelihood given the speaker data set $O_{new} = \{x_n\}$

$$\hat{\lambda} = \arg\max L(\lambda|O_{new}). \tag{2.35}$$

In practice, the MLLR adaptation is trained iteratively using the expectation-maximization (EM) technique, which will be reviewed in chapter 5. The number of transform depends on the amount of adaptation data. With a small amount of data, the MLLR will use only one global transform for the whole acoustic model. However, the number of transforms will increase when more data are available. Similar GMMs can share the same transform. To find the number of transforms, a regression class tree is used.

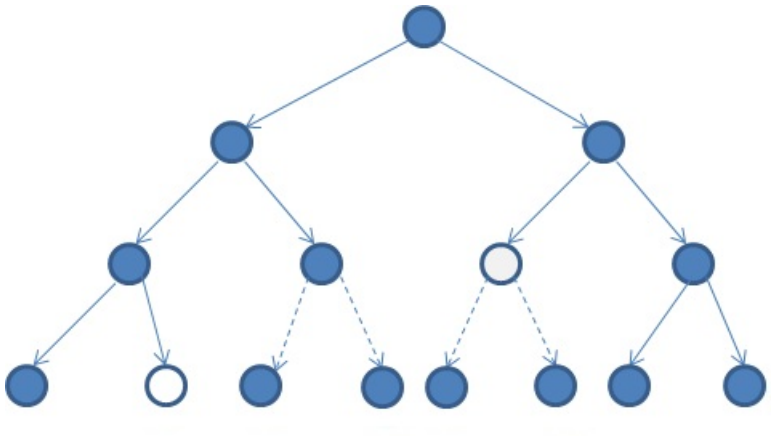

Figure 2-4: MLLR - A regression class tree is expanded if more data is available

Similar to MAP adaptation, the hypotheses from first SI decoding pass are used as the data for adaptation. Even though there are still errors in the hypotheses, the adaptation still offers good gain in most cases since it helps to reduce the mismatch between the training and testing conditions. Furthermore, MLLR and MAP can be combined to arrive at a better result compared to a single technique.

2.5 Language Model

In addition to the acoustic model, the *language model* [35] plays a very important role in any statistical ASR system. In theory, we need to estimate the probability $P(w_1, w_2, ..., w_n)$ for all possible $w_1, w_2, ..., w_n$

$$P(w_1, w_2, ..., w_n) = \prod_{i=1}^{n} P(w_i|w_1, w_2, ..., w_{i-1}). \tag{2.36}$$

The number of probabilities to be estimated grows exponentially as the history length and number of unique words grows. In order to reduce the number of probabilities to be estimated, one solution is the use of n-gram, which assumes that only $(n-1)$ words in history are relevant.

2.5.1 Language Model Training

Given the counts from the training data set, the *maximum likelihood* (ML) estimate will maximize the likelihood function. To derive the estimate, we assume that there are s different n-grams $\alpha_1, ..., \alpha_s$, with the counts of $c_1, ..., c_s$, and $N = \sum_i c_i$. Let $p_1, ..., p_s$ be the true probabilities that we need to estimate. For *independent and identically distributed* (iid) samples, the total likelihood function to be optimized has the form

$$\begin{aligned} L(p_1, ..., p_s|\alpha_1, ..., \alpha_s) &= Pr(\alpha_1, ..., \alpha_s|p_1, ..., p_s) \\ &= \prod_{i=1}^{s} Pr(\alpha_i|p_i) \\ &= \prod_{i=1}^{s} p_i^{c_i}(1-p_i)^{N-c_i}. \end{aligned} \tag{2.37}$$

Taking the partial derivatives of the likelihood function with respect to all parameters p_i, and setting them to 0, we have the ML estimate of the probabilities given the counts:

$$p_i = \frac{c_i}{N}. \tag{2.38}$$

The major problem with the ML method is that when an n-gram does not occur in the training data, its LM score is zero, which makes the total score zero regardless of how big the AM score is. The zero probability issue for LM is solved by using smoothing techniques. Central to many smoothing techniques is the Good-Turing estimate [28], [6]. Instead of using the real count r of any n-gram, we use a "discounting" count r^*:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}, \tag{2.39}$$

and the probability is computed by using this discounting count as

$$p(\alpha) = \frac{r^*}{N} \tag{2.40}$$

where n_r is the number of n-grams that occurs exactly r times in the training data. The total count does not change compared to the true count distribution:

$$N = \sum_{r=0}^{\infty} n_r r^* = \sum_{r=0}^{\infty} n_r (r+1) \frac{n_{r+1}}{n_r} = \sum_{r=1}^{\infty} r n_r. \tag{2.41}$$

We made a similar assumption as when we derived the ML estimate above. We want to estimate the true probability of an n-gram α_i that occurs r times. Even though we do not know the identity of the n-gram α_i, we know that it is generated by one of the candidates $p_1, ..., p_s$. This probability is the expected value:

$$E(p_i|c(\alpha_i) = r) = \sum_{j=1}^{s} p(i = j|c(\alpha_i) = r) p_j. \tag{2.42}$$

The probability $p(i = j|c(\alpha_i) = r)$ is the probability that an unknown n-gram α_i with r counts is actually the jth n-gram α_j. We can rewrite it as

$$\begin{aligned} p(i = j|c(\alpha_i) = r) &= \frac{p(c(\alpha_j) = r)}{\sum_{j=1}^{s} p(c(\alpha_j) = r)} \\ &= \frac{C_r^N p_j^r (1 - p_j)^(N - r)}{\sum_{j=1}^{s} C_r^N p_j^r (1 - p_j)^{N-r}} \\ &= \frac{p_j^r (1 - p_j)^(N - r)}{\sum_{j=1}^{s} p_j^r (1 - p_j)^{N-r}}. \end{aligned} \tag{2.43}$$

Substituting this into Eq. (2.42), we get:

$$E(p_i|c(\alpha_i) = r) = \frac{\sum_{j=1}^{s} p_j^{r+1} (1 - p_j)^{N-r}}{\sum_{j=1}^{s} p_j^r (1 - p_j)^{N-r}}. \tag{2.44}$$

Now, consider $E_N(n_r)$, the expected number of n-grams with exactly r counts, given that there are a total of N counts. This is equal to the sum of the probability that each n-gram has exactly r counts:

$$E_N(n_r) = \sum_{j=1}^{s} (c(\alpha_i) = r) = \sum_{j=1}^{s} C_r^N p_j^r (1 - p_j)^{N-r}. \tag{2.45}$$

Substituting this equation into (2.44) yields

$$E(p_i|c(\alpha_i) = r) = \frac{r+1}{N+1}\frac{E_{N+1}(n_{r+1})}{E_N(n_r)}. \tag{2.46}$$

This is an estimate for the expected probability of an n-gram α_i with r counts. From Eq. (2.26), the discounting count r^* is estimated by

$$r^* = Np(\alpha_i) = N\frac{r+1}{N+1}\frac{E_{N+1}(n_{r+1})}{E_N(n_r)} \approx (r+1)\frac{n_{r+1}}{n_r}. \tag{2.47}$$

The Good-Turing estimate was extended by Katz in [35] by adding the combination of higher-order models with lower-order models. To describe Katz smoothing for the order combination, we use bi-gram models that mean the n-gram with only one word in the history is considered. For a bi-gram (w_{i-1}, w_i) with count $r = c(w_{i-1}, w_i)$, we estimate the discounting counts by using the formula

$$c(w_{i-1}, w_i) = \begin{cases} r^* & \text{if } r > 0 \\ \gamma(w_{i-1})p(w_i) & \text{if } r = 0. \end{cases}$$

The back-off value γ is estimated so that the total probability for the whole distribution under Katz smoothing still satisfies the probability distribution axiom:

$$\gamma(w_{i-1}) = \frac{1 - \sum_{w_i:r>0} p(w_i|w_{i-1})}{\sum_{w_i:r=0} p(w_i)} = \frac{1 - \sum_{w_i:r>0} p(w_i|w_{i-1})}{1 - \sum_{w_i:r>0} p(w_i)}. \tag{2.48}$$

When $n = 2$ (i.e., $w_i|w_{i-1}$), we call them bi-grams. Similarly, tri-grams are $(w_i|w_{i-1}, w_{i-2})$. During decoding, the tri-gram probability of the sequence (w_{i-2}, w_{i-1}, w_i) is obtained as follows:

$$p(w_{i-2}, w_{i-1}, w_i) = \begin{cases} p(w_i|w_{i-2}, w_{i-1}) & \text{if the tri-gram exists,} \\ \gamma(w_{i-2}, w_{i-1}) * p(w_i|w_{i-1}) & \text{if the bigram exists,} \\ \gamma(w_{i-2}, w_{i-1}) * \gamma(w_{i-1}) * p(w_i) & \text{otherwise.} \end{cases}$$

2.5.2 Interpolation Multiple LMs

Another approach to smooth the LM is to interpolate multiple sub-models, which are trained from different data sources. Building different models from different sources helps to reserve their n-gram distributions and avoid situation where the model is dominated by the large data corpora. The sub-models are then linear interpolated (LI) with the optimized weights. The interpolated model score has a form of

$$p^{LI}(w_i|w_{i-n+1}^{i-1}) = \sum_{k=1}^{K} \lambda_k P_k(w_i|w_{i-n+1}^{i-1}) \tag{2.49}$$

where λ_k is the interpolation weight for the k^{th} component. To ensure the whole n-gram distribution summing to 1, the weights must satisfy the condition

$$\sum_{k=1}^{K} \lambda_k = 1. \tag{2.50}$$

To estimate the interpolation weights, the EM algorithm has been used [8]:

1. Initialization

$$\lambda_k = \frac{1}{K} \tag{2.51}$$

2. Estimation-step: re-estimate the weights based on the previous weights and probabilities

$$\hat{\lambda}_j^t = \sum_{w_i} \frac{\lambda_j^t P_j(w_i|w_{i-n+1}^{i-1})}{\sum_{k=1}^{K} \lambda_k^t P_k(w_i|w_{i-n+1}^{i-1})} \tag{2.52}$$

3. Maximization-step: The weights are normalized to satisfy the distribution probability theorem

$$\lambda_j^{t+1} = \frac{\hat{\lambda}_j^t}{\sum_{k=1}^{K} \hat{\lambda}_k^t} \tag{2.53}$$

To check for convergence criterion, the total log-likelihood of the data of the current iteration is compared to the previous one:

$$l(\lambda^t) - l(\lambda^{t+1}) < \epsilon \tag{2.54}$$

where the iteration log-likelihood is defined as

$$l(\lambda^t) = \sum_{w_i} \log \sum_{k=1}^{K} \lambda_k^t P_k(w_i | w_{i-n+1}^{i-1}). \tag{2.55}$$

2.6 Decoding Process

Given a sequence of feature vectors, the decoding process combines the AM and LM scores to find the most likely sequence of words. For the early step, the model with fewer parameters is used to make the system more robust, and for the later step, the model has more parameters to achieve higher discriminative abilities. In the BBN Byblos system [55], there are a total of six AM models: three for speaker independent, and another three for speaker dependent. The LM complexities vary by the n-gram orders. The recognition process includes three passes: forward decoding, backward decoding, and lattice re-scoring.

2.6.1 Forward Decoding Pass

The first pass uses a bi-gram language model (a bi-gram $w_i|w_{i-1}$, for current word w_i with one word for history w_{i-1}), and within word context, a tri-phone state tied mixture (STM) HMM to process a fast-match search. The tri-phone state (for example, a[b]c), given the same center phoneme and state, shares the same GMM components (512 bins on average), and the mixture weights are also shared based on the decision tree clustering. Fast-match forward pass records the word ending times and scores. The outputs of this stage are the most likely word ends per frame and their likelihood scores. $\alpha(w, t)$ is the probability of the speech from the beginning of the utterance up to time t with word w times the LM score of that word sequence.

2.6.2 Backward Decoding Pass

The outputs from forward pass are used to restrict the search space to make the decoding in the backward pass less expensive with more detailed models. The approximate tri-gram language model (a tri-gram $w_i|w_{i-1}w_{i-2}$, one word with two words for history) is used in the pass. The within-word quinphone (for example, a,b[c]d,e) state clustered tied mixture (SCTM) is used similar to the STM model.

Backward pass records the word beginning times, scores, and the path history for N-best generation later. $\beta(w_2, t)$ is the probability of speech from the end of the utterance back to time t, given the likely word sequence ending with word w_2 times the LM score of that word sequence.

If $\alpha(w_3, t)\beta(w_2, t)Pr(w_3|w_1, w_2) > threshold$, w_3 will be active, where w_1 is the best preceding word of w_2. After matching back to the beginning of the utterance, from the saved word times and the path history, the decoder generates the tri-gram word lattice.

2.6.3 Lattice Re-scoring and Post Processing Passes

The final pass re-scores the lattices from the backward pass using cross-word quinphone acoustic models (SCTM crossword) and tri-gram language models. The outputs of this pass are the top N hypotheses. In the beam search steps, the decoder saves the computation and storage by using the approximate tri-grams. Given the current w_2, the tri-gram $p(w_3|w_1, w_2)$ can be computed by using only the best w_1 history that w_2 used. This sub-optimal tri-gram from the second pass is replaced with the optimal score in the word lattice construction step, after the word boundaries have been identified.

The N-best hypotheses from the lattice re-scoring pass are re-ranked to select the best hypothesis. Only language models such as higher order n-gram or neural network based models are used in this pass to score again the N-best hypotheses.

2.7 System Combination

In the Viterbi framework, the decoding systems find most likely state sequence through the HMM by combining acoustic and language scores. It is equivalent to finding the most likely sentence or optimizing the sentence error rate (SER), while the word error rate (WER) is the most used evaluation metric. This is a sub-optimal performance issue. The minimum Bayes' risk (MBR) [86] decoding integrates the evaluation metric into the optimization function so that the hypothesis sequence is optimized on WER.

$$\hat{W} = \arg\min_{i} P(O|W_i)L(W_i, W). \tag{2.56}$$

The loss function, $L(W_i, W)$, is trained to optimize the WER criterion or any other metric. To reduce the search space, MBR decoding only considers the word candidates from the lattices generated by the baseline models. The lattices are also compressed more through a merging of the starting and end points of the words within a time window. The MBR decoding was originally used to apply on the N-best lists to improve the WER for one system. However, as long as the value of the posterior $P(O|W_i)$ and the loss function $L(W_i, W)$ can be computed and scaled properly, the MBR approach can be used to combine multiple systems. System combination is actually used in most state-of-the-art ASR systems. The two most popular word-level combination approaches are ROVER (Recognition Output Voting Error Reduction) and CNC (Confusion Network Combination), which also belong to the MRB framework.

2.7.1 ROVER Combination

Recognizer output voting error reduction (ROVER)[17], developed at NIST, is a voting method for combining the ASR systems. The 1-best hypothesis from the individual systems estimate the word level confidence scores by using decoding features such as acoustic and language model scores, posterior probabilities, etc. One of the system's hypothesis is used to align other systems' ones based on Levenshtein, or

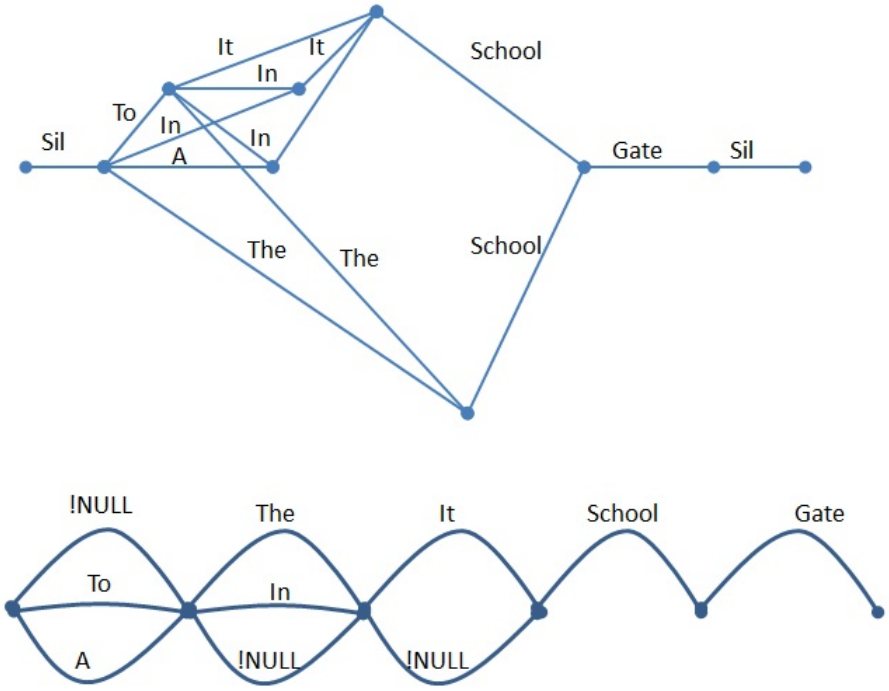

Figure 2-5: Confusion network generated from lattices

edit distance, algorithm, which is also used to measure WER for ASR system. There are multiple ways to align the transcriptions, and the optimal algorithm based on dynamic programming aims to minimize the total number of edits, including substitution, deletion, and insertion. In the alignment sample below, there are 3 edits, two substitutions and one insertion. The output from the alignment step is word transi-

S_1:	This	isn't	!NULL	right
S_2:	It	is	not	right
Result:	Sub	Sub	Ins	Cor

The Levenshtein alignments: S_1 and S_2

tion network (WTN); and ROVER combination chooses the word by voting with or without the confident scores. One possible voting form is to tune the parameter β on another tuning set, and then use it on the test data. This represents a trade-off between the majority voting and unweighted posterior combination:

$$\hat{W} = \arg\max_{W} \beta \frac{count(W)}{S} + (1-\beta)P(W). \tag{2.57}$$

In the case of two systems, the ROVER has to pick the word with higher confident score when the two systems disagree, and that leads to an issue if the scores from two systems are not scaled reasonably. The ROVER combination works better with more than two individual systems because the majority vote will reduce the risk of unreliable scores.

2.7.2 Confusion Network Combination

While the ROVER algorithm uses only the 1-best hypothesis from each system to combine, the confusion network combination (CNC) [16][42] uses the lattices to generate the confusion networks with posterior probabilities. Those confusion networks are then merged into a single confusion network based on the time stamps. The posterior probabilities are re-normalized for each segment. Each system has its own

scale factor λ_s that is optimized on a tuning set to achieve the minimal total WER. The word level voting scheme is based on

$$\hat{W} = \arg\max_{W} \sum_{s=1}^{S} \lambda_s P(W|s). \tag{2.58}$$

In general, CNC seems to be more robust with a small number of systems than ROVER due to the use of the lattices instead of the 1-best only. The CNC also suffers from the unreliable confidence scores from the systems. Both CNC and ROVER operate better when the best system in terms of WER is used to align other systems to generate the merged CNC or WTN. The experiments have shown that the two system combination approaches work well when the individual systems have comparable performances, but they are not too similar.

2.8 Summary

In this chapter, we presented an overview of an automatic speech recognition (ASR) system. We have shown how an ASR system uses its main components, acoustic and language models in the recognition process. The acoustic model (AM) is based on the hidden Markov model (HMM) framework, which is trained on the maximum likelihood (ML) criterion. The discriminative approaches such as MMI or MPE usually outperform the ML. We have presented the adaptation techniques such as MAP or MLLR to reduce the difference of the training and testing condition. The chapter has also reviewed the language model training to estimate the probability of a word, given a history of a sequence of words. To make a language model more robust, multiple models are trained and interpolated, which helps to reserve the characteristic of the data sources. The models are then merged based on the optimized weights estimated from a held-out set. In addition to the regular word-based n-grams, the language models such as class-based or neural network-based models are used to interpolate. By using the AM and LM, an ASR system usually processes two-stage decoding pro-

cess. The first stage uses the speaker independent models, and the second stage uses the hypotheses from the first stage to adapt the models and re-run the recognition again. Each stage decoding has three passes. The passes start with the simple models to reduce the search space for the later ones with more complex models. To improve the recognition performance, the results from multiple systems are combined using ROVER or CNC techniques.

Chapter 3

Feature Extraction

3.1 Introduction

The traditional spectral acoustic features are widely used in most state-of-the-art ASR systems. To generate those features, the spectrum of speech waveform is sliced and encoded to capture the most information and reduce the very high dimensionality of the raw speech signal. Despite their wide usage, the traditional features do not carry the phoneme information explicitly, which could cause recognition errors especially for the first pass with weaker acoustic and language models. The errors could be fixed by re-scoring steps with stronger models in the later passes, but some errors will not be recovered. In this chapter, we review traditional feature extraction as well as propose the discriminative feature for ASR system.

3.2 Standard Feature Extraction

The feature extraction process transforms a speech waveform into a sequence of parameter vectors. This sequence is assumed to form an exact representation of the speech waveform (typically 10 ms per frame, 25 ms window size). The two most common types of speech features are *mel frequency cepstral coefficients* (MFCCs) [7]

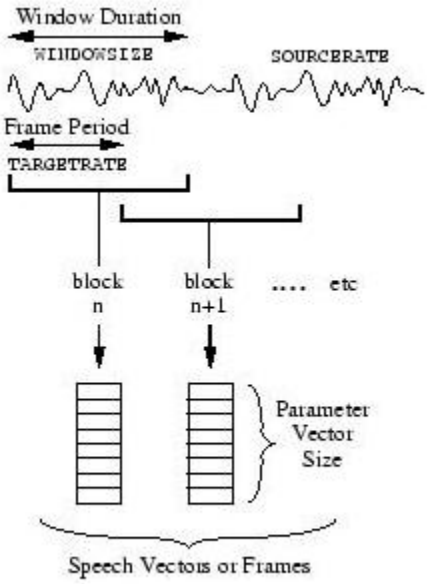

Figure 3-1: Feature extraction typical windows: 10ms overlap, 25 ms window size.

and *perceptual linear prediction* (PLP) [30].

3.2.1 Mel Frequency Cepstral Coefficients (MFCC)

The first approach, MFCC, takes the Fourier transform on the speech signal to obtain the spectrum. The spectrum power, called the mel-scale, is scaled and approximated to the response of the human ear. The mel frequencies take the algorithm and apply it to a discrete cosine transform to smoothen out the spectral and decorrelate the feature elements.

The Fourier transform of the signal $x[n]$ is given by

$$X[k] = \sum_{n=0}^{N-1} x[n]w[k-n]\exp(\frac{2\pi j}{N}kn), \tag{3.1}$$

with N as the length of the frame, and the Hamming window

$$w[n] = 0.54 - 0.46\cos(\frac{2\pi n}{N-1}). \tag{3.2}$$

The power spectrum is

$$S[k] = (\text{real}(X[k]))^2 + (\text{imag}(X[k]))^2. \tag{3.3}$$

The mel spectrum is given by

$$m[l] = \sum_{k=0}^{N/2} S[k]mel[k], \tag{3.4}$$

with $l = 0, ..., L-1$, where L is the number of mel weighting filters.

In addition, the mel filter is a series of L bandpass filters designed to simulate the bandpass filtering

$$mel(f) = 2595 \ln(1 + \frac{f}{700}). \tag{3.5}$$

A discrete cosine transform is applied to the natural logarithm of the mel spectrum to obtain the mel cepstrum:

$$c[n] = \sum_{i=0}^{L-1} \ln(m[i]) \cos(\frac{\pi n}{2L}(2i+1)) \tag{3.6}$$

where C is the number of cepstral coefficients, and $n = 0, ..., C-1$.

3.2.2 Perceptual Linear Prediction (PLP)

The second approach, PLP, computes linear prediction coefficients from a perceptually weighted non-linearly compressed power spectrum. The linear prediction coefficients are then used as the feature vectors. PLP was derived from the *linear predictive* (LP) approach [40], but PLP analysis is more consistent with human hearing. As discussed for MFCC features, the speech signal is applied to the Hamming window, the Fourier transform is taken, and the spectrum is computed. Instead of mel filters

like in MFCCs, PLP analysis uses a bark filter to warp the spectrum frequency:

$$\Omega(\omega) = 6\ln(\frac{w}{1200\pi} + \sqrt{[(\frac{\omega}{1200\pi})^2 + 1]}). \tag{3.7}$$

Auto-regressive modeling (AR) estimate coefficients are applied to the warped spectrum to minimize the errors. For an all-pole model with order p, and AR estimates $a_1, ..., a_p$ to minimize E, we get

$$R_{i+1} = a_1 R_i + a_2 R_{i-1} + ... + a_p R_{i-p+1} + E. \tag{3.8}$$

This problem is solved by using Yule-Walker equations or other more efficient methods by Levinson.

3.3 Usage of Spectral Features in Acoustic Model

The spectral features are used as the input for acoustic model (AM) training. During decoding, the acoustic scores are combined with language model scores to generate the most likely word sequence. Ultimately, the input features need to have a discriminative power, so they will be more likely recognized correctly as a phoneme state given a trained acoustic model.

As presented in Chapter 2, the most widely used framework for acoustic modeling is hidden Markov model (HMM), which does not have a long memory of the sequence of states. With the short memory issue of HMM, the spectral features are more powerful for acoustic modeling when each input feature vector captures more information than just one single frame of 25 ms length. In practice, the current spectral feature frame is concatenated with its neighbor frames or their derivatives to form a higher dimensional feature vector. The concatenated features require higher computational complexity due to longer vector lengths. Moreover, not all dimensions of the concatenated features are really necessary due to their similarities. To reduce

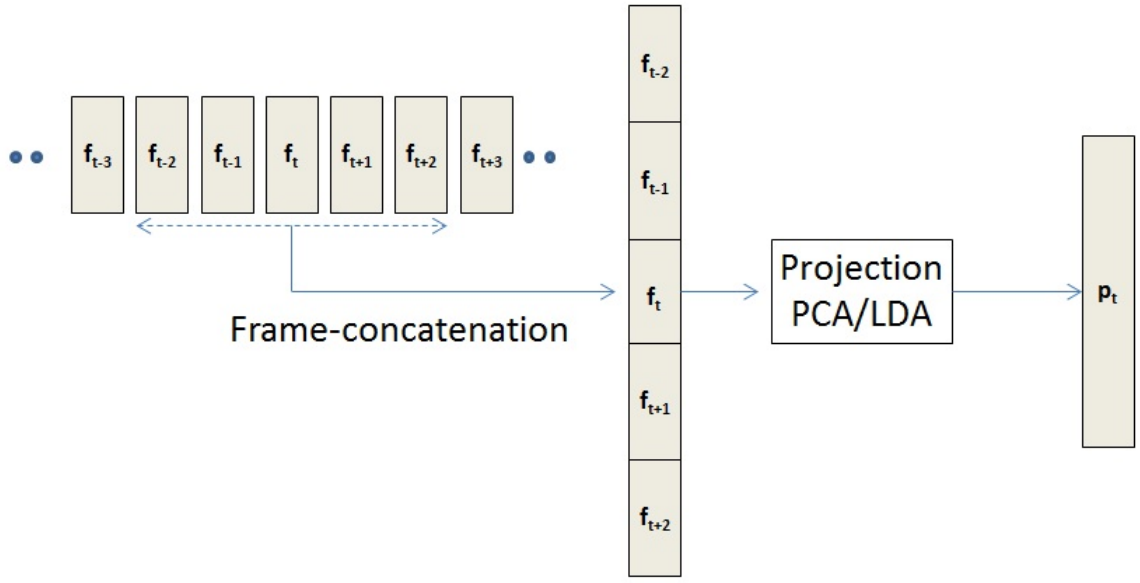

Figure 3-2: Feature concatenation and projection for AM training and decoding.

the computational complexity without losing the discriminative power, the concatenated features are projected to a lower order by linear or non-linear transforms. The projected features are then used as inputs of AM training and decoding.

3.3.1 Linear Transforms

One of the most used dimensional reduction techniques for digital signal processing (DSP) and pattern recognition is **principle component analysis** (PCA) [12]. PCA finds orthogonal components and ranks them based on their corresponding eigenvalues, which present the variability of the data in those directions. The corresponding eigenvectors combine to form the projection matrix to transform the features. The dimensions of the projected features are usually less than the original feature vectors. The designated length of PCA projected feature is driven by the application's data.

Figure 3-3: A sample of Principle Component Analysis (PCA) for feature dimensional reduction in 2-D data.

Given the matrix data X, the first eigenvector is the solution of

$$w_{(1)} = arg\max\{\frac{w^T X^T X w}{w^T w}\}. \tag{3.9}$$

The k^{th} eigenvector is found recursively by

$$w_{(k)} = arg\max\{\frac{w^T X_{k-1}^T X_{k-1} w}{w^T w}\} \tag{3.10}$$

where

$$X_{k-1} = X - \sum_{s}^{k-1} X w_{(s)} w_{(s)}^T. \tag{3.11}$$

PCA projects the features to sub-spaces that have the largest variances, which could be fit to an image processing application. However, PCA does not guarantee that the projected features have the same discriminative power as the original ones.

Compared to PCA, **linear discriminant analysis** (LDA) [12] has proven to be a better choice for feature transform in pattern classification. Instead of projecting the features to the direction with largest variances of data X, LDA tries to separate the data based on their truth labels. The transform matrix incorporates the eigenvectors

corresponding to the eigenvalues of the ratio of the between classes Σ_b variance to the within classes Σ variance.

$$S = \frac{w^T \Sigma_b w}{w^T \Sigma w} = \frac{w^T \Sigma_b \Sigma^{-1} w}{w^T w}. \tag{3.12}$$

The between classes variance is defined by the sample co-variance of the class means:

$$\Sigma_b = \sum_{i=1}^{C} (\mu_i - \mu)(\mu_i - u)^T. \tag{3.13}$$

The within classes variance is the sum of all class variances:

$$\Sigma = \sum_{i=1}^{C} \sum_{x \in c_i} (x - \mu_i)(x - \mu_i)^T. \tag{3.14}$$

LDA is useful for feature dimensional reduction, but it also has limitations. LDA requires that the feature dimensions are un-correlated to be modeled in diagonal variances in Gaussian mixtures. This issue is resolved by applying a transform like maximum likelihood linear transform (MLLT) [26] on top of LDA. Overall, LDA or any of its variations are inexpensive to train, but they are not always correlated to the recognition accuracy because they are not optimized directly on the recognition accuracy.

3.3.2 Discriminative Feature Transforms

As presented in chapter 2, discriminative training approaches such as maximum mutual information (MMI) and minimum phoneme error (MPE) lead to better recognition performances than maximum likelihood (ML). Those discriminative approaches are trained to separate the "correct" hypothesis from the similar ones, while ML tunes the model's parameters to fit the training data:

$$\lambda_{MPE} = arg \max_{\lambda} \sum_{r=1}^{R} \frac{\sum_{W_i} p(O|W_i, \lambda) p(W_i) \epsilon(W_i, W_{ref})}{\sum_{W_i} p(O|W_i, \lambda) p(W_i)} \tag{3.15}$$

where $\epsilon(W_h, W_{ref})$ is a measure of the phoneme accuracy in each word hypothesis W_i. Given the time stamp of the phoneme boundaries, each phoneme in W_i is compared against the time in the reference W_{ref}, and $\epsilon(W_h, W_{ref})$ is the overlap percentage between these two word sequences.

Besides applying on model spaces to train HMM parameters, MPE optimization [63] is also used to train transforms for features. The original feature x_t is transformed to y_t.

$$y_t = x_t + Mh_t \tag{3.16}$$

where h_t is a very high dimension vector to capture the discriminative information; this could be derived from the original feature x_t or any other features to capture different kinds of information. And M is the projection matrix to reduce the high dimensional feature h_t to the same length as the original feature x_t.

The original feature x_t is added to the projected feature as complementary to h_t, and it also helps to avoid an unstable issue during initialization. In practice, the high dimensional feature h_t is usually generated by computing the posteriors for the given original frame x_t and its neighbors over about 100,000 Gaussians. Those posteriors are stacked together to form a sparse vector, in which most of the elements are 0. The projection matrix M is learned from gradient descent method.

Region dependent transform (RDT) [89] is another discriminative method to project features. The objective function of RDT is also based on the MPE criterion. RDT uses a set of Gaussian mixtures corresponding to the "regions" to compute the posteriors given a feature vector. The regions can be the phonemes or phonemes' states, where each has its own transform matrix A_i. The original feature x_t is trans-

formed to y_t.

$$y_t = \sum_{i=1}^{N} p(i|o_t) A_i x_t \tag{3.17}$$

where N is the number regions, and $p(i|o_t)$ is the posterior for region i given a feature vector o_t, which can be the same as x_t or a different feature at time t.

The feature y_t is a weighted sum of all regions' transformed feature. When there is only one region, the posterior probability is 1. The RDT region matrix A_i is trained also on a gradient descent method. The study on MLP features [51] shows that a different feature o_t in RDT compared to x_t yields a better performance as a result of the complementary benefit.

3.4 Discriminative Features

The objective of the standard feature extraction is to convert the input speech to spectral vectors, which, in fact, do not carry phoneme information explicitly. To increase the context information, the spectral feature vectors are concatenated with their neighbors and used as input for acoustic model training. To improve the discriminative power of the spectral features, the linear or non-linear transforms are applied.

In the recent years, discriminative or probabilistic features have been studied and applied in ASR systems. The posterior probability of a sample belonging to a class of phoneme with or without context is estimated by using classifiers such as a neural network or a decision tree. The probabilistic features are then used in GMM/HMM training as the spectral features.

3.4.1 Multi-Layer Perceptrons Features

Multi-Layer Perceptrons (MLP) [18][91] have been used to extract discriminative features. A single neural network with multiple hidden layers is connected by links with weights.

Given the input vectors $X = [x_0, x_1, .., x_m]$, their corresponding weights $\{w_0, w_1, .., w_m\}$, and the bias b, the output has a form of

$$o_t = b + \sum_{i=0}^{m} w_i x_i, \tag{3.18}$$

and the activation function $\varphi(.)$ such as sigmoid is defined by

$$z = \frac{1}{1 + e^{-o_t}}. \tag{3.19}$$

The objective function of MLP is defined by the cost function to measure the error of the hypothesis versus the truth. One of the cost functions is the mean squared error

$$\varepsilon = \sum_{n=1}^{N} (z(x_n) - y_n)^2. \tag{3.20}$$

Another cost function for MLP, where the hypothesis is the probability of a sample belonging to a class, is defined by the cross-entropy

$$\varepsilon = \sum_{n=1}^{N} \sum_{i=0}^{m} y_{ni} \ln(z_{ni}) + (1 - y_{ni}) \ln(1 - z_{ni}). \tag{3.21}$$

MLP is trained using the back-propagation technique to minimize the cost function of the gradient descent approach. The weights of links are updated

$$w_i \leftarrow w_i + \eta \Delta wi = w_i + \eta \frac{\delta \varepsilon}{\delta w_i} \tag{3.22}$$

where η is the learning rate.

Since MLP can handle very long input vectors, in the work of using Temporal patterns (TRAPs) [31], input features are up to 0.5 or 1 second length instead of 25 ms window like PLP or MFCC features. The long input features help MLP to capture long context information. The final posterior probabilities from the output layer, or any intermediate layer, are then used as new features to train acoustic models for speech recognition systems. The studies from SRI, IBM, and BBN show that MLP features work better when combined with the traditional features, such as PLP or MFCC.

3.5 Thesis Scope

Motivated by the success of MLP features, the work laid out in this thesis investigates AdaBoost algorithm to generate probabilistic features in ASR applications. The Gaussian Mixture Model (GMM) classifiers, which have been used for acoustic modeling in most state-of-the-art ASR systems, serve as weak-learners for AdaBoost.

GMMs have been researched and used successfully in most of state-of-the-art ASR system, over the last decades. They are used to model the acoustic feature inputs associated with HMM states. By using EM algorithm, GMMs with enough parameters can fit most data. Moreover there are techniques such as BIC [81] to avoid the overfitting issue.

Despite all these advantages, a limitation of GMMs is that they require large number of parameters. These parameters are learned statistically from training samples, and with the classifiers that compare the distances from the means to determine what class a sample belongs to. The samples around the boundaries, thus, can be classified wrongly among the classes.

On the other hand, AdaBoost algorithm has shown its usefulness to combine the weak-learners to improve their performances. While the GMM training attempts to fit the data using EM, AdaBoost adjusts the classifiers based on their classification

accuracy. The iterative training then helps to improve the committee's results at the end.

Instead of training one set of GMMs, AdaBoost adjusts the samples' weight to train one set of GMMs per iteration. Thus, the training process is computationally expensive requiring improvements in the algorithm to make the training process feasible for thousands hours of audio.

The main goals of the thesis therefore are as follows: 1) Propose and implement tools to train AdaBoost with GMM base classifiers in parallel to handle thousands of hours of data; 2) Train AdaBoost probabilistic features; 3) Use the AdaBoost features in ASR; 4) Do analysis of the results and compare with other discriminative features.

Chapter 4

Adaptive Boosting

4.1 Introduction

Ensembles are meta-algorithms that use multiple models to combine classification rules. In general, the output from the ensemble is more accurate than any individual classifier. The two most popular methods to construct an ensemble are *Bagging*, introduced by Breiman [5], and *Boosting*, introduced by Freud and Schapire [21]. Both methods are based on re-sampling techniques to obtain different sets of training samples for each individual classifier.

Adaptive Boosting (AdaBoost) is the first practical version of Boosting. Instead of re-sampling the training set, AdaBoost adjusts the training sample weights based on the errors of the current classifier. The next iteration classifier is trained on the newly weighted training samples to fix the hard samples misclassified by previous classifiers.

4.2 Bagging Algorithm

A classifier is unstable if small changes in training data lead to significant differences in classification accuracy. Bagging algorithm improves recognition for unstable classifiers

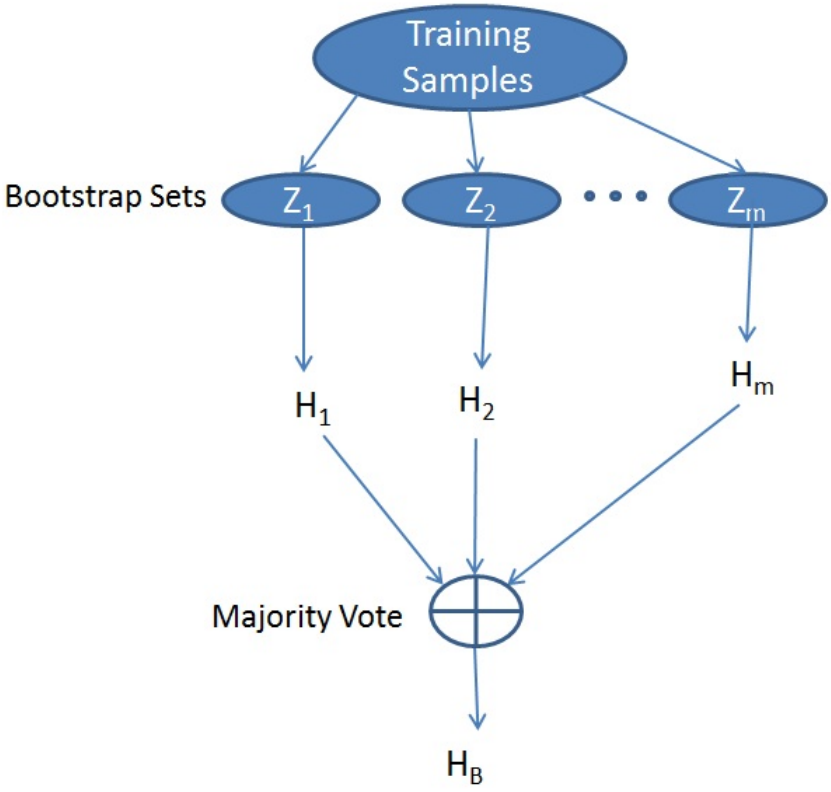

Figure 4-1: Bagging algorithm - modified from T. Hatie, Elements of Statistical Learning, 2nd edition, 2001

by smoothing over discontinuities. Given a standard training set D with size of n, the Bagging algorithm generates m new training sets, each of size $n' < n$, by sampling examples from D uniformly with replacement. By sampling with replacement, it is likely that some examples will be repeated in each new training set, leading to bootstrap set. Each bootstrap set is then used to train a different component classifier. All the component classifiers are the same form such as neural network, Gaussian mixture, or decision tree. The different parameter values are due to different training sets. The final decision of Bagging algorithm is based on a majority voting scheme. By averaging the hypotheses over the bootstrap sets, Bagging algorithm reduces the variance of the base classifiers with respect to the same training data set D.

Bagging Algorithm for Classification
Input: $Z = \{z_1, z_2, .., z_n\}$, with $z_i = (x_i, y_i)$ as training sample m, number of training sets For $i = 1, .., m$: Train weak learner H_i by using the bootstrap set Z_i
Output: $H(x) = sign(\sum_{i=1}^{m} H_i(x))$, final classification for each sample x

4.3 Boosting Algorithm

In 1989, Schapire proposed Boosting algorithm and showed that a weak learner, which is slightly better than a random guessing classifier, could improve performance by using two additional classifiers trained on filtered sets of training data. While Bagging algorithm is effective for unstable classifiers, Boosting is powerful combining multiple "base" classifiers to generate the final outputs, which is significantly better than any base classifier. Similar to the Bagging algorithm, the final decision of Boosting is a majority voting scheme.

The input of the algorithm is a set of samples (x_1, y_1), (x_2, y_2),..., (x_m, y_m) where x_i is a training sample and $y_i \in \{-1, 1\}$ is its label or the truth. In step 1, the first weak learner H_1 is trained on N randomly selected samples from Z. A new set of other N samples where half are mis-classified by H_1 is used to train the weak learner H_2. In step 3, the third weak learner H_3 is trained on the set of N samples, in which H_1 and H_2 disagree on.

Boosting Algorithm [Schapire 1989]
Input: $Z = \{z_1, z_2, .., z_n\}$, with $z_i = (x_i, y_i)$ as training set.
1. Train weak learner H_1 on a random set of N samples, a subset of Z. 2. Train weak learner H_2 on subset N samples, half of them misclassified by H_1 3. Train weak learner H_3 on N samples, which H_1 and H_2 disagree on.
Output: Final classifier $H(x) = sign(\sum_{i=1}^{3} H_i(x))$

As we can see, the connection between Boosting and Bagging is that both algorithms randomly select the training sets and are based on a majority voting scheme to generate a final decision. However, Boosting reduces the randomness factor by

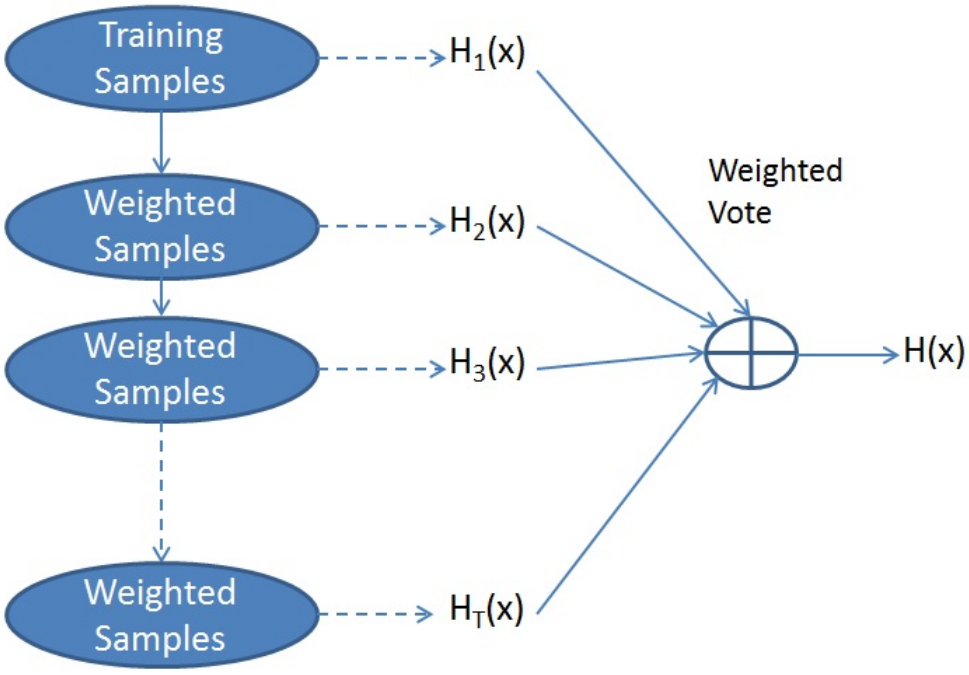

Figure 4-2: AdaBoost algorithm - modified from T. Hatie, Elements of Statistical Learning, 2nd edition, 2001

focusing more on the subset of samples that the first two weak learners disagree on, while Bagging just uses the bootstrap sets selected randomly without taking into account the performance of data sets.

4.4 Adaptive Boosting Algorithm

In 1997, Freund and Schapire proposed Adaptive Boosting (AdaBoost) algorithm [22]. Instead of randomly sampling the training set as in both Boosting and Bagging algorithms, AdaBoost re-weights training samples. Figure 4-2 shows the main framework of the AdaBoost algorithm illustrating how the training set stays the same for all times, but the training samples are re-weighted after each iteration.

This algorithm is called adaptive boosting (AdaBoost) since the weights of the training samples are changed over the iteration to make sure that the hard samples

have a better chance to be classified correctly. The samples' weights are calculated based on the accuracy of the same base classifier obtained in the previous iteration. Training samples misclassified by previous classifiers are given higher weights when used to train the next classifier. Compared to the Boosting algorithm, the AdaBoost has more training iterations, and it only stops when reaching the converged condition. Table 4.1 shows a comparison between two algorithms. The final outputs are the weighted sum of all individual classifiers where the weights are estimated from the performance of the classifiers.

Feature	Boosting	Adaptive Boosting
Training samples	Random electing	Re-weighting
# iterations	3	Until converged
Final decision	Majority vote	Weighted vote

Table 4.1: Comparison of Boosting and AdaBoost algorithms

4.4.1 Binary AdaBoost Algorithm

Discrete AdaBoost Algorithm

1. $t = 0$: $w_i^1 = \frac{1}{N}$.
2. Repeat for $t = 1, 2, .., T$:
   (2.a) call Weak-learn, get back a hypothesis $h_t : X \to \{-1, 1\}$.
   (2.b) Call the error of h_t: $\varepsilon_t = E[I(h_t(x_i) \neq y_i)]$.
   (2.c) Set $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$.
   (2.d) Set the new weights vector to be $w_i^{t+1} = w_i^t \beta_t^{1-I(h_t(x_i)\neq y_i)}$.
3. Get the hypothesis output: $h_f(x) = sign \sum_{t=1}^{T} \log \frac{1}{\beta_t} h_t(x)$

The first version of AdaBoost was called AdaBoost.M1 or Discrete AdaBoost algorithm, since the output for each base classifier is either -1, or 1. At the initialized step, all the sample weights are set equally. The weak learner is called to get the hypothesis back $\in \{-1, 1\}$. The weak learner's error ε_t is computed based on the total difference between truths and hypotheses. In step (2.c), the weak learner weight β_t is set as a function of the iteration estimated errors, or pseudo-loss, so β_t is less than

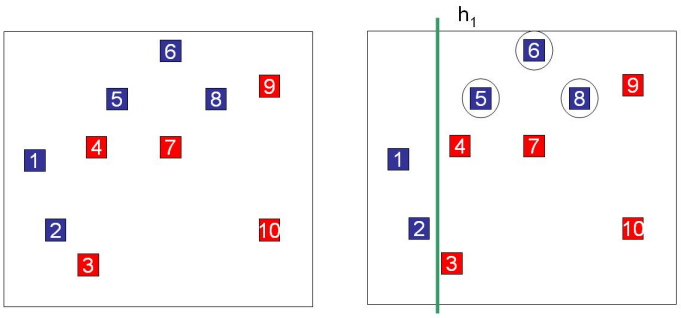

Figure 4-3: The first weak learner - The left figure shows the samples needed to classification. The right figure shows the classifier h_1 = vertical line, the iteration error $\varepsilon_1 = \frac{3}{10}$, and the iteration classifier weight 2.33

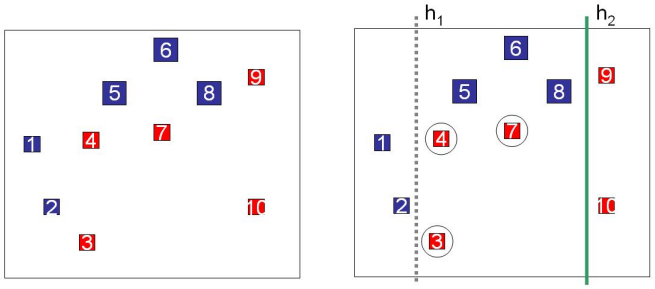

Figure 4-4: The second weak learner - The left figure shows the samples after the weights are changed. The right figure shows the classifier h_2 = vertical line, the iteration error $\varepsilon_2 = 0.21$, and the iteration classifier weight 1.51.

1. The weights of training samples are changed "adaptively" based on accuracy of the same base classifier obtained in the previous iteration. Training samples misclassified by one of the classifiers are given higher weight when used to train the next classifier as we can see in step (2.d) of the algorithm, since $\beta_t < 1$. The final output of the AdaBoost algorithm is weighted sum of the iteration classifiers, and with lower error or higher β_t, the classifier contributes a higher weight on the final results. For a quick view of the two-class AdaBoost algorithm, we use a simple classification example in figures 4-3 to 4-6. In this simple example, we aim to classify all the training samples with two labels, red and blue. We can see that the data points are not separable by a straight-line classifier. By using AdaBoost, we can combine three weak classifiers to

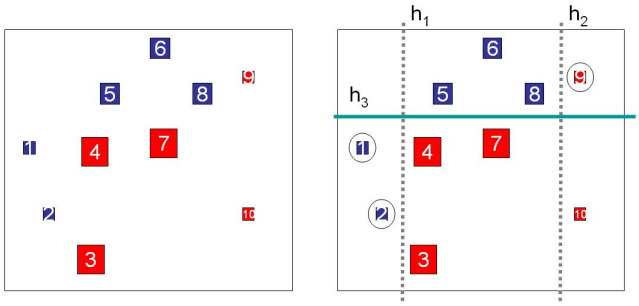

Figure 4-5: The third weak learner - With the updated distribution D_t weak classifiers, h_3 = horizontal line, the iteration error $\varepsilon_3 = 0.13$ and the iteration classifier weight 1.08.

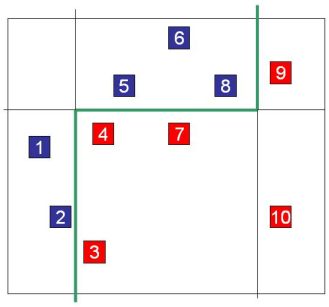

Figure 4-6: The final output - The mixture combination of all the three "weak" classifiers from figures above; the data is classified perfectly in this final result.

classify the data perfectly. At each iteration, the samples' weight are updated based on the value of the errors.

At the first iteration, the weak learner h_1 is the vertical line in Figure 4-3. It classifies 10 samples into two groups, with blue on the left and red on the right. Three out of the 10 samples (samples 5, 6, and 8) are misclassified. With the equal weights, the error of the iteration is $\varepsilon_1 = 0.3$, and the weight β_1 is computed accordingly. The weights of training samples are changed based on the formula in step (2.d). The weights of the three misclassified samples are increased, while the weights of the rest are decreased.

The weak learner is called at the second iteration with new sample weights, and the results are as shown in Figure 4-4. For this iteration, the misclassified samples are 3, 4, and 7. The error ε_2 is 0.21, instead of 0.3, due to the weighted samples. The sample weights are updated based on the error and the previous weights.

The weak learner in the third iteration returns the results in Figure 4-5. The outputs of the three weak learners are then combined with weights β_t, and we can see that the data is classified correctly for all samples.

4.4.2 Additive Model Fits AdaBoost

In 2000, Friedman published a paper [23] explaining how AdaBoost works. In this work, Friedman demonstrates how an additive model can be used for gradient descent algorithms in signal processing or neural networks. An additive model is presented as a set of elementary functions with a form:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{4.1}$$

where $\{\beta_m, m = 1, 2, .., M\}$ are coefficients, and $b(x; \gamma) \in R$ are basic functions of x and parameters γ.

The objective of an additive model is to find the set of coefficients $\{\beta\}$ and pa-

rameters $\{\gamma\}$ to minimize a loss function given the training data set. Depending on the application, the loss function could be the total likelihood, average square error, or other similar forms

$$min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)). \tag{4.2}$$

Instead of optimizing the whole loss function once, which could require very expensive computational complexity, the algorithm optimizes individual loss functions independently. That means at each step, the loss function is optimized

$$min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, \beta b(x_i; \gamma)) \tag{4.3}$$

Forward Additive Modeling Algorithm
1. $m = 0$: $f_0(x) = 0$.
2. Repeat for $m = 1, 2, .., M$:
(2.a) Estimate $(\beta_m, \gamma_m) = \arg min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
(2.b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

The way that the Discrete AdaBoost algorithm trains the weak learners is equivalent to the optimization of the additive model with the exponential loss function:

$$L(y, f(x)) = e^{-yf(x)}. \tag{4.4}$$

Given a set of training data $\{(x_i, y_i)\}$, with training instances x_i, and corresponding truths y_i. Assume that we have a set of weak learners, and we want to estimate the weights for them to achieve the best classifying result. For a training sample x_i, a weak learner t has a hypothesis $h_t(x_i) \in \{-1, 1\}$. The final classifying hypothesis for x_i is:

$$H(x_i) = \beta_1 h_1(x_i) + \beta_2 h_2(x_i) + .. + \beta_T h_T(x_i) \tag{4.5}$$

At each iteration t, we want to select and add a new weak leaner t to the current combination $H_{t-1}(x_i)$ to the extended combination:

$$H_t(x_i) = H_{t-1}(x_i) + \beta_t h_t(x_i) \tag{4.6}$$

where

$$H_{t-1}(x_i) = \beta_1 h_1(x_i) + \beta_2 h_2(x_i) + .. + \beta_{t-1} h_{t-1}(x_i). \tag{4.7}$$

To estimate the weights $\{\beta_t\}$, we need to define an optimization function, and there is more than one way to do so. In this AdaBoost algorithm, the exponential loss function is used. Other loss functions will be discussed later for comparison.

$$E = \sum_{i=1}^{N} e^{-y_i H_t(x_i)} \tag{4.8}$$

$$= \sum_{i=1}^{N} e^{-y_i (H_{t-1}(x_i) + \beta_t h_t(x_i))}. \tag{4.9}$$

Since our attention is to train the next weak classifier t and to find the weights β_t, we break the loss function into two parts, one presenting the past loss and another the next classifier.

$$E = \sum_{i=1}^{N} e^{-y_i H_{t-1}(x_i)} e^{-y_i \beta_t h_t(x_i)} \tag{4.10}$$

$$= \sum_{i=1}^{N} w_i^{(t)} e^{-y_i \beta_t h_t(x_i)} \tag{4.11}$$

where the weight for the sample x_i at iteration m is

$$w_i^{(t)} = e^{-y_i H_{t-1}(x_i)} = w_i^{(t-1)} e^{-\beta_t y_i h_{t-1}(x_i)}. \tag{4.12}$$

When x_i is classified correctly, its weight decreases

$$w_i^{(t)} = w_i^{(m-1)} e^{-\beta_t}. \tag{4.13}$$

Otherwise, it increases

$$w_i^{(t)} = w_i^{(m-1)} e^{\beta_t}. \tag{4.14}$$

The set of $\{w_i^{(t)}\}$ is used as the sample weight to train the current weak classifier m. Since y_i is either -1 or $+1$, we can divide the loss function into two parts

$$\begin{aligned} E &= \sum_{y_i = h_t(x_i)} w_i^{(t)} e^{-\beta_t} + \sum_{y_i \neq h_t(x_i)} w_i^{(t)} e^{\beta_t} && (4.15) \\ &= (\sum_{y_i = h_t(x_i)} w_i^{(t)}) e^{-\beta_t} + (\sum_{y_i \neq h_t(x_i)} w_i^{(t)}) e^{\beta_t}. && (4.16) \end{aligned}$$

At each training iteration, AdaBoost re-normalizes the training sample weights so that the total sum of the weights is 1. As we can see, the total loss varies from the minimum at $e^{-\beta_t}$ when all samples are classified correctly and to the maximum at e^{β_t} when all are classified incorrectly. To find the optimal weight for β_t, we take the derivative of the loss function with respect to β_t, and set the derivative to 0.

$$\frac{\partial E}{\partial \beta_t} = -(\sum_{y_i = h_t(x_i)} w_i^{(t)}) e^{-\beta_t} + (\sum_{y_i \neq h_t(x_i)} w_i^{(t)}) e^{\beta_t} = 0 \tag{4.17}$$

Let W_c be the total weight of the correct samples, and W_e be the total weight of the in-correct samples.

$$W_c = \sum_{y_i = h_t(x_i)} w_i^{(t)} \tag{4.18}$$

$$W_e = \sum_{y_i \neq h_t(x_i)} w_i^{(t)} \tag{4.19}$$

The weight β_t will be

$$\beta_t = \frac{1}{2}\log\frac{W_c}{W_e} \tag{4.20}$$

$$= \frac{1}{2}\log\frac{1-e_t}{e_t} \tag{4.21}$$

where the weighted error rate for the current weak learner is

$$e_t = \frac{W_e}{W_c + W_e}. \tag{4.22}$$

4.4.3 Loss Functions

In pattern classification, loss functions are used to represent the cost of an event or action as a real number [70]. Instead of using the classification error, the loss function allows making error in close cases. In other words, some classification errors are more costly than others. The optimization problems find the parameters to minimize the loss functions. There are many ways to define a loss function, such as least mean square (LMS), misclassification $(0-1)$, and exponential. Figure 4-7 shows a comparison of the most common loss functions in decision theory. In the binary case, a sample x_i is classified correctly when

$$y_i f(x_i) > 0. \tag{4.23}$$

A robust classifier should have a loss function that has a higher value on the negative side and a lower value on the positive side.

The first loss function is based on the LMS approach, which has a form of

$$L_{LMS} = (y - f(x))^2. \tag{4.24}$$

The squared error loss function is a convex function. It penalizes the negative side; it reduces at $f \in [0, 1]$, but increases again when $yf > 1$. Remember that $f \in R$, so

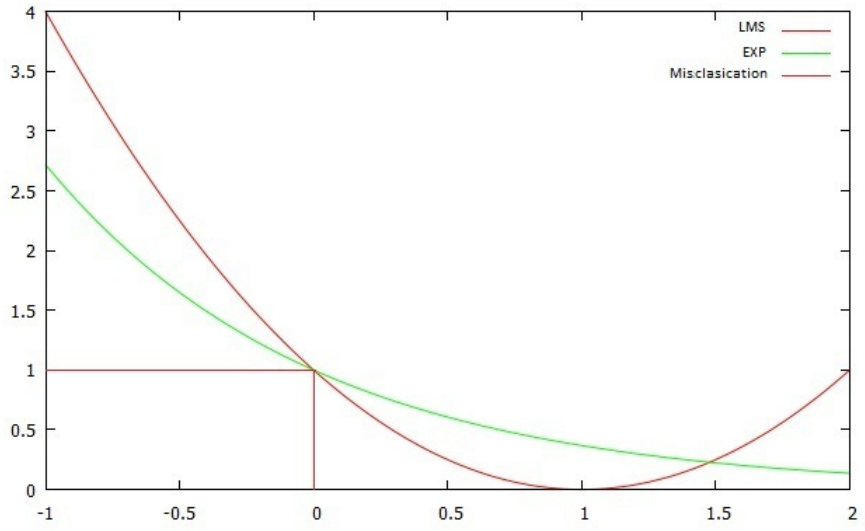

Figure 4-7: Loss functions modified from T. Hatie, "Elements of Statistical Learning, 2nd edition, 2001"

when $f > 1$ or $f < -1$ this means that the output is classified with higher certainty.

The second loss function relies on the classification error. The misclassification, or $0-1$, loss function works perfectly on the training data, but it is not a robust choice for a classifier.

$$L_{\neq} = I(sign(f) \neq y). \tag{4.25}$$

The $0-1$ loss function only has a unit penalty on the negative side, and no penalty on the positive side, so it is too sensitive around $f(x) = 0$.

The third loss function is an exponential function. The exponential loss function is a better choice for the classifier since it is monotone decreasing and smooth. It penalizes the negative size more heavily than the positive one.

$$L_{exp} = e^{(-yf(x))}. \tag{4.26}$$

Training weak learners for AdaBoost is equivalent to finding basis functions and their

weights in an additive model. In the case of the binary AdaBoost, the additive model uses the exponential loss function.

Misclassification	$I(sign(f) \neq y)$
Squared Error	$(y-f)^2$
Exponential	$e^{(-yf)}$

Table 4.2: Different loss functions

4.5 Multiclass AdaBoost

For the first part of the chapter, we used the binary case to introduce AdaBoost where AdaBoost is optimized by minimizing a defined loss function. The multiclass AdaBoost [92][73] is generalized from the binary case. The hypotheses from each weak learner $h_f : X \to Y$ are the probabilities, or confidences, of the sample belonging to the classes. The loss function for multi-class case has a form of

$$L(y,f) = e^{(-\frac{1}{K}(y_1 f_1 + .. + y_K f_K))} \quad (4.27)$$

$$= e^{(-\frac{1}{K} y^T f)}. \quad (4.28)$$

In the "one vs. all" multi-class AdaBoost algorithm by Zhu [92], the classification

Multi-Class AdaBoost Algorithm [Ji Zhu 2006]

1. $t = 0$: $w_i^1 = \frac{1}{N}$.
2. Repeat for $m = 1, 2, .., M$:
   (2.a) Train the classifier using the weight $\{w_i\}$ on training data
   (2.b) Compute error $err^{(t)} = \frac{\sum_{i=1}^{n} w_i I(c_i \neq T^{(t)}(x_i))}{\sum_{i=1}^{n} w_i}$
   (2.c) Compute $\beta^{(t)} = log(K-1)\frac{1-err^{(t)}}{err^{(t)}}$
   (2.d) Set the new weights vector to be $w_i^{(t+1)} = w_i^{(t)} e^{\beta^{(t)} I(c_i \neq T^{(t)}(x_i))}$
   (2.e) Re-normalize $\{w_i\}$
3. Get the hypothesis output: $h(x) = \arg max_k \sum_{m=1}^{M} \beta^{(t)} I(T^{(t)}(x_i) = k)$

function $I(c_i \neq T^{(t)}(x_i))$ is used to compare the result of the classifier, so the value of c_i is represented as a K-dimensional vector.

$$y = \begin{cases} 1 & \text{if } c = k \\ -\frac{1}{K-1} & \text{if } c \neq k \end{cases}$$

By feeding the multi-class loss function (4.28) to the additive modeling, it becomes

Forward Additive Modeling Algorithm
1. $m = 0$: $f_0(x) = 0$.
2. Repeat for $m = 1, 2, .., M$:
(2.a) Estimate $(\beta_m, \gamma_m) = \arg min_{\beta,\gamma} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i; \gamma))$.
(2.b) Set $f_m(x) = f_{m-1}(x) + \beta_m b(x; \gamma_m)$.

to find $b(x)$ and β in

$$(\beta, \gamma) = \arg min_{\beta,\gamma} \sum_{i=1}^{N} w_i e^{(-\frac{1}{K}\beta y_i^T b(x_i;\gamma))} \tag{4.29}$$

where $w_i = e^{(-\frac{1}{k} y_i^T f_{m-1}(x_i))}$ is the sample i weight, and $b(x_i, \gamma)$ is defined as the basic function or the return value of the current weak learner. For a multi-class case this is encoded as

$$b(x_i, \gamma) = \begin{cases} 1 & \text{if } T(x_i) = k \\ -\frac{1}{K-1} & \text{if } T(x_i) \neq k \end{cases}$$

We split the right hand side of (4.30) into two parts

$$\begin{aligned} & \sum_{i=1}^{N} w_i e^{(-\frac{1}{K}\beta y_i^T b(x_i;\gamma))} \\ = & \sum_{c_i = T(x_i)} w_i e^{(-\frac{1}{K}\beta y_i^T b(x_i;\gamma))} + \sum_{c_i \neq T(x_i)} w_i e^{(-\frac{1}{K}\beta y_i^T b(x_i;\gamma))} \\ = & e^{-\frac{\beta}{K-1}} \sum_{c_i = T(x_i)} w_i + e^{-\frac{\beta}{(K-1)^2}} \sum_{c_i \neq T(x_i)} w_i \\ = & e^{-\frac{\beta}{K-1}} \sum_{i} w_i + (e^{-\frac{\beta}{(K-1)^2}} - e^{-\frac{\beta}{K-1}}) \sum_{c_i \neq T(x_i)} w_i \\ = & e^{-\frac{\beta}{K-1}} \sum_{i} w_i + (e^{-\frac{\beta}{(K-1)^2}} - e^{-\frac{\beta}{K-1}}) \sum_{i} w_i I(c_i \neq T(x_i)) \end{aligned} \tag{4.30}$$

The first part of (4.30) does not only depend the current classifier, but also on the previous classifiers' results. The value of basic function $b(x_i; \gamma)$ is optimized when

$$T^{(m)}(x_i) = \arg min \sum_{i=1}^{N} w_i I(c_i \neq T(x_i)) \tag{4.31}$$

Let $err^{(m)}$ be the total weighted error of the current iteration

$$err^{(m)} = \frac{\sum_{i=1}^{N} w_i I(c_i \neq T(x_i)}{\sum_{i=1}^{N} w_i} \tag{4.32}$$

Now plug the value of $T^{(m)}$ back to (4.31) and the new definition of error $err^{(m)}$ to solve the value of β_m by getting the partial derivative with respect to β, and set it to 0. The optimized value of $\beta^{(m)}$ gives the error

$$\beta^{(m)} = \frac{(K-1)^2}{K} \log(K-1) \frac{1 - err^{(m)}}{err^{(m)}} \tag{4.33}$$

As we can see, this form of weights in the multi-class case is the same as in the AdaBoost binary case when K is set to 2. The defined classification function $I(c_i \neq T(x_i))$ could be modified to fit the case when the outputs from AdaBoost are a vector of probabilities. In the next chapter, we will discuss the usage of AdaBoost algorithm for multi-class Gaussian mixture classifiers.

4.6 Summary

This chapter introduced meta-algorithms such as Bagging and Boosting to combine based-classifiers, and the combined results are generally seen as better than any individual classifier. The first practical version of Boosting is Adaptive Boosting (AdaBoost). Instead of randomly sampling the training set like in both Boosting and Bagging algorithms, the adaptive boosting (AdaBoost) re-weights training samples. Training samples misclassified by previous classifiers are given higher weights when

used to train the next classifier. Training the weak learners for AdaBoost is equivalent to finding the basic functions for the additive model where a loss function presenting the cost of a misclassification action is defined. The original binary-case AdaBoost algorithm was extended to work with the multi-class case.

Chapter 5

AdaBoost for Gaussian Mixture Models

5.1 Introduction

The adaptive boosting (AdaBoost) algorithm is applied to combine the hypotheses of the weak learners. In automatic speech recognition (ASR) applications, the Gaussian mixture technique has been used to model acoustic features successfully. In this chapter, we study AdaBoost to combine with Gaussian mixture classifiers. The inputs are acoustic spectral features and the final outputs are the posterior probabilities of each sample given all phonetic classes.

5.2 The Gaussian Distribution

The pattern recognition application predicts the class of an object given its observation. The concept of uncertainty is a nature of the predicting process due to the limited size of the training data or the noise in measurement. The probability theory has been used as a main framework in pattern recognition applications. When combining with optimization techniques, the probability theory helps to make opti-

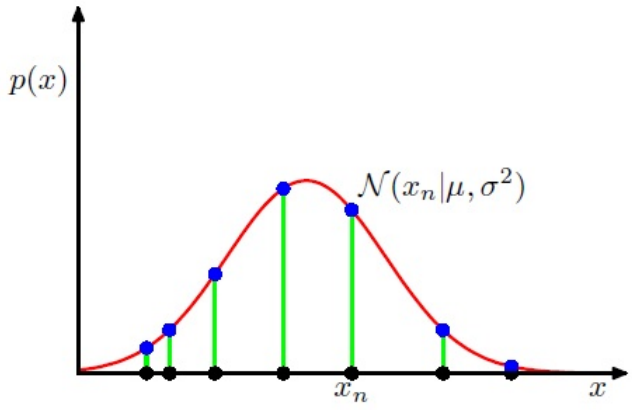

Figure 5-1: Gaussian distribution - $p(x_n)$ is the likelihood of sample x_n.

mal predictions given the training data set, which could be incomplete or contain errors or noise. The Gaussian (or normal) distribution is one of the most widely used probability distributions for continuous variables. Given a variable x, the normal distribution is defined by

$$p(x|\mu,\sigma^2) = \mathcal{N}(x|\mu,\sigma^2) \tag{5.1}$$

$$= \frac{1}{(2\pi\sigma^2)^{1/2}} exp\{-\frac{1}{2\sigma^2}(x-\mu)^2\} \tag{5.2}$$

where μ and σ^2 are the mean and the variance respectively. The most common approach to estimating the distribution's parameters is to maximize the (log) likelihood function for the whole training data set $X = \{x_i, .., x_N\}$.

$$\log p(X|\mu,\sigma^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{N}{2}\log\sigma^2 - \frac{N}{2}log(2\pi). \tag{5.3}$$

By maximizing the log likelihood function with respect to the mean μ, the mean μ_{ML} is computed as the samples' mean

$$\mu_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{5.4}$$

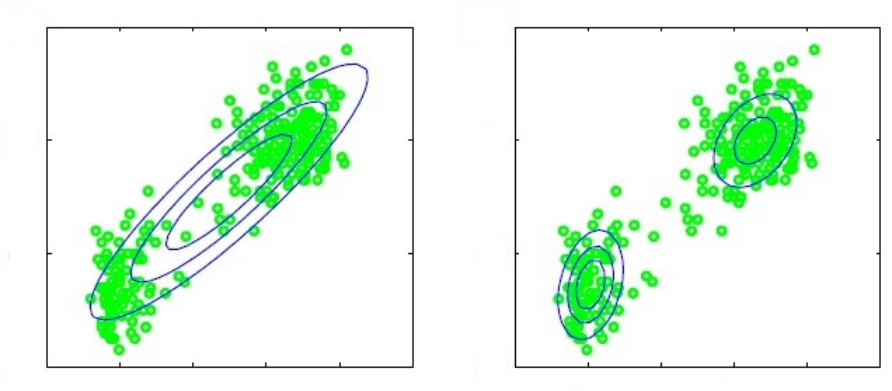

Figure 5-2: Failure of a single Gaussian distribution
(The data should be modeled using two Gaussian distributions)

And the variance has form of

$$\sigma^2_{ML} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu_{ML})^2. \tag{5.5}$$

The value of variance is biased to the training data by a factor of $\frac{N-1}{N}$. However, when N is big enough ($\rightarrow \infty$), the σ^2_{ML} becomes the "true" variance σ^2.

A single Gaussian distribution has limitations in modeling the real data because it is based on an assumption that the whole training set distributes around the mean. However, that cannot always be the case in the real data such as shown in Figure 5.2. Obviously, those data points should be modeled better by using two normal distributions instead of one. The mixture of Guassians are then proposed with a form of M densities

$$p(x) = \sum_{j=1}^{M} m_j \mathcal{N}(x|\mu_j, \Sigma_j). \tag{5.6}$$

There are two main problems that need to be solved in order to train the Gaussian mixtures. The first is to find the correct number of Gaussian clusters to model well the data, and then avoid the over fitting issue. The second is to estimate the Gaussian parameters based on the number of clusters.

5.3 K-means Clustering

The K-means algorithm partitions the data set $\{x_1, .., x_N\}$ into clusters. Assume that the number of clusters is given. Each cluster has its own mean μ_k with $k = 1, .., K$. And the means $\{\mu_k\}$ represent the centers of the clusters, where the distance from all points of the cluster to its mean is closest compared to any other means. The objective function is defined by the total distance of all points to their cluster mean

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2 \tag{5.7}$$

where r_{nk} shows if the sample x_n belongs to the cluster k by

$$r_{nk} = \begin{cases} 1 & \text{if } x_n \text{ belongs to cluster } k \\ 0 & \text{otherwise} \end{cases}$$

The goal of the K-means algorithm is to find $\{r_{nk}\}$ and $\{\mu_k\}$ to minimize J. This can be done through a two-step procedure. In the first step, the means $\{\mu_k\}$ are fixed to find the clusters for minimizing J. And in the second step, the the clusters $\{r_{nk}\}$ are then used to find the means $\{\mu_k\}$. The algorithm can set a convergent condition where either J stops reducing or it reaches to the number of iterations.

In the first step of the optimization, J can be represented as N independent terms

$$J = \sum_{k=1}^{K} r_{1k} ||x_1 - \mu_k||^2 + \sum_{k=1}^{K} r_{2k} ||x_2 - \mu_k||^2 + .. + \sum_{k=1}^{K} r_{Nk} ||x_N - \mu_k||^2. \tag{5.8}$$

Then, minimizing J is equivalent to minimizing each individual term $\sum_{k=1}^{K} r_{nk} ||x_n - \mu_k||^2$ with respect to r_{nk}. Hence, the obvious solution for that is $r_{nk} = 1$ for the minimum value of $||x_n - \mu_k||^2$

$$r_{nk} = \begin{cases} 1 & \text{if } k = \arg\min_j ||x_n - \mu_j||^2 \\ 0 & \text{otherwise} \end{cases}$$

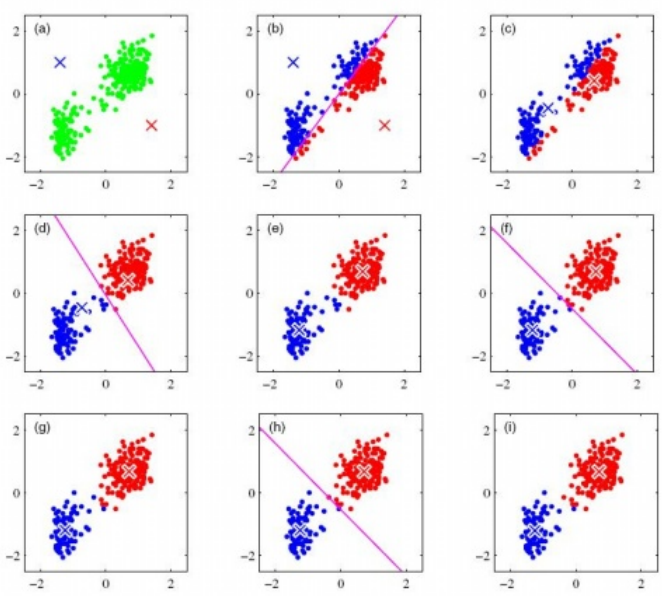

Figure 5-3: K-means - On the data set that a single Gaussian distribution does not fit.

With the new clusters $\{r_{nk}\}$, the means $\{\mu_k\}$ can be found by getting the derivative of J with respect to μ_k

$$\frac{\partial J}{\partial \mu_k} = 2 \sum_{n=1}^{N} r_{nk}(x_n - \mu_k). \tag{5.9}$$

And, the solution for μ_k is just the mean of all the points belonging to the cluster k assigned from the previous step.

$$\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}. \tag{5.10}$$

To have a quick view of the K-means, we use the same data set from the previous section when a single Gaussian could not fit them. Even with two initial means not belonging to the clusters, the K-means re-assigns the means to inside the clusters just after a few iterations. Ding and He [11] use triangle inequality theorem to prove:

$$J^{opt}(K) > J^{opt}(K+1) \tag{5.11}$$

with $J^{opt}(K)$ as the optimal value of the K-means objective function J with K clusters. This means the K-means objective functions is monotonically increased with the value of K. During K-means optimization, the model complexity, measured by K, should be penalized to avoid the over-fitting problem. One of the methods to select the model order is Bayesian information criterion (BIC) to minimize the objective function

$$BIC = \arg\min_{K} -2\ln p_K(x,\theta) + K\ln N. \tag{5.12}$$

The first part of the BIC objective function is to try to find the parameters θ to get the higher likelihood $p_K(x,\theta)$, and the second part with K penalties the higher value of K. With a too low number of clusters, K-means models the data poorly, while a very high number could cause an over-fitting problem. The K-means algorithm with the fast convergence could be used to initialize parameters for the Gaussian mixture, which will be discussed in the next section.

5.4 Gaussian Mixture Models

As we discussed above, one single Gaussian distribution cannot always fit the real data properly in the case where the data spreads to more than one cluster. K-means clustering is an option to model the data, but it is limited by its hard boundary. Mixture of Gaussians are a weighted sum of Gaussian component densities, and they are more general compared to a single Gaussian and, therefore, handle the points near the cluster boundary better than the K-means clustering, thanks to the soft membership. Each point of the training set contributes to all the distributions with different weights, which depend on the distance from the point to the cluster mean. That is why the set of Gaussians are called the Gaussian mixture.

Each component density has its own mean, variance, and mixture weight. Let sample x be a D-dimensional data vector, m_j (with $j = 1, .., M$) the mixture weights, and $\mathcal{N}(x|\mu_j, \Sigma_j)$ the Gaussian densities. A Gaussian mixture model is a weighted

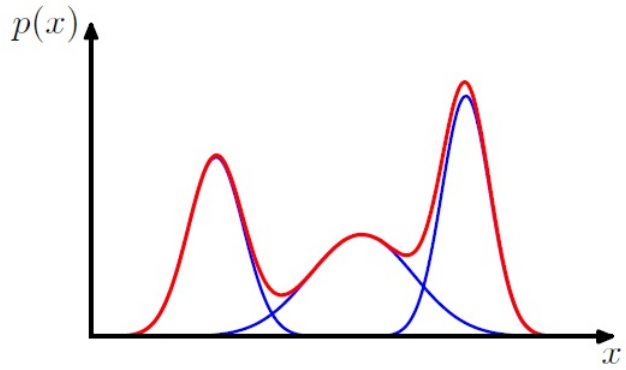

Figure 5-4: Gaussian mixture distribution - Three Gaussian distributions

sum of M components,

$$p(x|\theta) = \sum_{j=1}^{M} m_j \mathcal{N}(x|\mu_j, \Sigma_j) \tag{5.13}$$

where each component density has a form of

$$\mathcal{N}(x|\mu_j, \Sigma_j) = \frac{1}{(2\pi)^{D/2}|\Sigma_i|^{1/2}} exp\{-\frac{1}{2}(x-\mu_j)'\Sigma_j^{-1}(x-\mu_j)\}. \tag{5.14}$$

In the single Gaussian distribution, the parameters $\theta = (\mu_j, \Sigma_j, m_j)_{j=1}^{k}$ can be optimized by using a maximum likelihood framework given the training samples. However, there is an issue of singularity when optimizing a mixture of Gaussians under maximum likelihood.

$$\begin{aligned} l(\theta, D) &= \sum_{i=1}^{N} \log(\sum_{j=1}^{k} m_j \mathcal{N}(x|\mu_j, \Sigma_j)) \\ &= \alpha \log \prod_{i=1}^{N} \sum_{j=1}^{k} \frac{m_j}{det(\Sigma_j)} \exp(-\frac{1}{2}(x_i-\mu_j)^T \Sigma_j^{-1}(x_i-\mu_j)). \end{aligned} \tag{5.15}$$

Assume that one of the Gaussian components j has its mean $\mu_j = x_i$, and $det(\Sigma_j) \to 0$, or the total likelihood function is $\to \infty$. This singularity issue occurs whenever one of the Gaussian includes only one sample. Even though all the samples contribute a

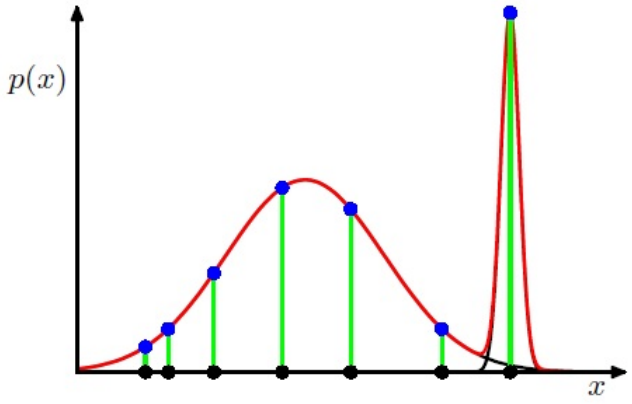

Figure 5-5: Singularity GMM - (The second Gaussian has the variance $\sigma \to 0$)

weight to the total likelihood, the singular Gaussian is still dominated by that sample at the mean due to a very high mixture weight. The example is shown in Figure 5.5. Instead of optimizing the likelihood directly, GMM parameters are mostly estimated from training data using the iterative expectation-maximization (EM) algorithm [8], [12]. And to avoid the singularity issue, some heuristics steps such as re-arranging cluster operations are used in K-means. The detail of those operations will be shown in the next chapter of implementation.

5.4.1 Expectation and Maximization (EM) algorithm

To simplify the description of the EM algorithm, we modify the goal of the problem of finding all parameters $\theta = (\mu_j, \Sigma_j, m_j)_{j=1}^k$ to just finding the means $u =< \mu_1, ..., \mu_k >$ of k clusters or k normal distributions. Given the training samples $X = \{x_1, ..., x_N\}$, we need to estimate the hidden variables $Z = \{< z_{i1}, ..., z_{ik} >\}$, where $z_{ij} = 1$ if x_i sample is generated by the j^{th} normal distribution; otherwise $z_{ij} = 0$. The general form of the EM algorithm repeats the following two steps until convergence: For each training sample set $X = \{x_1, .., x_N\}$, we define instance set $Y = \{y_1, .., y_N\}$, with

Expectation-Maximization (EM) Algorithm
1. Estimation Step Compute the objective auxiliary function $Q(u'\|u)$ for the estimate parameters u' by using the current parameters u and training samples X. 2. Maximization Step Update the model parameters u' to maximize the objective auxiliary function $Q(u'\|u)$.

$y_i =< x_i, z_{i1}, .., z_{ik} >$. The posterior probability of an instance y_i has a form of

$$p(y_i|u') = p(x_i, z_{i1}, .., z_{ik}|u') = \frac{1}{\sqrt{2\pi\sigma^2}} exp(-\frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2). \tag{5.16}$$

The total log of the probability of all N training samples is

$$\begin{aligned} \ln P(Y|u') &= \ln \prod_{i=1}^{N} p(y_i|u') \\ &= \sum_{i=1}^{N} \ln p(y_i|u') \\ &= \sum_{i=1}^{N} (\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2). \end{aligned} \tag{5.17}$$

We take the expected value of this $\ln P(Y|u')$ over the probability distribution Y. Since $< x_i >$ is given, the only part that we need to take for the expected value is z_ij. For any linear function $f(z)$ of z, we have the following equality:

$$E[f(z)] = f(E[z]).$$

Applying this equality to the expected value of $\ln p(Y|u')$, we have

$$\begin{aligned} E[\ln P(Y|u')] &= E[\sum_{i=1}^{N} (\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} z_{ij}(x_i - \mu'_j)^2)] \\ &= \sum_{i=1}^{N} (\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2). \end{aligned} \tag{5.18}$$

Therefore, the function $Q(u'|u)$ that we need to optimize is

$$Q(u'|u) = \sum_{i=1}^{N}(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2) \tag{5.19}$$

where $u' =< \mu'_1, ..., \mu'_k >$ and $E[z_{ij}]$ is computed based on the current parameter $u =< \mu_1, ..., \mu_k >$ and training data X. This expected value $E[z_i j]$ is the probability of sample x_i generated by the distribution j.

$$E[z_{ij}] = \frac{exp(-\frac{1}{2\sigma^2}(x_i - \mu_j)^2)}{\sum_{n=1}^{k} exp(-\frac{1}{2\sigma^2}(x_i - \mu_n)^2)}. \tag{5.20}$$

Thus, we first define the function Q based on the expected $E[z_{ij}]$ term (estimation step). We then find the parameters $\mu_1, ..., \mu_k$ to maximize this function Q (maximization step):

$$\begin{aligned} \arg\max_{u'} Q(u'|u) &= \arg\max_{u'} \sum_{i=1}^{N}(\ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2) \\ &= \arg\min_{u'} \sum_{i=1}^{N}\sum_{j=1}^{k} E[z_{ij}](x_i - \mu'_j)^2. \end{aligned} \tag{5.21}$$

Therefore, to maximize the log likelihood, we minimize the weighted sum of the squared errors. By setting the derivative to 0, we get the value of any mean μ_j.

$$\mu_j = \frac{\sum_{i=1}^{m} E[z_{ij}]x_i}{\sum_{i=1}^{m} E[z_{ij}]}. \tag{5.22}$$

Even though we make an assumption that all the variances are the same to simplify the optimizing process, we still update the variances in practice. In other words, after updating the means, the variances are also recalculated by using the new means, and the probability $E[z_{ij}]$. The mixture weights are also updated by adding the expected value $E[z_{ij}]$ for all N samples. The EM algorithm will repeat the following steps until convergence.

1. Estimation. Using the current parameters $\theta^{(t)} = \{\mu_j^{(t)}, \Sigma_j^{(t)}, m_j^{(t)}\}$, the expected $E[z_{ij}]$ are computed as:

$$E[z_{ij}]^{(t)} = \frac{m_j^{(t)} \frac{1}{\sqrt{2\pi\sigma_j^{(t)2}}} exp(-\frac{1}{2\sigma_j^{(t)2}}(x_i - \mu_j^{(t)})^2)}{\sum_{n=1}^{k} m_n^{(t)} \frac{1}{\sqrt{2\pi\sigma_n^{(t)2}}} exp(-\frac{1}{2\sigma_n^{(t)2}}(x_i - \mu_n^{(t)})^2)}. \tag{5.23}$$

2. Maximization. Using the expected $E[z_{ij}]$, all the parameters $\theta^{(t+1)} = \{\mu_j^{(t+1)}, \Sigma_j^{(t+1)}, m_j^{(t+1)}\}$ are updated:

$$m_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)}}{\sum_{j=1}^{k}\sum_{i=1}^{N} E[z_{ij}]^{(t)}} \tag{5.24}$$

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)} x_i}{\sum_{i=1}^{N} E[z_{ij}]^{(t)}} \tag{5.25}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)} (x_i - \mu_j^{(t+1)})(x_i - \mu_j^{(t+1)})^T}{\sum_{i=1}^{N} E[z_{ij}]^{(t)}}. \tag{5.26}$$

5.4.2 EM Initialization by K-means

The very first step E of an EM algorithm needs initial means for all the clusters to start optimizing. One way is to use random samples as the initial means, but it could reach to a sub optimal solution where one of the random initial means does not represent a cluster with enough samples. In this case, it will have a very small variance. The second issue of the EM is that it converges much slower compared to K-means. The third issue of EM is that it is much more computationally expensive due to the Gaussian computation, and is followed by mean, variance and mixture weight updating. It is cheaper for K-means with just distance comparison and the mean updating. To avoid these issues, the K-means step runs re-arranging operations to find a good enough starting point for EM.

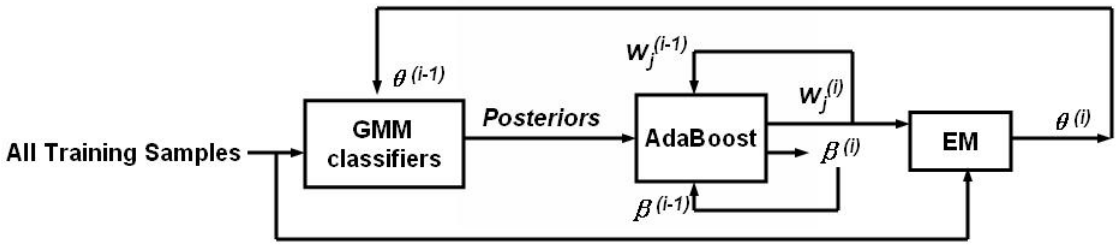

Figure 5-6: AdaBoost for GMM -(The GMM classification results are feeded to AdaBoost to estimate the sample weights, which are then used to update GMM parameters for the next iteration.)

5.5 AdaBoost for Gaussian Mixture Classifiers

As discussed in previous sections, GMM-based classifiers are a fit solution to model acoustic model features, and AdaBoost is suitable to combine the base classifiers. In this work, AdaBoost is applied on top of GMM-based classifiers to enhance their performance. AdaBoost has an ability to improve from previous iteration errors by focusing on the hard samples. The key step in AdaBoost algorithm is the sample weight estimation, which improves the classification performance by adjusting the samples' weight to focus more on the hard samples based on results from previous iterations. The current weights are based not only on the classification results but also on all previous weight values. The classification results are computed by the Gaussian mixtures. The classifier at each iteration has a form:

$$h_t(x|y) = \frac{\sum_j m_y^j \mathcal{N}(x|y_j)}{\sum_c \sum_j m_c^j \mathcal{N}(x|c_j)}. \tag{5.27}$$

For a particular sample x, $\mathcal{N}(x|c_j)$ is the j^{th} component of the Gaussian of class c, and m_c^j is its mixture weight. The GMM parameters are trained on the weighted samples where the weights are estimated by AdaBoost from the classification errors from the past iterations.

The cluster mean is estimated at

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} w_i^{(t)} x_i}{\sum_{i=1}^{N} w_i^{(t)}} \tag{5.28}$$

And the cluster variance has a form of

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{N} (w_i^{(t)} x_i - \mu_j^{(t+1)})(w_i^{(t)} x_i - \mu_j^{(t+1)})^T}{\sum_{i=1}^{N} w_i^{(t)}} \tag{5.29}$$

where $w_i^{(t)}$ is the sample i weight at iteration t.

The whole training procedure can be handled by a single job, where all the training sample weights and GMM parameters for all the classes can be updated serially. For a large system with thousands of hours of audio training data, this approach could take long to finish. We could reduce the whole training latency by using a modified version of AdaBoost for GMM, where the training data are divided into subsets, so that each training job can handle the training data belonging to a class.

5.6 Summary

In this chapter, we reviewed GMM classifiers training based on K-means and EM algorithms. To apply AdaBoost to combine GMM classifiers, training Gaussian parameters are updated by getting the feedback from AdaBoost to adjust the samples weight accordingly to focus on the hard samples in the next training iteration. In the case of the large ASR system with billion of training samples, the AdaBoost need to be modified to finish training in a reasonable time without sacrificing accuracy.

Chapter 6

Implementation

6.1 Introduction

In the previous two chapters, we have presented the usage of AdaBoost algorithm to combine GMM based classifiers by adjusting the weight of training samples. The main tasks of GMM-AdaBoost training, which require very expensive computation, include posterior calculation, sample weight estimation based on classification results, and EM/GMM training with samples' weight from AdaBoost training. In the real ASR application with billions of samples of training data, the AdaBoost algorithm is modified so that the whole training process is feasible by splitting into multiple parallel jobs. After training, the GMM models and iteration weights are then used to generate AdaBoost posterior features.

6.2 Training Data Preparation

To prepare the labels for training, a baseline acoustic model is trained on PLP (Perceptual Linear Prediction) features. The baseline acoustic model is then used to perform Viterbi forced alignment on the training features of the transcript to generate the training labels at frame level, which contains the sequence of the most likely

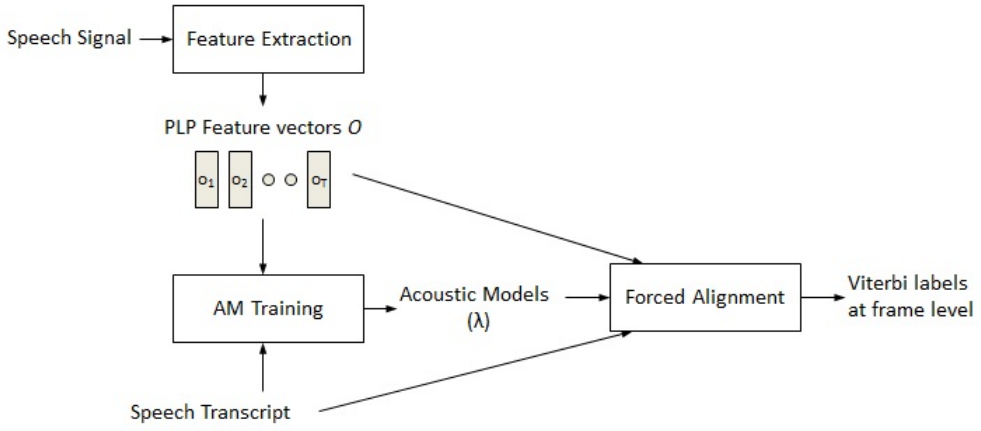

Figure 6-1: Training data preparation

states of phonemes in context.

The Viterbi alignment framework is also used in acoustic model training, presented in Chapter 2. The number of classes for AdaBoost training is determined by the number of possible labels in the whole training data set, which depends on the number of possible within-context phoneme states in the baseline acoustic model. The baseline acoustic model is tuned in order to get the designated number of possible phonemes in context states without losing the model's performance, which is measured on a development set.

The BBN Byblos system has the ability to model an acoustic model from the most complex quinphone as state-clustered tied-mixture (SCTM) to the simplest one as phonetic tied-mixture (PTM), as described in the BBN byblos system [55].

To study the behavior of the multiclass AdaBoost algorithm using the GMMs proposed above, we set up three data sets derived from a large English broadcast news speech corpus used at BBN for the EARS and GALE programs. The first data set used for training comprises 50 hours of randomly selected broadcast news shows. The second data set, also randomly selected, consists of 3 hours used for cross-validation purposes. The third data set is the development test set that consists of 3 hours of

CNN's news shows aired in 2004. The training data set was phonetically aligned to obtain the *truth* labels for training the AdaBoost model. The cross-validation data set was also phonetically aligned to obtain the *truth* labels to measure the frame classification accuracy of the AdaBoost model.

6.3 Implementation of AdaBoost for GMM

The implementation of AdaBoost for GMM classifiers in this work is modified from AdaBoost.M2 of by Freund [22] to work as multiple processes in parallel.

Given the set of training data with a size of N: $X = \{x_1, .., x_N\}$, and their corresponding labels $Y = \{y_1, .., y_N | y_i \in y_1, .., y_C\}$, AdaBoost algorithm trains a series of GMM-based classifiers and their weights $\{\beta_t\}$ for combined hypotheses with minimal overall error at the end.

$$H(x|y) = \sum_t (\log \frac{1}{\beta_t} h_t(x|y)) \tag{6.1}$$

where $h_t(x|y)$ represents the base classifiers at iteration t for a pair of sample and class (x, y).

The goal of AdaBoost can be considered as two optimization problems. The first problem is to find the set of weight coefficients $\{\beta_t\}$ given a list of classifiers with estimate errors. In section 4.4, we presented the formula to estimate the classifier coefficients β_t for each iteration based on the pseudo-loss ε_t.

$$\beta_t = \frac{\varepsilon_t}{1 - \varepsilon_t}. \tag{6.2}$$

The second problem for AdaBoost is to train a set of classifiers, which will be combined to generate the minimal overall error for the final hypothesis. In section 4.3 of additive mode, we have showed that AdaBoost algorithm is greedy. The overall optimization

is equivalent to minimizing the loss function for each iteration. To minimize the loss function, AdaBoost adjusts the samples' weight to fix the errors that the classifiers made in the previous iterations. This leads to a two-step optimization problem like the expectation-maximization (EM):

- **Estimation Step:** Estimate the base classifiers' parameters using the current sample weights.
- **Maximization Step:** Compute the samples' weight based on the classification results.

6.3.1 Sample Weight Optimization

For each iteration, the AdaBoost algorithm minimizes the defined loss function ε_t^c. Because of adaptive and accumulative characteristics of the algorithm, the current loss function actually depends on all previous training iterations up to the current one. In the other words, AdaBoost tries to fix the mistakes from all previous classifiers.
To work with GMM classifiers, the loss function (or pseudo-loss) at iteration t is defined as

$$\varepsilon_t^c = \frac{1}{2}\sum_j \sum_y \frac{D_t}{Z_t}(1 - h(x, c) + h(x, y)) \tag{6.3}$$

where the classification results are computed as the posterior of a sample x given a class y

$$h_t(x|y) = \frac{\sum_j m_y^j \mathcal{N}(x|y_j)}{\sum_c \sum_j m_c^j \mathcal{N}(x|c_j)}. \tag{6.4}$$

To optimize the loss function, we define $D_t(x, y)$ as the mislabel distribution of a pair sample and class (x, y). $D(x, y)$ measures the level of classification error for each pair of feature and label.

With Z_t as the normalized factor, the loss function can be seen as the weighted errors that compare the difference between the truth class posterior $h(x|c)$ and all other posteriors $h(x|y)$. The errors are weighted by the mislabel distribution $D(x, y)$.

During the training, the AdaBoost algorithm normalizes and keeps the mislabel distribution $\sum_y \sum_i D_t(x_i, y) = 1$. The current value $D_t(x_i, y)$ updates, based on the previous value $D_{t-1}(x_i, y)$ and the classification error, for that particular feature/label pair (x_i, y).

As presented in chapter 4, the AdaBoost algorithm needs to be optimized for the whole training set where each sample has its own weight computed by comparing the hypothesis against the truth. The GMM-based classifiers with parameters for a particular class are trained only on the samples from that class alone. The detail sample weights can be replaced by the sum of the mislabel distribution over all the classes.

$$w_t(x) = \sum_y D_t(x, y). \tag{6.5}$$

This simplification makes it possible to split the whole parameter training to multiple independent processes where each process depends on only the training samples belonging to a particular class. These processes can then be run parallel in a computer distributed system. This splitting speed-up makes it feasible for the training of billions of samples.

6.3.2 Parallel AdaBoost (version 1)

As described above, the three most computationally expensive parts for AdaBoost are Gaussian parameter training, classification, and sample weight modification. In the previous section, the proposal to use the simplified sample weight made it possible to process sample weight modifications and GMM training in parallel. Moreover, the classification, or posterior probability computation, for all samples given all classes

can also process independently as long as the GMMs are available. Let us define:

- **D:** The mislabel distribution matrix to measure the level of classification error for each pair feature and label (x, y)
- $W = \{w_1, .., w_N\}$**:** The training sample weight for all training sample x_i at each iteration t, which estimated by the total value of $\sum_y D_t(x, y)$
- ε_t^c**:** The total pseudo-loss or error at the t^{th} iteration over all training samples
- Z_t**:** The normalized factor to make sure the total mislabel distribution $\sum_y \sum_i D_t(x_i, y) = 1$
- $h_t(x|y)$**:** The posterior probability for the sample x given the class y

The pseudo-code of first version of parallel AdaBoost for GMM is shown below:

Multiclass AdaBoost for GMM training in Parallel (Version 1)
For each iteration t: 1. For each class c in parallel (1.a) $t = 0$: Initialize $D_t(x, y) = \frac{1}{k-1}$, $y \neq c$, else 0 (1.b) $t > 0$: Update $D_t(x, y) = \frac{D_{t-1}}{Z_{t-1}} * \beta_{t-1}^{\frac{(1+h(x,c)-h(x,y))}{2}}$ (1.c) weight sample x as $w_t(x) = \sum_y D_t(x, y)$ (1.d) $t = 0$: train GMM for the class c with equally sample weights (1.e) $t > 0$: train GMM for the class c with sample weights $w_{t-1}(x)$ (1.f) compute unnormalized pseudo-loss $\varepsilon_t^c = \frac{1}{2} \sum_j \sum_y D_t(1 - h(x, c) + h(x, y))$ (1.g) store the class normalizer $Z_c = \sum_j \sum_y D_t(x_j, y)$. 2. Sync GMMs from all C classes 3. For each class c in parallel: do classification: compute $h(x\|y)$ for every pair of (x, y) 4. Sync when all c jobs done (4.a) Compute total normalizer $Z_t = \sum_c Z_c$. (4.b) Compute total pseudo-loss $\varepsilon_t = \sum_c \frac{\varepsilon_t^c}{Z_t}$ (4.c) Classifier weight $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

At the first iteration, the mislabel distribution matrices $D(x, y)$ are initialized

equally for all other classes $y \neq c$ for a particular class c (step 1.a)

$$D(x,y) = \begin{cases} \frac{1}{N(C-1)} & \text{if } y \neq c \\ 0 & \text{if } y = c. \end{cases}$$

The initialization formula shows that for a particular class c the mislabel distribution value $D(x,c)$ always stays at 0, while the value $D_t(x,y)$ for any other class y represents level of error that the classifier makes for any pair (x,y) at iteration t.
In step (1.b) the mislabel distribution value will be increased or decreased relatively during training based on the classification results.

$$D_t(x,y) = \frac{D_{t-1}}{Z_{t-1}} * \beta_{t-1}^{\frac{(1+h(x|c)-h(x|y))}{2}} \tag{6.6}$$

where $h(x|y)$ is the posterior probability for the sample x given the class y. We can see that when the likelihood value of $h(x|y)$ is relatively "large" compared to the likelihood of the truth $h(x|c)$, or the sample is classified incorrectly, the distribution $D_t(x,y)$ will increase relatively compared with other samples in the class since β_t is positive and less than 1. By adjusting the value of weights D, AdaBoost minimizes the loss function.

Only in the initialization iteration, the GMM are trained with equal weight (1.d). They are re-trained with a new set of samples' weight in step (1.e).

$$w_t(x) = \sum_y D_t(x,y). \tag{6.7}$$

After all the GMM parameters are retrained, they need to be merged at sync step (2). The merged GMM for all classes are distributed to all processes in the next iteration, and the newly trained GMMs are then used to perform classification on the whole training data in step (3). Besides merging the models, the sync step also normalizes the samples' weight cross all classes in step (4).

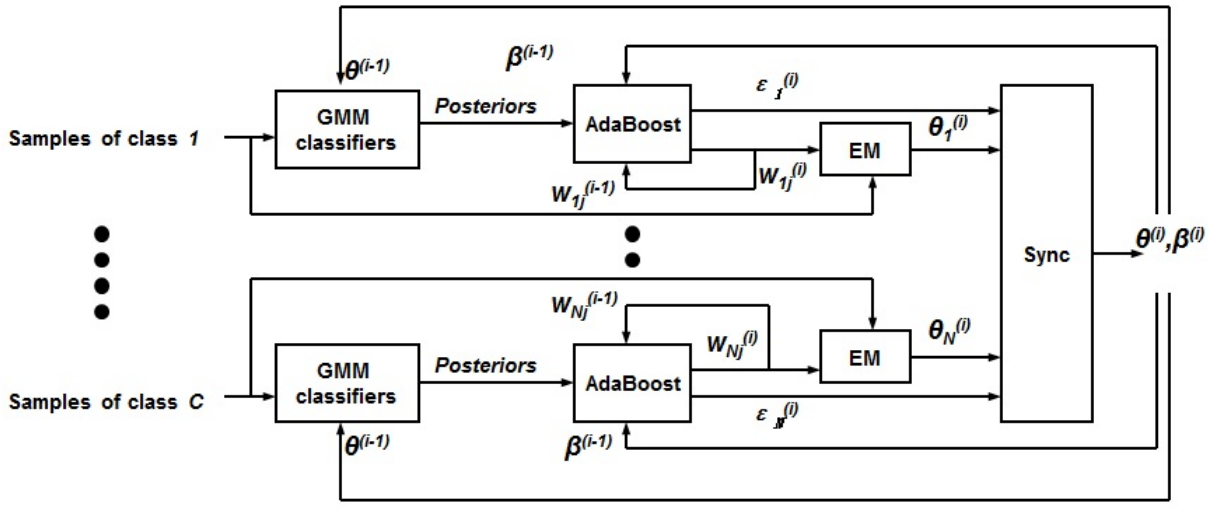

Figure 6-2: AdaBoost for GMM
(N classes in parallel)

Compared to the serial version where the whole training AdaBoost/GMM is handled by a single process, the first version of parallel AdaBoost reduces the total elapse time by splitting the computationally expensive steps into parallel processes.

6.3.3 Parallel AdaBoost (version 2)

The first version still needs four steps with two synchronizations in between. The sample weights after having been estimated in the previous iteration are needed to store to the disk and read back for GMM training in the next iteration, since each job for a particular class can be ended up in a different computer node for step 1 and 3, and the hosts may need to read the training data again in step (3). The two sync steps, which wait for all the jobs to be completed, can lead to long waiting times for each training iteration.

To reduce the synchronization time between the processes, the parallel AdaBoost is modified so that it only needs one sync step at the end of the iteration. Compared to the first version, the second version moves the classification step to the begining

Multiclass AdaBoost for GMM training in Parallel (Version 2)

1. For each class c in parallel:
   (1.a) $t > 0$: do classification (or posterior computation) using GMMs from previous iteration
   (1.b) $t = 0$: $D_t(x, y) = \frac{1}{k-1}$, $y \neq c$, else 0
   (1.c) $t > 0$: $D_t(x, y) = \frac{D_{t-1}}{Z_{t-1}} * \beta_{t-1}^{\frac{(1+h(x,c)-h(x,y))}{2}}$
   (1.d) weight sample x as $w_t(x) = \sum_y D_t(x, y)$
   (1.e) train a new GMM for class c with weighted samples
   (1.f) compute unnormalized pseudo-loss $\varepsilon_t^c = \frac{1}{2} \sum_j \sum_y D_t(1 - h(x, c) + h(x, y))$
   (1.g) store the class normalizer $Z_c = \sum_j \sum_y D_t(x_j, y)$.
2. Sync when all c jobs completed
   (2.a) Compute total normalizer $Z_t = \sum_c Z_c$.
   (2.b) Compute total pseudo-loss $\varepsilon_t = \sum_c \frac{\varepsilon_t^c}{Z_t}$
   (2.c) Compute classifier weight $\beta_t = \frac{\varepsilon_t}{1-\varepsilon_t}$

of the training. During the training in each class, the unnormalized pseudo-loss ε_t^c is computed and is normalized afterwards, which helps the procedure to be completed in parallel without the necessity of any synchronization step in the middle of each iteration. The total pseudo-loss will be normalized after the value of the total normalizer is computed.

6.3.4 GMM Training for Weighted Samples

The samples' weight, which were calculated based on the sum of all sample misclassified distributions over all classes, are used in the GMM training. For each class, a GMM is trained using the maximum likelihood (ML) criteria.

The Gaussian parameters (means, variances, and mixtures) are estimated by using the clustering algorithm K-means and EM. At first, the K-means algorithm is applied to all the samples belonging to the class. In the initialization step, k random samples are selected to be the means. The data is clustered in such a way that each sample belongs to the cluster with the nearest mean. K-means decides if the sample x_n

belongs to the cluster k by r_{nk}:

$$r_{nk} = \begin{cases} x_n \text{ belongs to cluster } k & \text{if } k = \arg\min_j ||x_n - \mu_j||^2 \\ \text{No} & \text{otherwise} \end{cases}$$

where μ_j is the mean of class j.

With an extremely low number of clusters, K-means may model the data poorly. On the other hand, a very high number can cause an over-fitting problem. The Bayesian information criterion (BIC) [78][81] is used to estimate the "correct" K. BIC maximizes the likelihood and also penalizes the model complexity at the same time. The Bayesian information criterion (BIC) rule selects the order to minimize

$$BIC = -2 \ln p_n(y, \hat{\theta}^n) + n \ln N \tag{6.8}$$

where n is the order of the model, and N is the training sample size.

During the K-means training, there are chances that the K-means algorithm is stuck in a local optimal solution. To avoid the local optimal issue, the clusters need to be re-arranged by using three operations [54].

- **Pruning:** A single Gaussian with too low number of samples is removed and merged with another one. This avoids Gaussians with extremely small variances.
- **Merging:** The Gausians are selected and merged to reduce to get to the desired number. The Gaussians are chosen to merge with the smallest decreasing in the likelihood.
- **Splitting:** The Gaussians with a disproportionately large percentage of the samples are split. The initial estimate for the two Gaussians are made by dividing the dimension with the highest variance.

By using these three operations, K-means can rearrange the Gaussians in a class, either in one fast step or several slow steps, thus ending with the desired number of

Gaussians.

After the K-means step is completed, the clusters for each class are then used as a starting point for EM training. Ultimately, after updating the samples' weight in each AdaBoost training iteration, the whole K-means/EM training needs to run through all the training samples to generate a new set of GMMs. However, in practice the K-Means only needs the very first AdaBoost iteration, when there are not sets of GMM available. And from the second iteration, the models from the previous iteration are re-used for EM training with the new set of weights. This short cut helps to reduce the whole training time significantly while there is no side effect in terms of loss of model accuracy. During the EM training, the variances and mixtures are updated after the means, even though when deriving the EM algorithm only the means are updated.

1. Estimation: Using the current parameters $\theta^{(t)} = \{\mu_j^{(t)}, \Sigma_j^{(t)}, m_j^{(t)}\}$, the expected $E[z_{ij}]$ are computed as:

$$E[z_{ij}]^{(t)} = \frac{m_j^{(t)} \frac{1}{\sqrt{2\pi\sigma_j^{(t)2}}} exp(-\frac{1}{2\sigma_j^{(t)2}}(w_i x_i - \mu_j^{(t)})^2)}{\sum_{n=1}^{k} m_n^{(t)} \frac{1}{\sqrt{2\pi\sigma_n^{(t)2}}} exp(-\frac{1}{2\sigma_n^{(t)2}}(w_i x_i - \mu_n^{(t)})^2)}. \tag{6.9}$$

2. Maximization: Using the expected $E[z_{ij}]^{(t)}$, all the parameters $\theta^{(t+1)} = \{\mu_j^{(t+1)}, \Sigma_j^{(t+1)}, m_j^{(t+1)}\}$ are updated:

$$m_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)}}{\sum_{j=1}^{k}\sum_{i=1}^{N} E[z_{ij}]^{(t)}}, \tag{6.10}$$

$$\mu_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)} w_i x_i}{\sum_{i=1}^{N} E[z_{ij}]^{(t)}}, \tag{6.11}$$

$$\Sigma_j^{(t+1)} = \frac{\sum_{i=1}^{N} E[z_{ij}]^{(t)} (w_i x_i - \mu_j^{(t+1)})(w_i x_i - \mu_j^{(t+1)})^T}{\sum_{i=1}^{N} E[z_{ij}]^{(t)}} \tag{6.12}$$

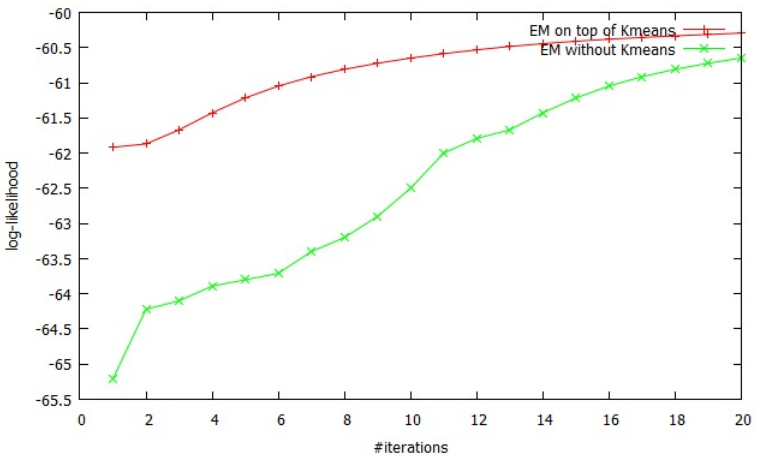

Figure 6-3: GMM/EM training with and without K-means initialization: The training with K-means converges faster.

where the weight of a sample x is the sum of all the mislabel distributions:

$$w_t(x) = \sum_y D_t(x, y). \tag{6.13}$$

In the first step, GMM training could be done by EM without K-means initialization. However, our experiment shows that the training converges faster and is more stable if EM is applied on top of K-means. To verify the effectiveness of K-means initialization for EM, we carried out a simulation on a set of data. In the first case, the EM is randomly initialized, while the K-means clusters are run before EM in the second case. Figure 6.3 illustrates the benefit of K-means in GMM training. The output from K-means including the mean of each cluster and covariance of the data points associated with cluster are used to initialize EM for the GMM training. From the second AdaBoost iteration, GMM training skips K-means and goes directly to EM. The Gaussians from the previous iteration are used as the initialization for the EM training. Figure 6.4 illustrates that by re-using the Gaussains for EM, the number of EM training iterations reduces by half and the log-likelihood is even better.

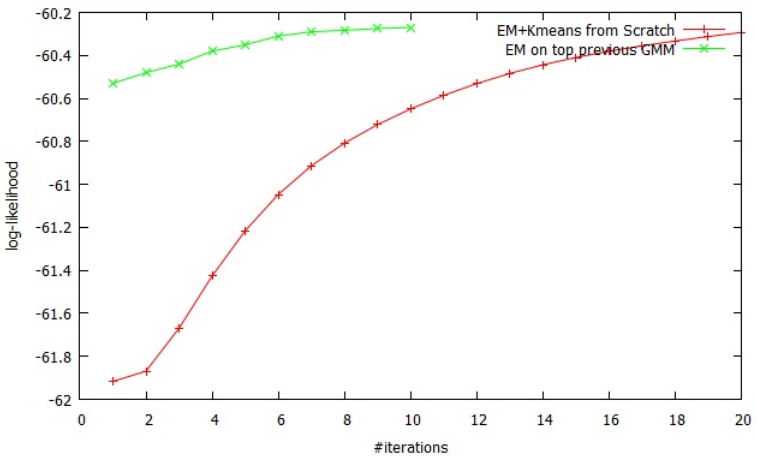

Figure 6-4: The effect of re-using the GMM from second iteration instead of K-means/EM from scratch

6.3.5 Hierarchical AdaBoost

For most GMM-based state-of-the-art acoustic model training, the data sets with thousands of hours are usually divided into smaller subsets. The model parameters are trained on each subset and synchronization jobs merge those parameters. The parameters are then distributed to the jobs in the next iteration. This acceleration technique helps to reduce the whole training time. In order to extend the training for AdaBoost to much bigger training sets, we need to apply a similar strategy. Instead of processing the training data of a whole training class, each job can process a smaller subset of the training samples for the class. The effect of such data division needs to be measured so that we do not lose classification accuracy.

As described in the previous section, the whole training for the group of frames from the same class only depend on the classification results from that particular class, and the total error can be synced up at the end. That leads to process the whole training in parallel in multiple jobs to make the whole training time reduced significantly. The whole AdaBoost for the GMM training procedure is shown in figure 6.5. We can then divide the multiclass AdaBoost training procedure into two stages. The first stage will be done in k parallel processes, for each class, in which the

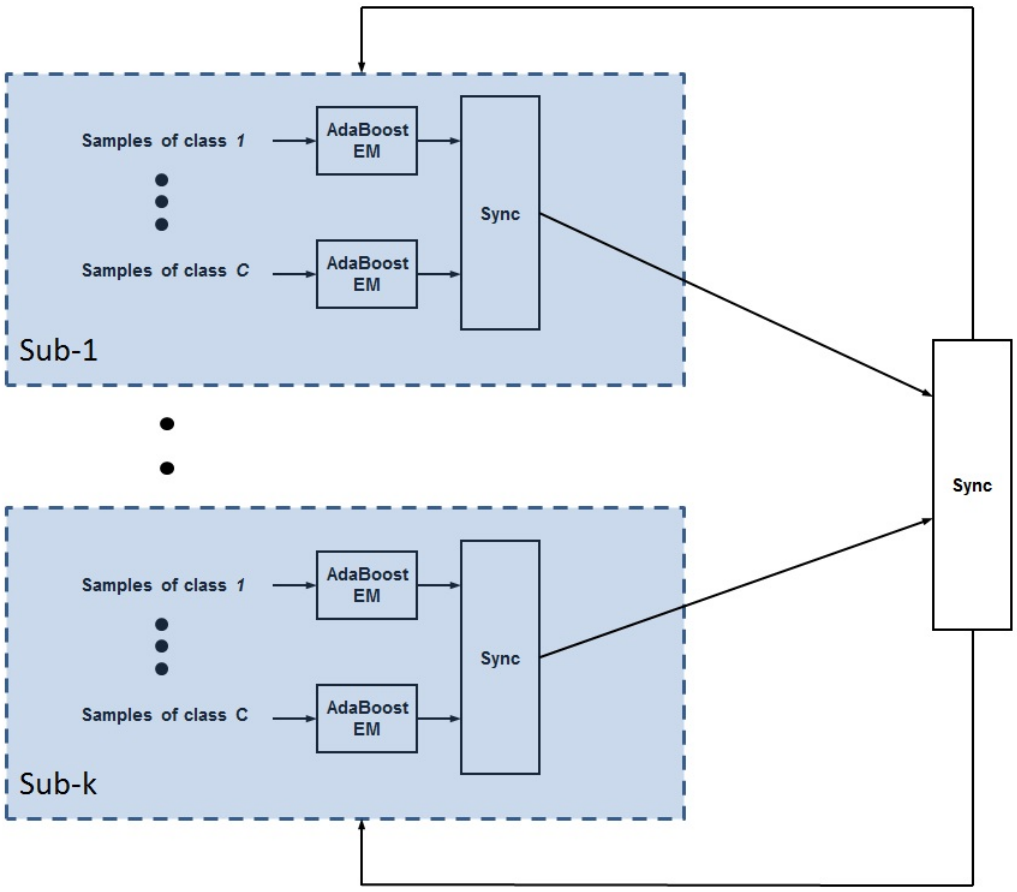

Figure 6-5: Hierarchical AdaBoost - Training data splitting to multiple subsets where each subset is split more to class jobs in parallel. There are two levels of GMM synchronization.

algorithm goes over only the data subset of that class to estimate the phoneme-specific GMM and saves relevant information for the calculation of the total pseudoloss in the second stage.

After all class jobs for the first iteration are done, all the outputs are merged into a single parameter file, and that file will be used in the next iteration by all the jobs to do the classification. At the synchronization step, the same GMM set for a particular class from multiple sub jobs are merged. For each step, the merging operation selects a pair of Gaussians that leads to the least increasing in the total likelihood. It loops over the set of Guassians until converged, which is determined when the designated

number of Guassian reaches or the loss of likelihood is higher than the threshold. For a pair of Gaussians $\{\mu_1, \Sigma_1\}$ and $\{\mu_2, \Sigma_2\}$, the merged parameters are computed as follows.

The merged mean is the individual means weighted by the mixture weights

$$\mu_{merged} = \frac{n_1\mu_1 + n_2\mu_2}{n_1 + n_2} \tag{6.14}$$

and, the merged variance has a form of

$$\Sigma_{merged} = \frac{n_1}{n_1 + n_2}\Sigma_1 + \frac{n_2}{n_1 + n_2}\Sigma_2 + \frac{n_1 n_2}{(n_1 + n_2)^2}(\mu_1 - \mu_2)^2 \tag{6.15}$$

where n_1 and n_2 are mixture weights for the first and second Gaussians, respectively.

6.3.6 Computation complexity

AdaBoost training requires very high computational demand. For each iteration, the posterior probabilities $h_t(x|y)$ for all pair training samples and classes need to be computed.

To estimate the computation complexity, let us define the training parameters:

- **N:** Number of training samples
- **T:** Number of training iterations
- **C:** Number of classes
- **K:** Number of Gaussians per class
- **S:** Number of data subsets
- i_1**:** Number of K-means iterations per AdaBoost iteration
- i_2**:** Number of EM iterations per AdaBoost iteration

- **L:** The feature vector length

The total running time to compute the posteriors is $O(TNCKL)$. The K-means/EM for GMM training requires $O(T(i_1 + i_2)NKL)$. The total running time is then $O(TNCK + T(i_1 + i_2)NKL)$. By re-using the GMM from the previous iteration, the total running time will reduce to $O(TNCK + Ti_2NKL)$ since the K-means only needs at the first iteration.

For a training data set with 50 hours of audio, it will take a single computer about 200 days to complete. The training time increases to 8000 days for 2000 hours of audio data. The parallel and hierarchical AdaBoost versions take advantage of the distributed computer system to reduce the training time by factors of C and CS, respectively. The total running time becomes feasible in a couple of days from thousands of days.

Amount of data	1-process	Parallel	Hierarchical
50 hrs	∼200 days	∼4 days	∼4 days
2000 hrs	∼8000 days	∼160 days	∼4 days

Table 6.1: The Estimate of Training Time Comparison for different AdaBoost Versions for different amount of training data

6.4 Feature Generation

After the training is complete, the GMM parameters $\theta_j = (m_j, \mu_j, \Sigma_j)$ for all classes and all iterations and all iteration classifier weights $\{\beta\}$ are then used to generate the posterior features. Given an input feature vector x the final posterior for a class j and T iterations has a form of

$$H(x|j) = \frac{\sum_{t=1}^{T} \log \frac{1}{\beta^{(i)}} h(x|\theta_j^{(i)})}{\sum_{t=1}^{T} \log \frac{1}{\beta^{(t)}}} \tag{6.16}$$

while the iteration hypothesis is computed by

$$h(x|\theta_j^{(t)}) = \frac{p(x|\theta_j^{(i)})}{\sum_{c=1}^{C} p(x|\theta_c^{(t)})} \tag{6.17}$$

with the likelihood $p(x|\theta_j^{(it)})$ for k GMM components has a form of

$$p(x|\theta_j^{(t)}) = \sum_{l=1}^{k} m_l \mathcal{N}(x|\theta_l). \tag{6.18}$$

The AdaBoost feature vector for a given frame x is

$$\begin{bmatrix} \log(H(x|1)) \\ \log(H(x|2)) \\ . \\ . \\ \log(H(x|C)) \end{bmatrix}$$

where C is the number of classes.

6.4.1 Classification Results

The frame accuracy rate (FAR) is used to evaluate AdaBoost classification. Given a feature frame x_i with the truth label of c, the frame is classified correctly if its final posterior $H(x_i|c)$ is the largest one for all classes. And the frame accuracy rate is the ratio between the number of correct frames to the total number of frames.

$$FAR = \frac{\sum_i^N Corr_i}{N} \tag{6.19}$$

where

$$Corr_i = \begin{cases} 1 & \text{if } H(x_i|c) = max(H(x_i|1), .., H(x_i|C)) \\ 0 & \text{else} \end{cases}$$

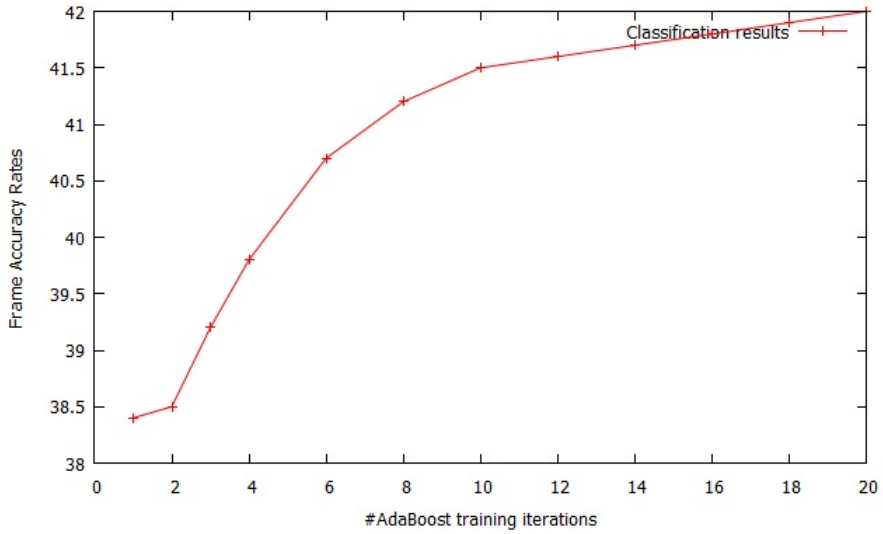

Figure 6-6: Classification results frame accuracy rates over number of AdaBoost training iterations with 512 GMMs.

The classification accuracy improves faster in the first iterations and then converges at around 20 iterations. The accuracy rates are reduced by at most 4% absolute (or more than 10% relative). The improvement of classification accuracy over the number of iterations is shown in Figure 6.6.

To measure the effect of the number of GMM components on the classifying results, we also tried 1024 and 2048-component GMMs. On the training set, the accuracy rate improved by 5% absolute when doubling the number of GMM components. The rate improved less than 1% absolute on the cross-validation set.

No. GMM Components	Training	Cross Validation
512	44.1	42.0
1024	49.3	42.6
2048	54.2	42.8

Table 6.2: Frame Accuracy Rates Over Number of GMM Components.

6.4.2 Error Analysis

To have a better understanding of the classification results, the confusion matrices (CM) are generated by quantifying the percentage of one phoneme classified by another one. The most confusable classifications for all phonemes are shown in the tables 6.3 to 6.6. The confusion matrices are broken down to vowel and consonant groups. Only the confusions higher than 10% are shown in those tables. The diagonal elements of the matrix represent the percentage of correct classification (hits), while the off-diagonal items are the misclassification rates. The results at the 20^{th} iterations show less confusion than the first iteration. The most confusable phonemes mostly have similar pronunciations to the truth phonemes such as AX and IX, P and F, S and Z, IX and IH.

Since the truth labels were generated by selecting the top one posterior probability of the Viterbi alignment, it is possible that a number of frames are biased to the baseline AM systems. There are frames that overlap with the previous or next phoneme. Frames within 10 ms of the boundaries are removed from the measurements in [38].

	AX	AA	OH	IX	EH	AXR	IH	ER	AE
AX	**47**			17					
AA		**31**	16						
OH		12	**37**						
IX				**18**			12		
EH					**31**				18
AXR						**30**		14	
IH				12			**32**		
ER								**42**	
AE					11				**40**

Table 6.3: The 1^{st} iteration: Most confusable vowel phonemes during classification.

	AX	AA	OH	IX	EH	AXR	IH	ER	AE
AX	**51**			21					
AA		**35**	15						
OH		12	**41**						
IX				**21**			10		
EH					**34**				17
AXR						**35**		14	
IH				12			**35**		
ER								**47**	
AE					11				**41**

Table 6.4: The 20^{th} iteration: Most confusable vowel phonemes during classification.

	P	F	R	N	M	S	V	SH	Z	B	ZH	TH	Y	CH
P	**32**	12												
F	13	**38**												
R			**41**											
N				**40**	11									
M				11	**35**									
S						**51**			13					
V							**39**			11				
SH								**57**		14				
Z						18			**44**					
B										**36**				
ZH								21			**42**			
TH		15										**25**		
Y													**46**	
CH								18						**40**

Table 6.5: The 1^{st} iteration: Most confusable consonant phonemes during classification

	P	F	R	N	M	S	V	SH	Z	B	ZH	TH	Y	CH
P	**36**	10												
F	8	**40**												
R			**44**											
N				**43**	10									
M				11	**41**									
S						**59**			16					
V							**42**			11				
SH								**65**		14				10
Z						21			**49**					
B							10			**38**				
ZH								21			**46**			
TH		15										**25**		
Y													**49**	
CH								18						**42**

Table 6.6: The 20^{th} iteration: Most confusable consonant phonemes during classification

6.5 Summary

In this chapter, the implementation for AdaBoost training was presented. To handle a large system with billions of training samples, parallel and hierarchical AdaBoost algorithms were proposed and implemented. During GMM training, K-means is only used in the very first AdaBoost training when there is no initial model for EM. From the second training, EM re-uses the GMM from the previous iteration instead of re-training K-means/EM from scratch. The parallel acceleration tricks help to reduce the training time from hundreds of days to less than one week. The frame accuracy rates are improved by more than 10%, when compared the 20^{th} iteration with the first one.

Chapter 7

AdaBoost In Large Vocabulary ASR

7.1 Introduction

In Chapter 6, the classification results for AdaBoost were shown and analyzed. In this Chapter, the AdaBoost features are used to train ASR systems. To establish a baseline for comparison, we set up an ASR system with PLP features for an English broadcast news (BN) system. We also use the posterior probability features derived by a Multi-Layer Perceptron (MLP) neural network as another baseline system. Even though AdaBoost features alone could be used in ASR system, they are still weaker than the standard features. Other studies have also suggested that combining probabilistic features with the traditional features can be useful. For use in an acoustic model (AM), the combined features are usually projected by transforms to reduce the dimension and reserve the discriminative power of the features. In this work, region dependent transform (RDT) is used on top of the combined features in the very large vocabulary ASR system for Arabic.

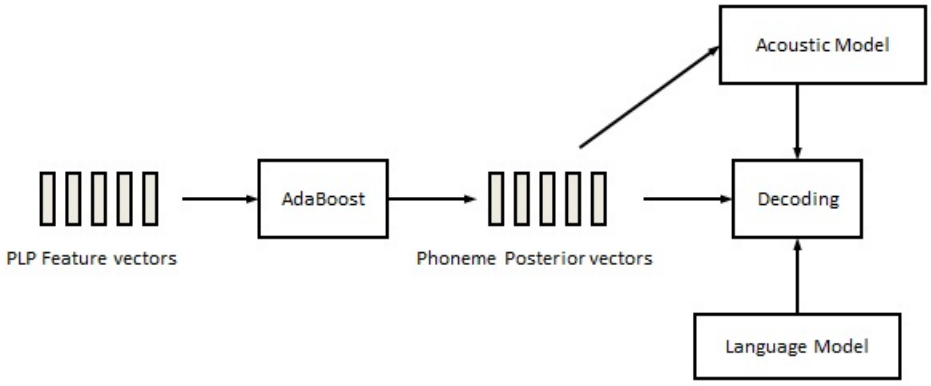

Figure 7-1: AdaBoost feature in ASR

7.2 AdaBoost in English ASR Systems

The acoustic model (AM) training data for this ASR English system comprises 50 hours of random selection from a 1300-hour system [55]. One development test set consists of 3 hours of CNN news shows aired in 2004. The baseline ASR system uses the standard PLP feature vectors. In this system, the standard 60-dimensional feature vectors comprising the 14 base cepstral and the normalized energy together with their first, second, and third derivatives, are reduced to 46 dimensions using Linear Discriminant Analysis (LDA) and then are decorrelated using maximum likelihood linear transform (MLLT). The language model (LM) is trained with Katz smoothing [35] on 1 billion-word text data.

7.2.1 Using AdaBoost Feature in ASR

The AdaBoost feature with each dimension as the logarithm of the posterior probability for each frame given a class are used in place of the standard features for acoustic model (AM) training and decoding. In the first set of experiments, AdaBoost features are used for AM training and decoding without any transform. The LDA and MLLT transforms are then applied on the features before training. The results of that set of

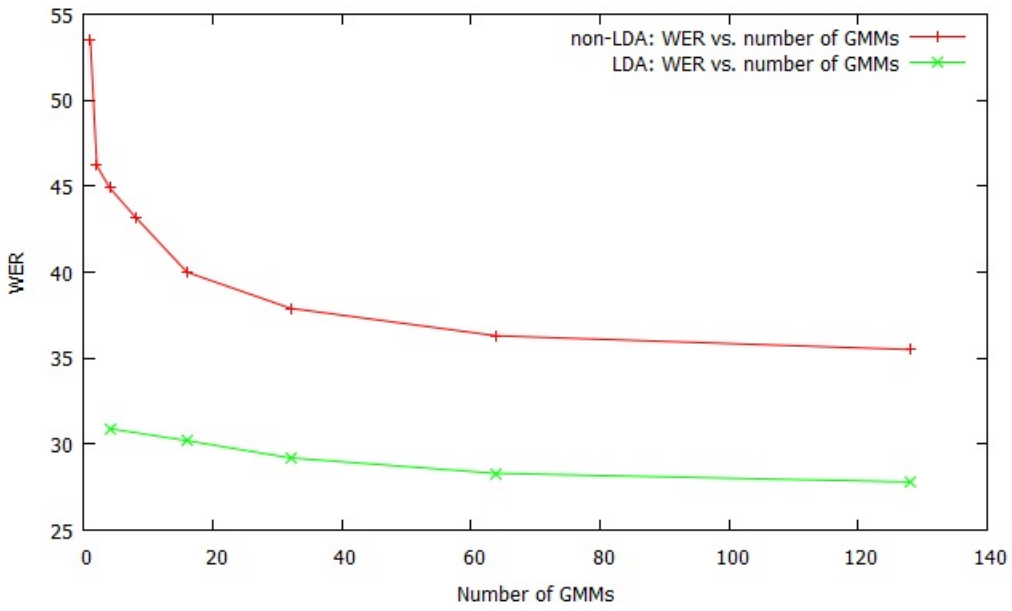

Figure 7-2: Results when applying LDA on AdaBoost features: WER versus number of GMM per class

experiments are illustrated in the figure 7.2. The effect of more Gaussians on WER is smaller when LDA and MLLT are applied. And, the transforms really help to reduce WER since all the AdaBoost feature dimensions are not totally un-correlated.

Table 7.1 shows the result with a different number of GMM components used for AdaBoost. When increasing the number of GMM components from 512 to 1024 and 2048, WERs are reduced by 2% to 4% (0.5% and 1.0% absolutely.)

System	**WER**
AdaBoost 512-component GMM	24.6
AdaBoost 1024-component GMM	24.1
AdaBoost 2048-component GMM	23.6

Table 7.1: Decoding Results with Different Numbers of GMM Components for AdaBoost Training.

The experiments with AdaBoost features were carried using different numbers of phonetic classes. Instead of using 50 phoneme classes as in the first set of experiments above, we increased the number of phonetic classes to 99 and 300. We replaced the

phoneme classes by using clustered allophones that are phonemes in specific phonetic contexts. When increasing the number of classes from 50 to 99, the WER of the new AdaBoost features was reduced by 1%. The ASR performance was improved further 3% (or 0.9% absolute) when the number of phonetic classes was increased to 300. These three results are in table 7.2.

No. classes	**WER**
50	24.6
99	24.3
300	23.4

Table 7.2: Decoding Results with Different Numbers of Classes.

Another baseline ASR system to compare with the AdaBoost features uses phoneme posterior probability features derived from a Multi-Layer Perceptron (MLP) neural network. The WER of this system is shown in the last row of Table 7.3. This is the typical performance (with some degradation) relative to standard PLP features when the MLP features are used in place of the PLP features as reported by all ASR research sites, such as LIMSI [18] and CU [60].

The AdaBoost model uses 512-component GMMs trained with 20 iterations. To compare to the baseline PLP system, the 50-dimensional AdaBoost features (representing the 50 phoneme posteriors) are reduced to 46 dimensions using LDA (as in the case of the PLP features) and then decorrelated using MLLT. Similar to the situation of the MLP features, the AdaBoost features produced higher WER than the baseline PLP system (24.6% versus 21.1%). However, the AdaBoost features are 0.8% absolute better than the MLP features (24.6% versus 25.4%). Again, it has been shown that probabilistic acoustic features (such as MLP) helped only if they have been used in addition to PLP features within a sophisticated concatenation or a combination framework [51].

System	WER
PLP	21.1
AdaBoost	24.6
ML P	25.4

Table 7.3: Decoding Results for PLP, MLP, and AdaBoost Features Alone.

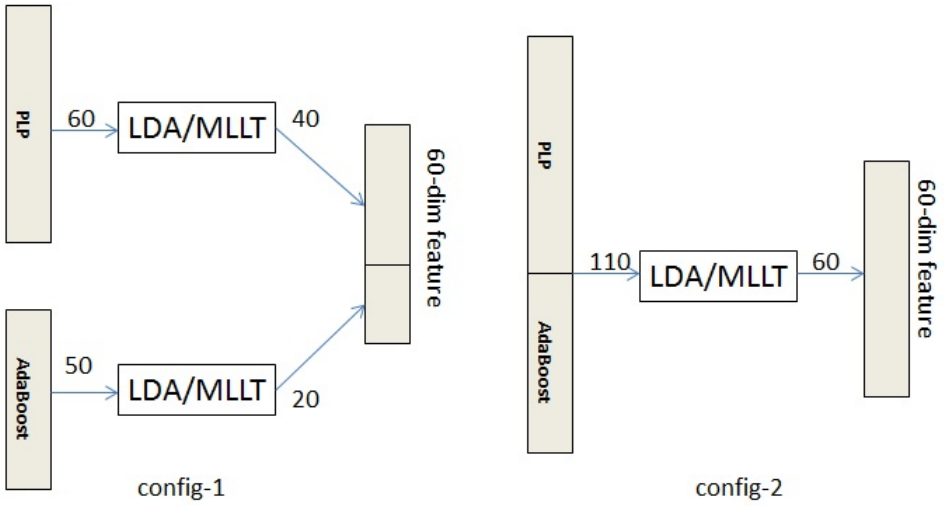

Figure 7-3: Different approaches for AdaBoost and PLP Feature Concatenation

7.2.2 Feature Combination

The ASR performance of the AdaBoost features is still behind the traditional PLP features, as observed in other works in the literature. However, the probabilistic features with their discriminative power can be combined with the traditional features to train the acoustic model (AM). There are multiple ways of combining the features, and the two most obvious are shown in the Figure 7.3. In the left hand side, the first approach, config-1, has the feature transformed and combined. The second approach, config-2, concatenates the features before transforming them.

The combined feature system outperforms the AdaBoost system. And the first configuration is slightly better than the PLP, while the second configuration is worse.

We then carried the unsupervised adaptation decoding experiments. The un-

System	UDEC	ADAPT
PLP	21.1	18.0
AdaBoost alone	24.6	20.1
AdaBoost+PLP config-1	20.9	18.3
AdaBoost+PLP config-2	22.1	18.7

Table 7.4: Decoding Results for PLP and AdaBoost Feature Combination

adapted hypotheses (UDEC) were used to update AM with constrained MLLR transforms [26]. The updated models were then used to re-decode 3 more passes (fast matching, lattice generation, and rescoring.) Even though, we observed a modest gain when combining AdaBoost and PLP features for the config-1, the gain is disappeared on the adaptation. The combined features even had higher WER compared to the PLP features alone. The results indicate that LDA and MLLT are not the best choice to transform the discriminative features, since they are not highly correlated to WER. LDA and MLLT treat all the feature dimensions in the same way, and the discriminative feature power could be lost through these transforms.

7.3 Region Dependent Transform for AdaBoost

The results from the Section 7.2 indicated that probabilistic features lost their discriminative power when they were combined with standard features through linear transforms as LDA/MLLT. Region dependent transform (RDT) [90] is used to project features by using a set of Gaussians to divide the acoustic space into multiple regions. In this section, RDT will be used to transform the combined AdaBoost and PLP features where the regions are the same as the classes in AdaBoost training. Since it is based on MPFE criterion, RDT helps to reserve the discriminative power of each feature while the redundant information is excluded during the dimension reduction. The transformed features are used to train SAT ML models, which are updated by MPFE training. We have built an AdaBoost system to use in our official Global Autonomous Language Exploitation (GALE) evaluation systems for Arabic.

7.3.1 Hierarchical AdaBoost Training

The acoustic models for Arabic are trained on 1700 hours of audio where the transcriptions of the data were processed through a light supervision data selection [58]. A baseline ML acoustic model was used to generate the Viterbi alignments, which are the truth labels for AdaBoost training. The hierarchical AdaBoost implementation, proposed in chapter 6, makes it feasible to process that large amount of data. The training jobs are run on a Sun grid engine (SGE) computer distributed system with hundreds of hosts. The grid distributes the jobs based on the availability of the hosts, the job's priority, and arriving time. An internal software was developed to utilize the SGE by allowing users to submit a list of jobs in parallel or make one job wait for a list of jobs done before being processed.

The whole 1700 hours of Arabic acoustic data is divided to 34 subsets with about 50 hours per subset. There are 36 phoneme classes for the Arabic system, so there are 1224 parallel jobs in total. After 36 single jobs of a particular subset are done, one sync job for that subset merges Gaussians and normalizing factors. The training iteration is then completed by a sync job to merge the 34 set of Gaussians after all the sync jobs for subsets are finished.

Table 7.5 below shows the training time for hierarchical AdaBoost algorithm and the estimate of the parallel approach where there is only one layer of process splitting. The training time was reduced by a factor of the number of the subsets, 34, given that there are enough computation resources available. The frame accuracy rate (FAR) for

Amount of data	**Parallel**	**Hierarchical**
50 hrs	∼4 days	∼4 days
1700 hrs	∼136 days	∼4 days

Table 7.5: Estimate Training Time Comparison for different AdaBoost Versions

the training and cross-validation set is shown in table 7.4. On the training sets, the FAR for the 50 hours set is lower than the 1700 hours set, but the larger set performs better on the cross-validation set. This indicates that the hierarchical AdaBoost works

well even with more parallel jobs, and the FAR on the cross-validation set shows that the large set of data makes the classifiers more robust.

No. GMM Components	Training	Cross Validation
50 hours	61.1	50.3
1700 hours	53.3	51.1

Table 7.6: Frame Accuracy Rates for the Training and Cross Validation sets Over Amount of Training Data

7.3.2 Region Dependent Transform

One AdaBoost feature vector with 36 dimensions of a particular frame concatenates with 9 frames of PLP features to form an input feature x_t with 161 dimensions for RDT training. The regions for RDT training are also the central phoneme of alignment labels, which were also used in AdaBoost training. Each region has its own transform matrix A_i. The output feature vector has a form of

$$y_t = \sum_{i=1}^{N} p(i|o_t) A_i x_t \tag{7.1}$$

where N is the number regions, and $p(i|o_t)$ is the AdaBoost posterior for the current feature frame o_t given the class i.

The RDT region matrix A_i is trained on the gradient descent method [90]. Given the concatenated features $Y = \{Y_1, ..X_R\}$ for R utterances and their corresponding references $\{W_1^{ref}, ..W_R^{ref}\}$, the objective function for RDT is based on MPE criterion

$$H_{\text{MPFE}}(Y, \lambda) = \sum_{r=1}^{R} \sum_{W_r} \frac{p(Y_r|W_r, \lambda)^{\beta} p(W_r) \epsilon(W_r, W_r^{ref})}{\sum_{W_r'} p(Y_r|W_r', \lambda)^{\beta} p(W_r')} \tag{7.2}$$

where $\epsilon(W_r, W_r^{ref})$ is a measurement of the phoneme frame accuracy rate for hypothesis W_r given the reference W_r^{ref}, and $\epsilon(W_h, W_{ref})$ is a percentage overlap between these two word sequences.

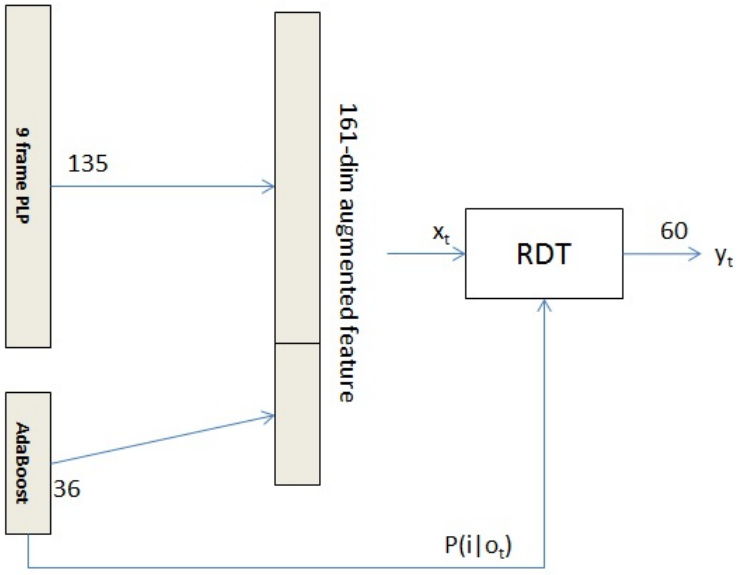

Figure 7-4: Region dependent transform (RDT) for AdaBoost and PLP features for Arabic: 36 regions for 36 phonemes.

$\lambda = \{\mu_1, .., \mu_k, \Sigma_1, .., \Sigma_k\}$ are a set of Gausian parameters trained on the concatenated features Y, and β is the exponent to control the contribution of acoustic and language scores.

The transform matrix A_i is updated by gradient decent using the derivative

$$\frac{\partial H_{\text{MPFE}}(Y,\lambda)}{\partial A_i} = \sum_{r,t} p(i|o_t^r) \frac{\partial H_{\text{MPFE}}(Y,\lambda)}{\partial y_t^t} x_t^{rT}. \tag{7.3}$$

That derivative of the MPFE objective function with respect to the feature y_t^t is expressed as

$$\frac{\partial H_{\text{MPFE}}(Y,\lambda)}{\partial y_t^t} = (\frac{\partial H_{\text{MPFE}}(Y,\lambda)}{\partial y_t^t})_\lambda + \frac{\partial H_{\text{MPFE}}(Y,\lambda)}{\partial \lambda} \frac{\partial \lambda}{\partial y_t^t}. \tag{7.4}$$

The first term is the derivative of the objective function with respect to y_t^t by fixing the λ where the second term is updated iteratively through the models and transform

updating.

The combined features with RDT are used for speaker-cluster-dependent ML-SAT training with constrained MLLR [26], as presented in chapter 2. The ML-SAT models and unigram language models (LM) are used to generate the lattices where the weak LMs help to make the lattices richer. The RDT ML-SAT models are updated by using the generated lattices for MPFE training [91]. The MPFE SAT models are used in the final stage of the decoding.

7.4 Building Arabic ASR systems

To participate in the Global Autonomous Language Exploitation (GALE) program's final evaluation, we have built AdaBoost system as one of multiple very large vocabulary ASR systems for Arabic. The systems have comparable performances, and the ROVER combination provides significant improvement compared to the "best" individual system. By comparing two ROVER combinations of many systems for with and without the AdaBoost system, the one with the AdaBoost system gave additional modest gain on top of a very strong baseline.

7.4.1 Lexical and Phonetic Modeling for Arabic Systems

In Arabic, compound words are generated by attaching affixes to the roots, which results a very large number of possible unique words. It is a big challenge to select a lexicon to have a good coverage given a random text. For example, a 65,000 Arabic lexicon has an out-of-vocabulary (OOV) rate of 5%, compared to 0.5% for the same size English lexicon. In ASR application, OOV words have no chance to be recognized, and each OOV word also causes recognition errors for its neighbor words. The second difficulty of Arabic ASR is to make the pronunciation for words since the short vowels are not written in most modern texts. The short vowels are diacritics placed above or below the letters, and they are necessary for generating pronunciations. To overcome

those issues, multiple systems with different lexical and phonetic modeling approaches have been built [56][85].

Word	**Prefix**	**Root**	**Suffix**
wktAbhm	w	ktAb	hm
Meaning	and	book	their

Table 7.7: One example of compound word in Arabic, written in Buckwalter format

Phonetic system: As described in [56][85], we have constructed a phonetic dictionary for Arabic by using the Buckwalter morphological analyzer [11] and manually-vocalized corpora. The Buckwalter analyzer generates all possible vowelization forms of a word by looking up from its dictionary. There are cases that the outputs from the analyzer are over generated, especially the long words. On the other hand, there are also words, which failed to generate vowelizations. The design of our word-based phonetic (P) system is a straightforward implementation of a typical ASR system. The recognition units are Arabic words, and each word is modeled by one or more sequences of phonemes of its pronunciation(s), which are looked up from a generated master dictionary.

Grapheme System: The word-based graphemic (G) system is similar to the phonetic system except that it does not use phonetic pronunciations and it has a much larger recognition vocabulary. The recognition units are also Arabic words but each word is modeled by exactly one sequence of letters of its spelling in Buckwalter format. Since the grapheme system just relies on the word's letters to make up the pronunciation, there is no issue of words without pronunciation like what was observed in the phonetic system.

The weakness in a grapheme system is that the pronunciations is not consistent with the way words are pronounced, and this could hurt ASR performance due to poor acoustic modeling. The grapheme system is cheaper in recognition since there is only one pronunciation per word, while on average it is four for phonetic system.

Data-driven morpheme system: The data driven morphemic system (M1) is

Word	**Grapheme**	**Phonetic**
wktAbhm	w-k-t-A-b-h-m	w-a-k-i-t-A-b-a-h-u-m
		w-a-k-i-t-A-b-h-u-m
		w-a-k-i-t-A-b-i-h-i-m
		w-a-k-i-t-A-b-u-h-u-m
		w-a-k-u-t-t-A-b-a-h-u-m
		w-a-k-u-t-t-A-b-h-u-m
		w-a-k-u-t-t-A-b-i-h-i-m
		w-a-k-u-t-t-A-b-u-h-u-m

Table 7.8: A Sample of Grapheme vs. Phonetic Pronunciations

a morpheme-based phonetic system [9]. Morphemes are determined through a simple morphological decomposition of the words without their context using a small set of affixes and a few rules. The compound words are decomposed to stem and affixes for both language and acoustic model training data. The affixes are glued back into the stem in the post-processing step in recognition.

The process also utilizes a list of 128,000 most frequent decomposable words that should not be decomposed (hereafter, referred to as a blacklist) since it has been shown that this really improves recognition performance.

By breaking the compound words into shorter units, which occurs more frequently, the morpheme based system reduces the OOV rate with a smaller lexicon, compared to the word based phonetic and grapheme systems above. The decomposition process also helps to lower the sparsity of the language training data.

Linguistically driven morpheme system: The linguistically driven morphemic system (M2) is another morpheme-based phonetic system, but the morphemes in this case are determined by a more sophisticated decomposition process. Briefly, the process consists of two steps. First, we use Sakhr's Arabic morphological analyzer [50] to decompose all AM and LM training data (of about two billion words.) Each word, if occurring more than once, can be decomposed into different sequences of morphemes depending on its contexts. Note that each word instance is decomposed into exactly

one sequence of morphemes of the form

$$morpheme = [prefix] + stem + [suffix] \tag{7.5}$$

where either the prefix or the suffix or both can be missing. Similar to the construction of the recognition units in system M1, we also utilize a blacklist of the 128,000 most frequently decomposable words.

7.4.2 Experimental Setup

The baseline for comparing the RDT(PLP+AdaBoost) system is the PLP system. Both systems use linguistically driven morphemes (M2). The training data for both language and acoustic models are decomposed to prefixes, stems, and suffixes.

Testing data: In this set of experiments, we used the official development (ad09) and evaluation (ae09) sets designed by NIST/LDC for the GALE program. Each set is about 3 hours long with broadcast news and broadcast conversation aired during 2009 on various television channels.

Language model training: The language model includes multiple n-gram components, which are trained on disjoint subsets of data with Kneser-Ney smoothing. The total amount of data includes 2.5 billion words of Arabic text data available to the GALE community. The sub N-gram components are then merged with the interpolation weights, which are optimized under an entropy based criterion using the EM algorithm. A 4-gram neural network LM [33] is used in N-best rescoring after the decoding process.

Acoustic model training: Nine successive frames of 14-dimensional and energy PLP features are concatenated, and then reduced the dimension to 60 by Linear Discriminant analysis (LDA) and maximum likelihood linear transform (MLLT). The speaker independent (SI) acoustic models are trained using the maximum likelihood (ML) criterion. The speaker cluster dependent (SAT) AMs are also trained on ML cri-

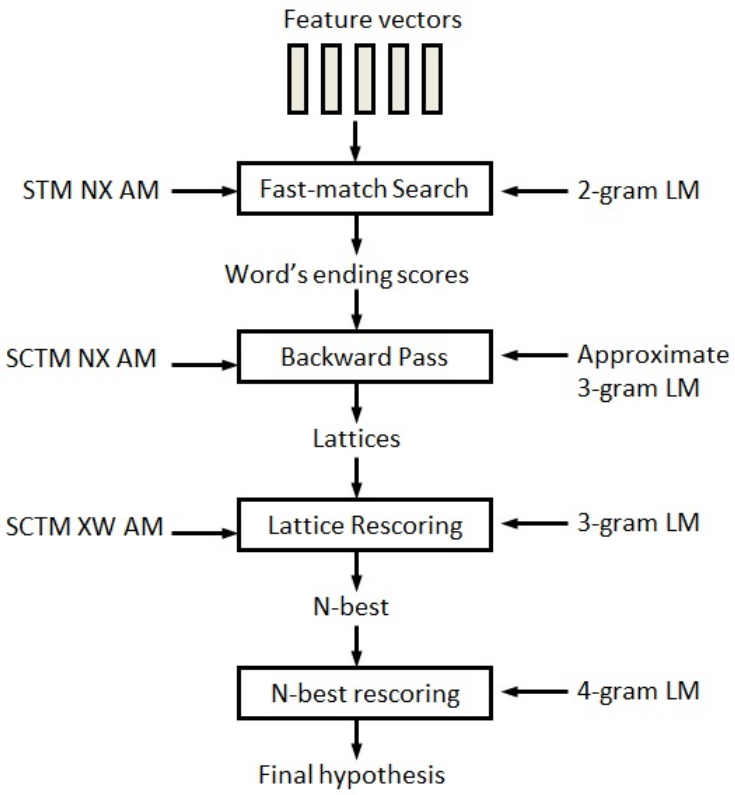

Figure 7-5: The decoding procedure in Byblos system

terion concatenated features after each 15-dimensional feature vector is transformed using the speaker cluster Constrained Maximum Likelihood Linear Regression (CMLLR) transform. The concatenated features are applied the LDA and MLLT to form 60-dimensional features, upon which are applied CMLLR for the second time. The ML models are used to generate the lattices for Minimum Phoneme-Frame Error (MPFE) training where both SI and SAT ML model parameters are updated.

Decoding: The decoding process is shown in Figure 7.5. The first pass of decoding is the fast match to mark the most likely word's ending at each frame using the state-tied-mixture (STM) AM and the bi-gram LM. By using the more specific models, state-clustered tide-mixture (SCTM) within word AM and the approximate tri-gram LM, the second pass of decoding generates the lattices based on the scores from the fast match pass. The lattices are then rescored by the cross-word SCTM AM and 3-gram LM to generate the N-best hypotheses. The N-best hypotheses are then re-ranked by using 4-gram LM to select the 1-best hypotheses. The 1-best hypotheses are then glued the affixes into their stems in a post processing step.

For the baseline PLP system, SI MPFE AM models are first used to decode the data, called un-adapted decoding pass. A speaker cluster step runs on each audio segment to find its closest speaker-cluster. The AM models for each speaker cluster on test data are updated using the hypotheses from the un-adapted decoding pass. The adapted MPFE models are then used to run three more passes of decoding. The un-adaption results for the PLP baseline system is used to adapt the model for RDT (PLP+AdaBoost) MPFE. The results in Table 7.9 shows that the RDT on

System	**ad09**	**ae09**
PLP	15.7	10.3
RDT(PLP+AdaBoost)	15.4	10.1

Table 7.9: Decoding results

PLP+AdaBoost features outperform the baseline systems by about 2% relative in terms of WER reduction for both development and evaluation sets.

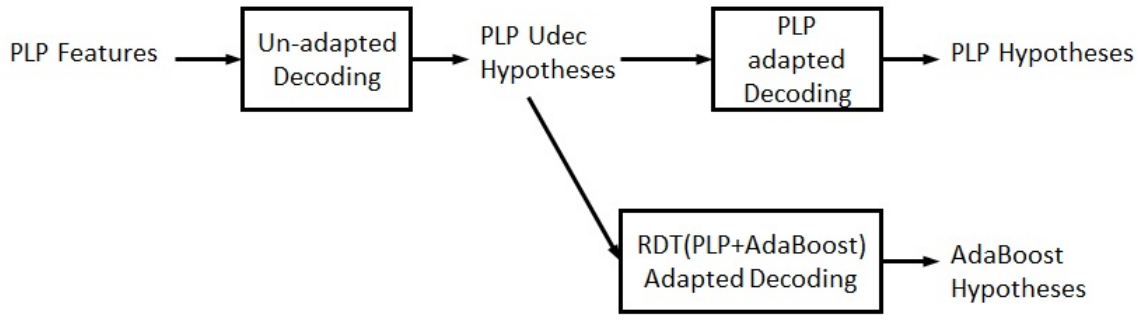

Figure 7-6: PLP and AdaBoost decoding processes

The results from our study [56] have shown that the most sophisticated morpheme systems outperform the word-based systems thanks to the lower OOV rates even with smaller lexicon sizes. The word-based grapheme and phonetic systems are still useful in system combination. To have a bigger impact on system combination by diverse but complementary systems, besides using different lexical and morphological units, we use the different input acoustic features such as PLP, MLP, and AdaBoost with and without RDT. Multiple systems are built and selected the best for the system combination. Compared to the best single system, the ROVER is 5-6% better. To quantify the effect of AdaBoost system, we perform two ROVER combinations for 11 systems without AdaBoost system to compare with the ROVER of 12 systems with the AdaBoost system. In all three sets, the gain of 1% in WER is obtained, given a very strong baseline of a 11-system combination.

ad09	**ae09**	**ad10**	Combination Configuration
13.2	8.5	12.1	ROVER-11 (without AdaBoost system)
13.1	8.4	12.0	ROVER-12 (with AdaBoost system)

Table 7.10: ROVER results for Arabic system

7.5 Summary

In this chapter, we presented the usage of AdaBoost features in ASR systems. At first, the AdaBoost features were used alone or concatenated with PLP features to compare with the standard features with LDA/MLLT transformation where the results of the combined features did not perform better than the baseline. The AdaBoost features are then transformed with RDT to train the SAT MPFE system. Compared to the baseline of the standard features, the gains were observed on all the test sets. Moreover, the AdaBoost system is also seen to contribute to the consistent gain in the ROVER combination even with a very strong baseline of 11 systems.

Chapter 8

Conclusions and Future Work

8.1 Conclusions

The most widely used ASR features over the past few decades have been PLP and MFCC. These features capture the signal spectrum and are not optimized for ASR accuracy. In this work, we have implemented AdaBoost based classifiers to transform the standard features to probabilistic features. The informative features were used as complementary information, together with the spectral features to improve the ASR performance.

AdaBoost algorithm is used to combine the GMM classifiers, which have been used successfully to model the HMM states for acoustic models. To speed up the training procedure for large amounts of data, we have proposed and implemented a hierarchical training strategy. The training data are split to reasonable number of subsets, and each subset is then divided to classes where each job only processes the data for one particular subset and class. This hierarchical strategy reduces the training time by a factor of a few hundreds.

To save the time during GMM training, the K-means is only used in the very first iteration where there is no initial model for EM. Moreover, the EM also re-uses the GMMs from the previous step instead of re-doing K-means initialization for every

training iteration. During training, the frame accuracy rates are used to evaluate the GMM/AdaBoost classifiers' performance. Compared to the "base" classifier of GMM, the AdaBoost for GMM has shown improvement in classifying results.

The AdaBoost features have been, then, used successfully to combine with spectal feature using RDT for ASR. Compared to the baseline of the standard features, the gains are observed on all the test sets. Moreover, the AdaBoost system also contributes consistent gains on the ROVER combination even with a very strong baseline ROVER of other 11 systems.

8.2 Potential Improvements

In this work, the discriminative features from AdaBoost are used as input for GMM/HMM by combining with baseline features, and transforming through linear or non-linear projections. And the discriminative power gained from the AdaBoost classifiers could be lost through the feature combination and the transform process. Instead, the posterior probabilities may be used directly as acoustic model scores for the decoding process [4]. This leads us to the following open studies to improve the usage of AdaBoost for ASR.

To use the class posteriors as the AM scaled scores, the number of classes for AdaBoost training should be increased to a comparable number as HMM states of the baseline acoustic models. Furthermore, The training needs to be fine-tuned with larger number of classes. For example, instead of the central phoneme without context dependent or HMM state, each class could represent a triphone or even quinphone cluster.

With more classes, the number of GMMs and parameters per GMM should be reduced. To make up for less number of parameters, the training could have more iterations. Another strategy to improve the AdaBoost training is to experiment with the classifier selection, which is to choose the best quality classifiers out of the all

iterations. With less number of GMM parameters, AdaBoost can have bigger impact on the final results with higher discriminative power.

With a reasonable number of classes as the number HMM states of the acoustic model, the AdaBoost posterior probabilities are scaled to likelihoods and integrated into the decoding systems. The acoustic model training with all the transforms could be replaced by the AdaBoost classifiers. The regular unsupervised adaptation could be tried as MAP adaptation for the iteration GMM parameters.

Appendix A

Training Parameters

A.1 Gaussian Variance is Biased in ML Training

The normal distribution has a form of:

$$p(x|\mu,\sigma^2) = \mathcal{N}(x|\mu,\sigma^2) = \frac{1}{(2\pi\sigma^2)^{1/2}} e^{-\frac{1}{2\sigma^2}(x-\mu)^2} \tag{A.1}$$

where μ is the mean, and σ^2 is the variance. The most common approach to estimate the distribution's parameters is to maximize (log) likelihood function for the whole training data set $X = \{x_i, .., x_N\}$.

$$\log p(X|\mu,\sigma^2) = -\frac{1}{2\sigma^2}\sum_{i=1}^{N}(x_i-\mu)^2 - \frac{N}{2}\log\sigma^2 - \frac{N}{2}log(2\pi). \tag{A.2}$$

Maximizing the log likelihood function with respect to μ, the mean is computed as the samples' mean:

$$\mu_{ML} = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{A.3}$$

And variance has a form:

$$\sigma^2_{ML} = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{A.4}$$

Assume that the data set $X = \{x_1, .., x_N\}$ are generated by the true distribution with μ, σ^2. We have

$$E[\mu_{ML}] = \mu. \tag{A.5}$$

$$\begin{aligned}
E[\sigma^2_{ML}] &= E[\frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2] \\
&= \frac{1}{N}E[\sum_{i=1}^{N} x_i^2 - 2N\mu^2 + N\mu^2] \\
&= E[x^2] - E[\mu^2] \\
&= (\sigma_x^2 - \mu^2) - (\sigma_\mu^2 - \mu^2) \\
&= \sigma_x^2 - \sigma^2_{(\sum_{i=1}^{N} x_i)} \\
&= \sigma_x^2 - \frac{1}{N}\sigma_x^2 \\
&= \frac{N-1}{N}\sigma_x^2
\end{aligned} \tag{A.6}$$

With a large enough number of samples of N, the ML variance will converge to the "true" one.

A.2 K-means Objective Function Decreases Monotonically

With $J^{opt}(K)$ be the optimal value of the K-means objective function J with K clusters [11].

$$J^{opt}(K) > J^{opt}(K+1). \tag{A.7}$$

We will prove the theorem by comparing K and $K+1$ clusters for the same set of training data. Assume that we could found K optimal clusters.

$$J^{opt}(K) = \sum_{i \in C_1} ||x_i - \mu_1||^2 + \sum_{i \in C_2} ||x_i - \mu_2||^2 + .. + \sum_{i \in C_K} ||x_i - \mu_K||^2. \tag{A.8}$$

Now we split one of the clusters optimally while keeping all other clusters unchanged. Without loss of generality, the last cluster K is split.

$$\begin{aligned} J^{C_K - split}(K+1) &= \sum_{i \in C_1} ||x_i - \mu_1||^2 + \sum_{i \in C_2} ||x_i - \mu_2||^2 + .. \\ &+ \sum_{i \in C_{K1}} ||x_i - \mu_{K1}||^2 + \sum_{i \in C_{K2}} ||x_i - \mu_{K2}||^2. \end{aligned} \tag{A.9}$$

Since the cluster K is split optimally to two sub-clusters $K1$ and $K2$, it is obvious that we have:

$$\sum_{i \in C_{K1}} ||x_i - \mu_K||^2 \geq \sum_{i \in C_{K1}} ||x_i - \mu_{K1}||^2. \tag{A.10}$$

$$\sum_{i \in C_{K2}} ||x_i - \mu_K||^2 \geq \sum_{i \in C_{K2}} ||x_i - \mu_{K2}||^2. \tag{A.11}$$

This is because the optimal value of $\sum_{i \in C_k} ||x_i - \mu_k||^2$ achieves when $\mu_k = \frac{\sum_{n=1}^{N} r_{nk} x_n}{\sum_{n=1}^{N} r_{nk}}$. We then have

$$J^{C_K - split}(K+1) < J^{opt}(K). \tag{A.12}$$

And the optimal value of $J^{opt}(K+1)$ must be better than the optimal value of $(K+1)$ clusters, which have only one that is fixed and the rest are fixed.

$$J^{opt}(K+1) < J^{C_K - split}(K+1) < J^{opt}(K). \tag{A.13}$$

Therefore, we have K-means objective function decreasing monotonically.

A.3 Model Order Selection Using Bayesian Information Criterion Rule

Maximum likelihood (ML) criterion is used widely to estimate the model parameters by maximizing the likelihood of the data given the models. However, ML criterion is not a good choice in model selection since the higher model order makes its likelihood lower; this is called overfitting. Bayesian information criterion (BIC) rule is a better choice for model selection when it reduces the likelihood and penalizes the model complexity at the same time.

Let y be the training data, and $p(y, \theta)$ be the probability density function (PDF) or likelihood function. Assume the $p_0(y)$ is the true PDF and $\hat{p}(y)$ is the estimate PDF. BIC aims to minimize KL (Kullback-Leibler) information function, or to get the estimate PDF as close as possible to the true one. The KL information function has a form:

$$D(p_0, \hat{p}) = \int p_0(y) \ln[\frac{p_0(y)}{\hat{p}(y)}] dy$$

$$\begin{aligned} &= E_0\{\ln[\frac{p_0(y)}{\hat{p}(y)}]\} \\ &= E_0\{\ln p_0(y)\} - E_0\{\ln \hat{p}(y)\}. \end{aligned} \tag{A.14}$$

Now we need to prove that $D(p_0, \hat{p})$ has distancy properties

$$\begin{aligned} D(p_0, \hat{p}) &\geq 0 \\ D(p_0, \hat{p}) &= 0 \text{ iff } p_0(y) = \hat{p}(y). \end{aligned} \tag{A.15}$$

At first we prove this by setting the derivative of the function to zero.

$$\begin{aligned} \ln x - x + 1 &\geq 0 \text{ for any } x > 0 \\ \ln x - x + 1 &= 0 \text{ iff } x = 1. \end{aligned} \tag{A.16}$$

Now using (A.14) with $x = \frac{\hat{p}(y)}{p_0(y)}$, we have:

$$\begin{aligned} D(p_0, \hat{p}) &= \int p_0(y) \ln[\frac{p_0(y)}{\hat{p}(y)}]dy \\ &= \int p_0(y)[-\ln \frac{\hat{p}(y)}{p_0(y)}]dy \\ &\geq \int p_0(y)[1 - \frac{\hat{p}(y)}{p_0(y)}]dy \\ &= \int p_0(y)dy - \int \hat{p}(y)dy \\ &= 1 - 1 = 0. \end{aligned} \tag{A.17}$$

And $D(p_0, \hat{p}) = 0$ iff $\frac{\hat{p}(y)}{p_0(y)} = 1$, or the estimate PDF $\hat{p}(y)$ is the the true on $p_0(y)$. Since $D(p_0, \hat{p}) = E_0\{\ln p_0(y)\} - E_0\{\ln \hat{p}(y) \geq 0$, so minimizing KL information is equivalent to maximizing the relative KL information

$$I(p_0, \hat{p}) = E_0\{\ln \hat{p}(y)\} \tag{A.18}$$

Because the true PDF $p_0(y)$ is unavailable and the estimate PDF $\hat{p}_n(y, \hat{\theta}^n)$ is monotonically increases with the model order n, the estimate PDF cannot be used to estimate the relative KL information. Another data set x with the same size and the same PDF with y for a cross validation purpose.

$$\begin{aligned} \ln \hat{p}(y) &= E_{\hat{\theta}_x}\{\ln p(y, \hat{\theta}_x)\} \\ &= \int p(y, \theta) p(\theta) d\theta. \end{aligned} \tag{A.19}$$

The second-order Taylor series expansion of $\ln p(y, \hat{\theta}_x)\}$ around $\hat{\theta}_x$ is

$$p(y, \hat{\theta}_x) \approx p(y, \hat{\theta}_y) e^{-\frac{1}{2}(\hat{\theta}_x - \hat{\theta}_y)^T J_y (\hat{\theta}_x - \hat{\theta}_y)}. \tag{A.20}$$

where J is the Fisher information.

Assume that $p(\theta)$ is flat around $\hat{\theta}$, the estimate PDF will have a form

$$\begin{aligned} \hat{p}(y) &= p(y, \hat{\theta}_y) p(\hat{\theta}_y) \int e^{-\frac{1}{2}(\hat{\theta}_x - \theta)^T J_y (\hat{\theta}_x - \theta)} d\theta \\ &= \frac{p(y, \hat{\theta}_y) p(\hat{\theta}_y) (2\pi)^{n/2}}{|\hat{J}|^{1/2}} \int \frac{1}{(2\pi)^{n/2} |\hat{J}^{1/2}|} e^{-\frac{1}{2}(\hat{\theta}_x - \theta)^T J_y (\hat{\theta}_x - \theta)} d\theta \\ &= \frac{p(y, \hat{\theta}_y) p(\hat{\theta}_y) (2\pi)^{n/2}}{|\hat{J}|^{1/2}} \end{aligned} \tag{A.21}$$

Hence, the relative KL information is estimated by combining (8.27) and (8.29)

$$\begin{aligned} \hat{I}(p_0, \hat{p}) &= \ln p(y, \hat{\theta}_y) + \ln p(\hat{\theta}_y) + \frac{n}{2} ln 2\pi - \frac{1}{2} |\hat{J}| \\ &= \ln p(y, \hat{\theta}_y) + \ln p(\hat{\theta}_y) + \frac{n}{2} ln 2\pi - \ln |NI| - \ln |\frac{1}{N} \hat{J}| \\ &= \ln p(y, \hat{\theta}_y) + \ln p(\hat{\theta}_y) + \frac{n}{2} ln 2\pi - n \ln N - O(1) \end{aligned} \tag{A.22}$$

The Bayesian information criterion (BIC) rule selects the order to minimize

$$BIC = -2 \ln p_n(y, \hat{\theta}^n) + n \ln N. \tag{A.23}$$

Appendix B

Arabic System in Detail

B.1 Building Arabic Master Dictionary

The Arabic master phonetic dictionary consists of about 1.2 million words, each with about 3.76 pronunciations on average. This is the master dictionary from which we derive phonetic pronunciations for all phonetic word-based and morpheme-based Arabic ASR systems.

The lexicon is selected based on the word frequency from the language model training data. High frequency words without pronunciations are discarded from the lexicon. Given the list of 490,000 top high frequency words that appear at least 30 times in our language model training data, there are 390,000 of them with pronunciations. That means there are 100,000 high frequency words that could not be analyzed by the tool and do not exist in the manually-vocalized corpora.

B.2 Data driven Morpheme system

For a candidate word to be decomposed, it has to satisfy the following five conditions:

1. It does not belong to the blacklist.

prefixes	Al bAl fAl kAl ll wAl b f k l s w
suffixes	An h hA hm hmA hn k km kn nA ny t th thA thm thmA thn tk tkm tm tnA tny tynA wA wh whA whm wk wkm wn wnA wny y yn

Table B.1: Data driven affix list for morpheme based system

2. It consists of at least one of the pre-determined affixes.

3. Its decomposed affixes' pronunciations match the pre-determined pronunciations.

4. Its stem must exist in the master dictionary.

5. Its stem must be at least two letters long.

B.3 Linguistically driven Morpheme system

The decomposable words are finally decomposed as follows:

1. If the word is in the blacklist, keep unchanged;
2. else if no prefix, decompose into stem and suffix;
3. else if no suffix, decompose into prefix and stem;
4. else if prefix+stem is in the blacklist, decompose into prefix+stem and suffix;
5. else if stem+suffix is in the blacklist, decompose into prefix and stem+suffix;
6. else decompose into prefix, stem, and suffix.

B.4 OOV Comparison

As shown in Table B.2, both word-based systems (P and G) have pretty high OOV rate for all the test sets even though they use rather large vocabulary. The graphemic system (G) has a bit lower OOV rate since it uses a larger vocabulary. In contrast to use words as recognition units, both morphemic systems (M1 and M2) have much lower OOV rate even though their vocabulary sizes are substantially smaller. It is

interesting to point out that the vocabulary of the M2 system has similar OOV rates across all test sets. Note that we scaled the morpheme OOV rate of the morpheme vocabulary using the decomposition rate (number of morphemes after decomposition divided by number of words before decomposition) to make it comparable to the standard word OOV rate.

Sys.	units	OOV-rate	Prons
P	390K	4.02	4.03
G	490K	3.43	1.00
M1	289K	1.31	4.43
M2	284K	0.66	3.69

Table B.2: OOV rates and number of pronunciations comparison

B.5 Detail Results of 12 Arabic Systems

ad9s	ae9s	ad10c	System	Lexical and Phonetic
14.4	9.8	13.7	RDT(PLP+MLP)	Linguistically-driven Morpheme
14.6	9.7	13.7	RDT(PLP+Pitch+MLP)	Linguistically-driven Morpheme
14.9	9.8	13.9	RDT(PLP+AdaBoost)	Linguistically-driven Morpheme
14.7	9.7	13.8	RDT(PLP+MLP)	Phonetic Data-driven Morpheme
15.0	10.3	14.5	RDT(MLP+PLP)	Graphemic Data-driven Morpheme
16.0	10.9	15.0	PLP	Graphemic Data-driven Morpheme
15.4	10.2	14.6	PLP	Phonetic Data-driven Morpheme
15.0	10.3	14.4	RDT(PLP+MLP)	Dialect-biased Graphemic Word-based
14.9	9.8	13.9	RDT(MLP+PLP)	Phonetic Word-based
14.6	10.3	14.5	RDT(PLP+MLP)	Graphemic Word-based
15.7	10.7	14.7	PLP	Graphemic Word-based
15.4	10.2	14.6	PLP	Phonetic Word-based
13.2	8.5	12.1	ROVER-11	(without AdaBoost system)
13.1	8.4	12.0	ROVER-12	(with AdaBoost system)

Table B.3: Individual and ROVER results for Arabic system

Bibliography

[1] T. Anastasakos, J. McDonough, and J. Makhoul. Speaker adaptive training a maximum likelihood approach to speaker normalization. *Proceedings of International Conference on Acoustics, Speech and Signal Processing*, April 1997.

[2] L.R. Bahl, P.F. Brown, P.V. de Souza, and R.L. Mercer. Maximum mutual information estimation of hidden markov model parameters for speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1986.

[3] Christopher M. Bishop. Pattern recognition and machine learning. *Information Science and Statistics*, 2007.

[4] H. Bourlard and N. Morgan. Connectionist speech recognition: A hybrid approach. *Kluwer Academic Publishers, Norwell, MA, USA*, 1993.

[5] L. Breiman. Bagging predictors. *Machine Learning*, 1996.

[6] Stanley F. Chen and Joshua Goodman. An empirical study of smoothing techniques for language modeling. *Technical Report 10-98, Harvard University*, 1998.

[7] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Accoustics, Speech and Signal Processing*, 1980.

[8] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 1977.

[9] L. Deng and J J. Chen. Sequence classification using the high-level features extracted from deep neural networks. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

[10] L. Deng and D. Yu. Deep convex network: A scalable architecture for speech pattern classification. *Proceedings of Interspeech*, 2011.

[11] Chris Ding and Xiaofeng He. Cluster merging and splitting in hierarchical clustering algorithms. *Data Mining*, 2002.

[12] Richard O. Duda, Peter E. Hart, and David G. Stork. Pattern classification. *Second Edition*, 2000.

[13] G. Eibl and R. Schapire. Multiclass boosting for weak classifiers. *In Journal of Machine Learning Research*, 2005.

[14] D.P.W. Ellis, R. Singh, and S. Sivadas. Tandem acoustic modeling in large-vocabulary recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2001.

[15] Spyros Matsoukas et al. Advances in transcription of broadcast news and conversational telephone speech within the combined ears bbn/limsi system. *Audio, Speech, and Language Processing, IEEE Transactions*, 2006.

[16] G. Evermann and P.C. Woodland. Large vocabulary decoding and confidence estimation using word posterior probabilities. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2000.

[17] J. Fiscus. A post-processing system to yield reduced word error rates: recognizer output voting error reduction (rover). *Proceedings of ASRU*, 1997.

[18] P. Fousek, L. Lamel, and J. L. Gauvain. Transcribing broadcast data using mlp features. *Proceedings of InterSpeech*, 2008.

[19] Yoav Freund. Boosting a weak learning algorithm by majority. *Information and Computation 121*, 1995.

[20] Yoav Freund and Robert Schapire. Experiments with a new boosting algorithm. *Machine Learning: Proceedings of the Thirteenth International Conference*, 1996.

[21] Yoav Freund and Robert Schapire. Game theory, on-line prediction and boosting. *Proceedings of the Ninth Annual Conference on Computational Learning Theory*, 1996.

[22] Yoav Freund and Robert Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 1997.

[23] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *Annals of Statistics 28*, 2000.

[24] Jerome H. Friedman. Stochastic gradient boosting. *Technical report*, 1999.

[25] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics 29*, 2001.

[26] M. J. F. Gales. Maximum likelihood linear regression speaker adaptation of continuous densisty hmms. *Computer speech and Language, vol 12*, 1997.

[27] M. J. F. Gales. Semi-tied covariance matrices for hidden markov models. *Transactions on Speech and Audio Processing*, May 1999.

[28] I.J. Good. The population frequencies of species and the estimation of population parameters. *Biometrika, 40(3 and 4)*, 1953.

[29] R. Haeb-Umbach and H. Ney. Linear discriminant analysis for improved large vocabulary continuous speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1992.

[30] Hynek Hermansky. Perceptual linear predictive (plp) analysis of speech. *Journal of the Acoustical Society of America*, 1989.

[31] Hynek Hermansky. Temporal patterns (traps) in asr of noisy speech. *Acoustics, Speech, and Signal Processing*, 1999.

[32] G. E. Hinton, S. Osindero, and Y. Teh. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[33] Moonyoung Kang, Tim Ng, and Long Nguyen. Mandarin word-character hybrid-input neural network language model. *Proceedings of INTERSPEECH*, 2011.

[34] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 1987.

[35] Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech, and Signal Processing, VOL. ASP-35*, 1987.

[36] B. Kingsbury. Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009.

[37] Reinhard Kneser and Hermann Ney. Improved backing-off for m-gram language modeling. *In Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1995.

[38] Andrew Lovitt, Joel Pinto, and Hynek Hermansky. On confusions in a phoneme recognizer. *IDIAP Research Report*, 2007.

[39] Jean luc Gauvain and Chin hui Lee. Maximum a posteriori estimation for multivariate gaussian mixture observations of markov chains. *IEEE Transactions on Speech and Audio Processing*, 1994.

[40] John Makhoul. Linear prediction: A tutorial review. *Proceedings of IEEE, 63:561-580*, 1975.

[41] L. Mangu. Finding consensus in speech recognition. *PhD Thesis*, 2000.

[42] L. Mangu, E. Brill, , and A. Stolcke. Finding consensus among words: Lattice-based word error minimization. *Proceedings of Eurospeech*, 1999.

[43] L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. *NIPS*, 2000.

[44] Spyros Matsoukas and Richard Schwartz. Improved speaker adaptation using speaker dependent feature projections. *Proceedings of ASRU*, November, 2003.

[45] Bernd T. Meyer, Matthias Wachter, Thomas Br, and Birger Kollmeier. Phoneme confusions in human and automatic speech recognition. *Proceedings of Interspeech*, 2007.

[46] T. M. Mitchell. Machine learning. *McGraw-Hill*, 1997.

[47] A. Mohamed, G. Dahl, and G. Hinton. Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing*, 2012.

[48] A. Mohamed, T. N. Sainath, G E. Dahl, B. Ramabhadran, G. E. Hinton, and M. Picheny. Deep belief networks using discriminative features for phone recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[49] A. Mohamed, D. Yu, and L. Deng. Investigation of full-sequence training of deep belief networks for speech recognition. *Proceedings of Interspeech*, 2010.

[50] Tim Ng, Kham Nguyen, Rabih Zbib, and Long Nguyen. Improved morphological decomposition for arabic broadcast news transcription. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2009.

[51] Tim Ng, Bing Zhang, and Long Nguyen. Jointly optimized discriminative features for speech recognition. *Proceedings of InterSpeech*, 2010.

[52] Kham Nguyen and John Makhoul. Arabic system combination. *CDSP workshop*, 2009.

[53] Kham Nguyen, Tim Ng, and Long Nguyen. Adaptive boosting features for automatic speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2012.

[54] Long Nguyen, Tasos Anastasakosy, Francis Kubala, Christopher LaPre, John Makhoul, Richard Schwartz, Nina Yuan, George Zavaliagkosy, and Ying Zhao. The 1994 bbn/byblos speech recognition system. *Proceedings of ARPA Spoken Language Systems and Technology Workshop*, 1994.

[55] Long Nguyen, Spyros Matsoukas, Jason Davenport, Frances Kubala, Richard Schwartz, and John Makhoul. Progress in transcription of broadcast news using byblos. *Speech Communication 38, pp. 213-230*, 2002.

[56] Long Nguyen, Tim Ng, Kham Nguyen, Rabih Zbib, and John Makhoul. Lexical and phonetic modeling for arabic automatic speech recognition. *InterSpeech*, 2009.

[57] Long Nguyen and Rich Schwartz. Single-tree method for grammar-directed search. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 1999.

[58] Long Nguyen and Bing Xiang. Light supervision in acoustic model training. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2004.

[59] Joseph Olive, Caitlin Christianson, and John McCary. Handbook of natural language processing and machine translation: Darpa global autonomous language exploitation. *Edition 1*, 2011.

[60] J. Park, F. Diehl, M. J. F. Gales, M. Tomalin, , and P. C. Woodland. Training and adapting mlp features for arabic speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, April 2009.

[61] D. Povey. Improvements to fmpe for discriminative training of features. *Proceedings of Interspeech*, 2005.

[62] D. Povey, D. Kanevsky, B. Kingsbury, B. Ramabhadran, G. Saon, and K. Visweswariah. Boosted mmi for model and feature-space discriminative train. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2008.

[63] D. Povey, B. Kingsbury, L. Mangu, G. Saon, H. Soltau, and G. Zweig. fmpe: Discriminatively trained features for speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2005.

[64] D. Povey and P.C. Woodland. Minimum phone error and ismoothing for improved discriminative training. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2002.

[65] L. R. Rabiner. A tutorial on hidden markov models and selected applications. *speech recognition. Proceedings of the IEEE*, 1989.

[66] L. R. Rabiner and B. H. Juang. An introduction to hidden markov models. *IEEE ASSP Magazine*, 1986.

[67] Lawrence Rabiner and Biing-Hwang Juang. Fundamentals of speech recognition. *Book*, 1993.

[68] G. Ratsch and M. Warmuth. Maximizing the margin with boosting. *Proceedings of the 15th Annual Conference on Computational Learning Theory*, 2002.

[69] Raul Rojas. Adaboost and the super bowl of classifiers a tutorial introduction to adaptive boosting. *Computer Science Department Freie Universit at Berlin*, 2009.

[70] L. Rosasco, E. De, Vito A. Caponnetto, M. Piana, and A. Verri. Are loss functions all the same? *Neural Computation*, 2003.

[71] S. Rosset, J. Zhu, and T. Hastie. Boosting as a regularized path to a maximum margin classifier. *Journal of Machine Learning Research*, 2004.

[72] S. Rosset, J. Zhu, and T. Hastie. Margin maximizing loss functions. *Advances in Neural Information Processing Systems*, 2004.

[73] Mohammad J. Saberian and Nuno Vasconcelos. Multiclass boosting: Theory and algorithms. *Internal Report*, 2011.

[74] T. N. Sainath, B. Ramabhadran, and M. Picheny. An exploration of large vocabulary tools for small vocabulary phonetic recognition. *IEEE Automatic Speech Recognition and Understanding Workshop*, 2009.

[75] R. Schapire. The boosting approach to machine learning: an overview. *MSRI workshop on Nonlinear Estimation and Classification*, 2002.

[76] R. Schapire, Y. Freund, P. Bartlett, and W. Lee. Boosting the margin: a new explanation for the effectiveness of voting methods. *Annals of Statistics 26*, 1998.

[77] R. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning 37*, 1999.

[78] Gideon E. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, 1978.

[79] Fei Sha and Lawrence K. Saul. Large margin gaussian mixture modeling for phonetic classification and recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[80] M. Siu, H. Gish, and F. Richardson. Improved estimation, evaluation and application of confidence measures for speech recognition. *Proceedings of Eurospeech*, 1997.

[81] Petre Stoica and Y. Selen. Model-order selection: a review of information criterion rules. *IEEE Signal Processing*, 2004.

[82] A. Stolcke, Y. Konig, and M. Weintraub. Explicit word error minimization in n-best list rescoring. *Proceedings of Eurospeech*, 1997.

[83] Karel Vesely, Arnab Ghoshal, Lukas Burget, and Daniel Povey. Sequence-discriminative training of deep neural networks. *Proc. Interspeech*, 2013.

[84] P.C. Woodland and D. Povey. Large scale mmie training for conversational telephone speech recognition. *em Proceedings of Speech Transcription Workshop*, 2000.

[85] Bing Xiang, Kham Nguyen, Long Nguyen, Rich Schwartz, and John Makhoul. Morphological decomposition for arabic broadcast news transcription. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[86] Haihua Xua, Daniel Povey, Lidia Manguc, and Jie Zhua. Minimum bayes risk decoding and system combination based on a recursion for edit distance. *Computer Speech and Language*, 2011.

[87] Pei Yin, Irfan Essa, Thad Starner, and James M. Rehg. Discriminative feature selection for hidden markov models using segmental boosting. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, May, 2008.

[88] S. Young. Large vocabulary continuous speech recognition: A review. *IEEE Signal Processing Magazine*, 1996.

[89] B. Zhang, S. Matsoukas, and R. Schwartz. Discriminatively trained region dependent feature transforms for speech recognition. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[90] B. Zhang, S. Matsoukas, and R. Schwartz. Recent progress on the discriminative region-dependent transform for speech feature extraction. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, 2006.

[91] Jing Zheng and Andreas Stolcke. Improved discriminative training using phone lattices. *Proceedings of InterSpeech*, 2005.

[92] J. Zhu, H. Zou, S. Rosset, and T. Hastie. Multiclass adaboost. *unpublished*, 2005.

[93] G. Zweig and M. Padmanabhan. Boosting gaussian mixtures in an lvcsr system. *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, June, 2000.