ERROR PREDICTION FOR SPEECH RECOGNITION USING ACOUSTIC AND LINGUISTIC CUES

By

Nikos E. Malandrakis

SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
DIPLOMA IN ELECTRONICS AND COMPUTER ENGINEERING
AT
TECHNICAL UNIVERSITY OF CRETE
CHANIA, GREECE
SEPTEMBER 2007

TECHNICAL UNIVERSITY OF CRETE
DEPARTMENT OF
ELECTRONICS AND COMPUTER ENGINEERING

The undersigned hereby certify that they have read and recommend to the Faculty of Undergraduate Studies for acceptance a thesis entitled "**Error prediction for speech recognition using acoustic and linguistic cues**" by **Nikos E. Malandrakis** in partial fulfillment of the requirements for the degree of **Diploma in Electronics and Computer Engineering**.

Dated: September 2007

Supervisor: _____
Assoc. Prof. Alexandros Potamianos

Readers: _____
Prof. Vasilis Digalakis

_____
Assoc. Prof. Athanasios Liavas

TECHNICAL UNIVERSITY OF CRETE

Date: **September 2007**

Author: **Nikos E. Malandrakis**

Title: **Error prediction for speech recognition using acoustic and linguistic cues**

Department: **Electronics and Computer Engineering**

Degree: **Diploma** Convocation: **September** Year: **2007**

Signature of Author

To my brother

Table of Contents

List of Figures

Abstract

In this work we explore a method of approximating a speech recognition system via a composition of finite state machines and use this approximation to predict word errors of a standard ASR system. Specifically, N-best recognition is used as an indication of word confusability which can help us define confidence scores or just classify the output of a speech recognizer as correct or wrong. We explore the factors affecting the accuracy of such a classifier, such as the classes of the individual models used as well as the practical limitations of the implementation and possible solutions.

Our motivations comes from work showing that the perplexity of such a model correlates well with the WER (word error rate) [9] of a speech recognition system. Taking that as a given, such a system should be capable of predicting the errors of an ASR system, the probability assigned to an assumption should correlate with the probability of that assumption being correct.

To test that theory we use a transducer composite system to assign probabilities to recognition assumptions knowing the correct word and to correct word assumptions given the recognized word. Then using these probabilities as features, we classify utterances as correct or wrong. Evaluating the performance of the method happens using ROC diagrams, showing the performance potential of the classifier in different roles.

Overall it is shown that such an approach is valid, with results being encouraging if not particularly good. Our implementation of some parts of the system, notably the acoustic model prove inadequate and should be the focus of any future work.

Acknowledgments

I want to express my appreciation to my supervisor Mr. Alexandros Potamianos for his patience and guidance.

I am also grateful to the members of the supervisory committee.

I would like also to thank my friends and family for their love and support.

Introduction

Predicting speech recognition errors is related to the concept of word confusability; words with similar phonetic sequences can be confused with each other, unless the language model asserts its influence to disambiguate them.

Research is being conducted towards the goal of quantifying word confusability, defining a metric for it. In parallel, research is targeted at methods of decreasing the confusability inherent in each speech recognizer via for example removing from the lexicon particular pronunciations.

Attempting to predict the errors of a speech recognizer seems like a logical next step, since the more confusable word is one that has a higher chance of a misrecognition. There are limits to what can be done; word confusability is defined over all pronunciations of each word, however in an actual speech recognition experiment only one of those is going to be used per sample and the exact pronunciation used affects the recognition outcome. Despite that it is still an experiment worth doing.

There are three parts in the approach, each with it's issues that need to be addressed. First of all there's feature selection, since word confusability is still work in progress with no definite calculation method. So some feature corelating with word error rate must be selected, an example being the probability of error over a N-best recognition.

Secondly a method of extracting said feature must be devised. Referencing related work, approximating the speech recognition process with a composition of finite state machines is a popular approach and one that creates an easily manipulated system. Using the probabilities/costs returned by such a system can serve as feature extraction.

The last part is classifying the samples based on the features via selecting the appropriate threshold(s). The unclear part of this is the relative weight of error types. A modern speech recognizer has a very low word error rate, so even a small percentage of samples being mis-predicted as mistakes can lead to the predictor "rejecting" more correct samples (numerically) than wrong samples. Therefore such an approach only makes sense assuming that the classifier has extremely good performance or that the cost of a wrong word is much higher than that of a false alarm.

This being preliminary work the feature extraction is based upon a system composed of simple models, the main question being if the approach even works. As such no single feature is picked, instead performance is compared across multiple alternatives. The threshold choice problem is left untouched, instead all meaningful thresholds are used in every test and ROC diagrams are produced by the multiple results.

Beyond that comes the question of why it works and what is the contribution of each component to the outcome. The contribution of each component serves as an evaluation of the components, indicating which would require a more complex (higher order) alternative. Beyond that, the performance difference between components of different orders is an answer to the question "do the results justify the use of more complex models?", which is important since the main weakness of the method is extreme computational complexity.

Outline

The thesis is organized as follows: Chapter 1 provides the necessary theoretical background. In Chapter 2 the principles behind our implementation are discussed and in Chapter 3 the experimental procedure is thoroughly described. In Chapter 4 an evaluation of the proposed system is provided and finally, in Chapter 5 some conclusions are provided along with an outline of future work and ideas that may improve the performance of the proposed system.

Chapter 1

Theory

1.1 Speech Recognition System [15]

A speech recognition system is the implementation of the equation

$$\hat{W} = arg \max_{W} P(W|O) \tag{1.1.1}$$

Where W represents linguistic assumptions and O represents the acoustic observations. As such , it provides the most likely linguistic assumption given the acoustic observation vectors.

Most speech recognition systems are based around Hidden Markov Models, because of their favorable characteristics. Creating an Automatic Speech Recognition (ASR) system requires a front end, which extracts features from the speech signal and acoustic and linguistic models, defining the statistic properties of phone and word sequences. The decoding process checks all expressions and decides on the most likely one as the speech recognition output. A schematic representation of such a system can be seen in figure (1.1)

The speech signal is viewed as locally stationary, meaning it's spectral characteristics stay constant for very small intervals. An ASR system's front end works based on that assumption, splitting the input signal in frames typically a few msec each and extracting a set of features from each frame.

Acoustic models are used to provide the probability of each acoustic observation vector sequence given the word W. This could be implemented by getting multiple examples of each word and determining the statistics of the vector

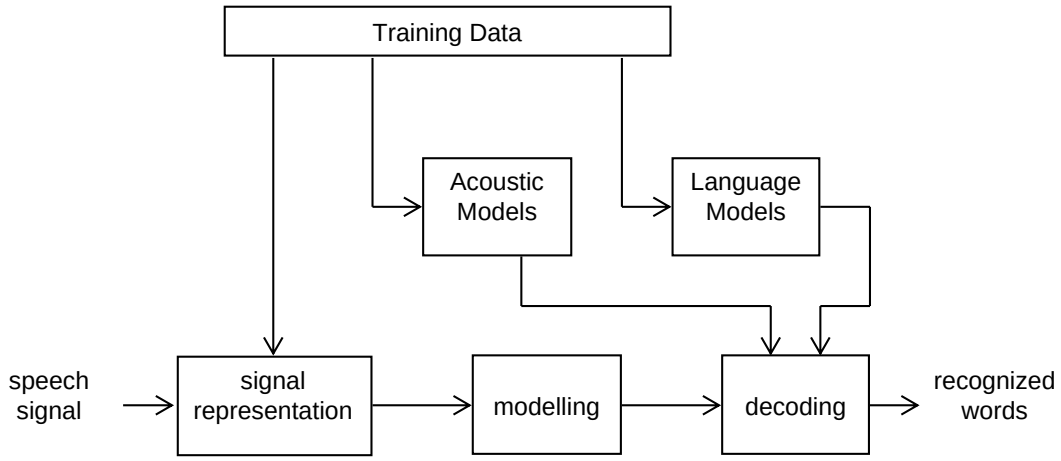

Figure 1.1: ASR System

sequences, however it is impractical (or even impossible) for large vocabulary systems. So we prefer decomposing words to basic sounds called phones. A pronunciation lexicon is created mapping each word to all corresponding phone sequences.

There is variation in the pronunciation of each phoneme depending on the speaker, conditions and acoustic context, requiring multiple HMM models for each phoneme. A typical approach involves the use of triphones, triads of phonemes focused on the middle (second) phoneme. Each phoneme has one HMM trained for each combination of left and right neighbors, resulting in a very large total number of HMMs. These models are trained in an iterative fashion.

Decoding takes all alternate paths of the recognition process and picks the best path by applying the Viterbi method.

1.2 Language Modeling [4]

1.2.1 Counting Words

Probability theory deals with predicting how likely it is that an event will take place. There are two interesting views of probabilities:

The objectivist view states that probabilities are real aspects of the world that can be measured by relative frequencies of outcomes of experiments. In contrast, according to the subjectivist view, probabilities are descriptions of an observer's degree of belief or uncertainty rather than having any external significance. These contrasting views are also referred to as Frequentist vs. Bayesian. Both views are relevant for linguistics; yet, the laws of probability theory remain the same under both interpretations.

Probabilities are based on counting things. In our case, statistical language processing requires computation of word probabilities. These probabilities are computed by counting words or lexical units in a corpus. The classical definition of probability, as given by Pascal is:
"*The probability of an event x is computed as the relative frequency with which x occurs in a sequence of n identical experiments.*"
So, the probability of a word w can be approximated by:

$$p(w) = \frac{occurrences \quad of \quad word \quad w}{number \quad of \quad words} \tag{1.2.1}$$

which is the relative frequency with which w occurs in the corpus.

1.2.2 Punctuation Marks

Suppose that we have to count the words of the following sentence from Shakespeare's Hamlet and compute the probability of the word "God".
"Oh God, I could be bounded in a nutshell and count myself a king of infinite space."
If we count punctuation marks as words, the sentence has totally 19 words, if we do not 17 words..

Whether we count punctuation marks as words depends on the task. Tasks such as grammar checking or author-identification must treat punctuation marks

as words because in these cases the location of the punctuation is important. Usually corpora of spoken language do not have punctuation marks.
In our work punctuation marks are not taken under consideration.

1.3 N-grams Probabilistic Model

This model proposes the assignment of probabilities to strings of words. Based on this method we can easily compute the probability of an entire sentence or predict the next word in a sequence.

The simplest probabilistic version of this model allows every word to have the same probability of following every other word. A more robust model let every word follow every other word, with the appearance of the following word to be depended on its normal frequency of occurrence. We still consider the individual relative frequency of each word. The following example, based on real data, can verify the precision of this simplistic approach.

Brown corpus [2] has $1,000,000$ words. The word "the" occurs 69,971 times in the corpus and the word "rabbit" occurs 11 times. Thus the probabilities are 0.07 and 0.00001 for the words "the" and "rabbit", respectively. Suppose that we have just read this part of a sentence: "Just then, the white...". Furthermore, suppose that we are curious about what the next word will be. If we use the simple model, we will conclude that the word "the" is the most possible word to follow "white". But this seems totally false because there is no meaning in the sentence "Just then, the white the". Doubtless the sentence "Just then, the white rabbit" sounds more reasonable. This example shows that the computation of the probability of word sequences must use the conditional probability of a word given the previous word. Particularly that means that the probability of a word given the previous one is higher than its probability otherwise.

1.3.1 Conditional probability and independence

The notion of conditional probability can be considered as a kind of updated probability of an event given some knowledge. The probability of an event before gaining additional knowledge is referred to as the prior probability of the event.

The new probability of the event estimated using the additional knowledge is called posterior probability of the event. The event of interest is formed by the occurrences of a word in the corpus. Using symbol Ω we denote the sample space, which is discrete, having finite number of elements. The sample space, which corresponds to a corpus, includes all the occurrences of each distinct word of the corpus. That is, the sample space Ω includes all the events of the corpus. For instance, assume a small corpus: "A tiny corpus tiny". For the previous corpus: Ω = { occurrence of the word "A", occurrence of the word "tiny", occurrence of the word "corpus", occurrence of the word "tiny" for second time }.

The event of the occurrence of "tiny" in the corpus is denoted as tiny and is: tiny = {occurrence of the word "tiny", occurrence of the word "tiny" for second time}.

We define the probability of the occurrence of the word "tiny" according to Eq. (1.2.1) as:

$$p(tiny) = \frac{|tiny|}{|\Omega|} \tag{1.3.1}$$

where $|tiny|$ is the number of elements in the set "tiny" and $|\Omega|$ is the number of elements in the probability space Ω. Thus $p(\Omega) = 1$. So, the probability of the event $p(tiny)$ is: $p(tiny) = \frac{|tiny|}{|\Omega|} = \frac{2}{4}$.

For the general case:

$$p(event) = \frac{|event|}{|\Omega|} \tag{1.3.2}$$

where each event is a subset of Ω.

The conditional probability of a word w_2 assuming that word w_1 has occurred ($p(w_1) > 0$), denoted $p(w_2|w_1)$ equals:

$$p(w_2|w_1) = \frac{p(w_2 \cap w_1)}{p(w_1)} \tag{1.3.3}$$

which can easily be transformed into:

$$p(w_2|w_1)p(w_1) = p(w_2 \cap w_1) \tag{1.3.4}$$

Eq. (1.3.3) gives:

$$p(w_1|w_2) = \frac{p(w_1 \cap w_2)}{p(w_2)} \tag{1.3.5}$$

We can do the conditionalization either way because set intersection is symmetric, $w_2 \cap w_1 = w_1 \cap w_2$. So Eq. (1.3.3) becomes:

$$p(w_2|w_1) = \frac{p(w_2)p(w_1|w_2)}{p(w_1)} \tag{1.3.6}$$

Two words w_1, w_2 are considered to be conditionally independent if $p(w_2 \cap w_1) = p(w_2)p(w_1)$.

Conditional probability and independence can be the basis for computing the probability of a string of words. A string of words can be represented as $w_1, w_2, ..., w_{n-1}, w_n$ or $w_{1..n}$. Assuming the occurrence of each word in the corpus as an independent occurrence, we can write the probability of a string of words as follows:$p(w_1, w_2, ..., w_{n-1}, w_n)$ or $p(w_{1..n})$.

Using the chain rule of probability we represent $p(w_{1..n})$ as:

$$p(w_{1..n}) = p(w_1)p(w_1|w_2)p(w_3|w_{1..2})...p(w_n|w_{1..n-1}) = \prod_{k=1}^{n} p(w_k|w_{1..k-1}) \tag{1.3.7}$$

Since there is not any easy way for computing the probability of a word given all the previous words, an alternative solution for this task is to find a satisfactory approximation. The bigram model proposed for solving this difficulty, assumes that the probability of a word depends only on the previous word. In other words, $p(w_n|w_{1..n-1})$ is approximated by the conditional probability of the word that preceded $p(w_n|w_{n-1})$. This approximation is referred as a Markov assumption. Markov models are probabilistic models, which predict a future event without much prior knowledge. In the case of bigram (first order Markov model) models they need to know only the preceding word.

It is obvious that the trigram (second order Markov model) model looks two words into the past. Generalizing bigrams and trigrams, N-grams are resulted by which the probability of a word given all the previous words can be approximated by the probability given only the previous N words.

$$p(w_n|w_{1..n-1}) \simeq p(w_n|w_{(n-N+1)..(n-1)}) \tag{1.3.8}$$

For a bigram grammar, $p(w_{1..n})$ can be found by combining Eq. (1.3.8) and

Eq. (1.3.7):

$$p(w_{1..n}) \simeq \prod_{k=1}^{n} p(w_k|w_{k-1}) \tag{1.3.9}$$

1.3.2 Smoothing

For any particular corpus, it is possible that some N-grams not to exist in this corpus. The consequence is that the N-gram model assigns zero probability to these N-grams. Also, using only relative frequencies to estimate N-grams probabilities might produce poor estimates when the counts are too small. This major problem raises the need to find a way of reevaluating zero probability and low probability N-grams and assigning them non-zero values. This procedure is called smoothing.

Add-One Smoothing

This algorithm suggests to take the bigram counts and before normalizing them to probabilities, to add one to all the counts. This algorithm is very simple and in practice does not perform well. However it stands as an introduction to the concept of smoothing that is implemented much better by other algorithms.

Considering the unsmoothed maximum likelihood estimate of the unigram probability:

$$p(w_x) = \frac{c(w_x)}{\sum_i c(w_i)} = \frac{c(w_x)}{N} \tag{1.3.10}$$

where $c(w_x)$ is the frequency (counts) of word w_x in the corpus and N represents the total number of word tokens in the corpus.

The basic idea of smoothing relies on the c's adjustment. The adjusted count for add-one smoothing is defined by adding one to the count c and then multiplying by the factor N/(N+V), which is a normalization factor. Then the adjusted count is:

$$c_i^* = \frac{(c_i + 1)N}{(N + V)} \tag{1.3.11}$$

where V is the vocabulary size of the corpus.

Eq. (1.3.11) can be turned into probabilities p_i^* by dividing with the total number of word tokens:

$$p_i^* = \frac{(c_i + 1)}{(N + V)} \tag{1.3.12}$$

Applying Eq. (1.3.12) to Eq. (1.3.10), the add-one-smoothed probability for a bigram is defined as:

$$p^*(w_n|w_{n-1}) = \frac{c(w_{n-1}w_n) + 1}{c(w_{n-1}) + V} \tag{1.3.13}$$

An alternative view of smoothing

Actually a smoothing algorithm discounts some non-zero counts. This is a way to find the probability mass, which will be assigned to the zero counts. An alternative way to refer to lowered counts c^* is to define a discount ratio d_c:

$$d_c = \frac{c^*}{c} \tag{1.3.14}$$

The choice of value one (1) which is added to the each count c is arbitrary. This affects the probability mass that is moved near the zero value. A solution to this problem is the choice of smaller values regarding the situation.

1.3.3 Witten-Bell Discounting

This algorithm is referred as Method C, a method initially introduced by Alistair Moffat [8] and is considered to perform better than Add-One smoothing. In [16], Witten and Bell surveyed and compared several approaches to the zero-frequency problem that have been used in text compression systems. Witten and Bell described the zero-frequency problem in the case of adaptive word coding assuming a coding scheme in which the encoder reads the next word of text, searches for it in a list and transmits an index extracted from the list in place of the word. If the next word is not appeared in the list, a special code, called escape code, must be transmitted followed by the unknown word. This new word is added to the encoder and decoder's lists in case it appears again. According to this method each word is assigned an associated frequency. The computing of

probability of the escape character, by estimating the likelihood of a novel word occurring, can solve the zero-frequency problem.

Similarly, a novel N-gram could then be assigned the probability of seeing it for the first time. The basic idea behind this conception is to "use the count of things we have seen once to help estimate the count of things we have never seen".

We can compute the probability of seeing a novel N-gram by counting the number of times we saw N-grams for the first time in the corpus. The count of the first-time seen N-grams is simply the number of N-gram types we have already seen.

Hence we can estimate the total probability mass of all the zero N-grams by dividing the number of N -gram types we have seen with the sum of number of tokens and the number of N -gram types we have seen:

$$\sum_{i:c_i=0} p_i^* = \frac{T}{N+T} \tag{1.3.15}$$

where T is the N-gram types we have already seen and N is the number of tokens.

Probability given by Eq. (1.3.15) is the total probability of unseen N-grams. This "amount of probability" needs to be divided in order to assign a part of it to each zero N-gram. A simple compromise is to divide equally. Letter Z denotes the total number of N-grams with count zero. So the equal share of the probability mass is:

$$p_i^* = \frac{T}{Z(N+T)} \tag{1.3.16}$$

The probability of all the seen N-grams is given by the equation:

$$p_i^* = \frac{c_i}{N+T'}, \quad if \quad c_i > 0 \tag{1.3.17}$$

Extending the Witten-Bell discounting to bigrams, the type-counts are conditioned on some history. The probability of seeing for first time a bigram w_{n-1} w_n is equivalent to the probability of seeing a new bigram starting with the word w_{n-1}.

According to the Eq. (1.3.14) the probability of a bigram w_x w_i we have not

seen is:

$$\sum_{i:c(w_x w_i)=0} p^*(w_i|w_x) = \frac{T(w_x)}{N(w_x)+T(w_x)} \tag{1.3.18}$$

where $T(w_x)$ is the number of bigram types on the previous word w_x we have already seen and $N(w_x)$ is the number of bigram tokens on the previous word w_x.

Distributing the probability mass of the Eq. (1.3.18) among all the unseen bigrams, we get:

$$p^*(w_i|w_{i-1}) = \frac{T(w_{i-1})}{Z(w_{i-1})(N+T(w_{i-1}))}, \quad \text{if} \quad c(w_{i-1}w_i) = 0 \tag{1.3.19}$$

where $Z(w_{i-1})$ is the total number of bigrams with w_{i-1} as the first word, that have zero counts.

For the non-zero bigrams, we parameterize T on the history:

$$p^*(w_i|w_x) = \frac{c(w_x w_i)}{c(w_x)+T(w_x)}, \quad \text{if} \quad c(w_{i-1}w_i) > 0 \tag{1.3.20}$$

1.3.4 Backoff

So far the algorithms we have presented have all made use of the frequency of an N-gram and have tried to compute the best estimate of its probability. In general N-grams that never appeared or appeared only few times, were given the same estimate. A reasonable extension of the previous methods (smoothing) is to try to build better estimates by looking at the frequency of the (N-1)-grams found in the N-gram.

If (N-1)-grams, found in the N-gram, are appeared rarely, then a low estimate is given to the N-gram. Otherwise, N-grams with (N-1)-grams of moderate frequency are given a higher probability estimate. This issue grounded in a more general discussion deals with combining multiple probability estimates making use of different models. That is, if there are no examples of a particular trigram, let's say $w_{n-2}w_{n-1}w_n$, the computation of $p(w_n|w_{n-1}w_{n-2})$ can be achieved through the use of the bigram probability $p(w_n|w_{n-1})$. In the same manner, if we have no examples of $w_{n-1}w_n$ in order to compute $p(w_n|w_{n-1})$, we can use the unigram probability $p(w_n)$.

In the backoff model, as described above, an N-gram model is built based on a (N-1)-gram model. We only look to a lower-order N-gram if we have no examples of a higher-order N-gram. So the backoff model for the trigram $w_{i-2}w_{i-1}w_i$ can be calculated from:
Case 1:

$$p(w_i|w_{i-2}w_{i-1}) = p(w_i|w_{i-2}w_{i-1}) \quad \text{if} \quad c(w_{i-2}w_{i-1}w_i) > 0 \tag{1.3.21}$$

Case 2:

$$p(w_i|w_{i-2}w_{i-1}) = a_1 p(w_i|w_{i-1}) \quad \text{if} \quad c(w_{i-2}w_{i-1}w_i) = 0 \quad \text{and} \quad c(w_{i-1}w_i) > 0 \tag{1.3.22}$$

Case 3:

$$p(w_i|w_{i-2}w_{i-1}) = a_2 p(w_i) \quad \text{otherwise} \tag{1.3.23}$$

Parameters a_1 and a_2 are weighting factors, which ensure that the result of the previous equation system is a true probability. For the general case the form of backoff is:

$$\hat{p}(w_n|w_{(n-N+1)..(n-1)}) =$$

$$\tilde{p}(w_n|w_{(n-N+1)..(n-1)}) + \theta(p(w_n|w_{(n-N+1)..(n-1)}))a\hat{p}(w_n|w_{(n-N+2)..(n-1)}) \tag{1.3.24}$$

The θ notation indicates a binary function that selects a lower-order model only if the higher-order model produces a zero probability. Specifically, if $x = 0$ then $\theta(x) = 1$, else $\theta(x) = 0$. Each $p(.)$ is a Maximum Likelihood Estimation.

1.3.5 A clever combination: Backoff and Discounting

As was previously shown, discounting methods are used to calculate the probability mass which is assigned to unseen events, assuming that they were equally probable. Combining discounting with backoff, this probability can be distributed more cleverly.

Consider the following example, which shows how backoff can lead to probability greater than 1: Using relative frequencies, $\sum_{i,j} p(w_n|w_iw_j) = 1$, which means that the probability of a word w_n over all N-gram contexts equals to 1. If

we use backoff in this case, adopting a lower order model, the probability of w_n will be greater than 1. So, discounting must be applied to backoff model.

Thus, the correct form of Eq. (1.3.23) is:

$$\hat{p}(w_n|w_{(n-N+1)..(n-1)}) =$$

$$\tilde{p}(w_n|w_{(n-N+1)..(n-1)})+\theta(p(w_n|w_{(n-N+1)..(n-1)}))a(w_{(n-N+1)..(n-1)})\hat{p}(w_n|w_{(n-N+2)..(n-1)}) \tag{1.3.25}$$

where $\tilde{p}(.)$ stands for the discounted MLE probabilities:

$$\tilde{p}(w_n|w_{(n-N+1)..(n-1)}) = \frac{c^*(w_{(n-N+1)..n})}{c(w_{1..(n-N+1)})} \tag{1.3.26}$$

Function a represents the amount of probability mass, which must be distributed from an N-gram to an (N-1)-gram:

$$a(w_n|w_{(n-N+1)..(n-1)}) = \frac{1-\sum_\beta \tilde{p}(w_n|w_{(n-N+1)..(n-1)})}{1-\sum_\beta \tilde{p}(w_n|w_{(n-N+2)..(n-1)})} \tag{1.3.27}$$

where β denotes $w_n : c(w_{(n-N+1)..(n-1)}) > 0$.

1.4 String Distance [10] [4]

String Distance is a measure of the similarity between two strings x and y, useful during string searching/matching. The metrics used for string distance can be also used for word distance and phone sequence distance. For example the problem of finding the distance between "such nice weather" and "a nice weather", the distance between "horse" and "horses" and the distance between "ah n iy" and "ah n ih" are functionally equivalent, with the distance being measured in different units since each sequence is composed of different building blocks (words, letters, phones respectively). The basis for string distance metrics is the list of fundamental editing operations ;

1. ***substitutions***: A character in x is replaced by the corresponding character in y.

2. ***insertions***: A character in y is inserted into x, thereby increasing the length of x by one character.

3. ***deletions***: A character in x is deleted, thereby decreasing the length of x by one character.

1.4.1 Hamming Distance

One distance metric is the Hamming distance, introduced by Richard Hamming in 1950. It is defined as the number of substitutions required to change one string into the other. For example the Hamming distance of strings "such nice weather" and "a nice weather" is 1 word, since by substituting "such" with "a" we transform the first string into the second one. The main problem with the Hamming distance is that it only applies to strings of equal length since there is no support for insertions and deletions.

1.4.2 Minimum Edit Distance

Minimum edit distance, also knows by as Levenshtein distance is a generalization of Hamming distance to include all fundamental operations (substitutions, insertions and deletions). It is defined as the minimum number of fundamental operations required to convert one string into the other. For example the minimum edit distance between "horse" and "horses" is 1 letter insertion, since by inserting an "s" at the end of "horse" we get "horses".

The minimum edit distance path can be represented in many ways, like this alignment ;

original word ; intention	i	n	t	e	n	ϵ	t	i	o	n
comparison ; execution	ϵ	e	x	e	c	u	t	i	o	n
cost	1	1	1	0	1	1	0	0	0	0

The minimum edit distance of "intention" and "execution" is 5.

The operation path that produces the minimum edit distance (minimum edit path) would be ;

original word ; intention	i	n	t	e	n	ϵ	t	i	o	n
comparison ; execution	ϵ	e	x	e	c	u	t	i	o	n
path	D	S	S	C	S	I	C	C	C	C

Where C,S,D and I correspond to Correct, Substitution, Deletion and Insertion respectively.

For these examples each operation has a cost of 1, however different costs or weights can be assigned to the different operations. For example we could give substitutions a cost of 1 and insertions and deletions a cost of 0.7, in which case the distance of "intention" and "execution" is 4.4. Implementation of the algorithm that finds the minimum edit distance and the corresponding path is done via dynamic programming.

1. *begin*
2. *initialize* A , x , y , m $\leftarrow$ length[x] , n $\leftarrow$ length[y]
3. C[0,0] $\leftarrow$ 0
4. i $\leftarrow$ 0
5. *do* i $\leftarrow$ i + 1
6. C[i,0] $\leftarrow$ i
7. *until* i = m
8. j $\leftarrow$ 0
9. *do* j $\leftarrow$ j + 1
10. C[0,j] $\leftarrow$ j
11. *until* j = n
12. i $\leftarrow$ 0 ; j $\leftarrow$ 0
13. *do* i $\leftarrow$ i + 1
14. *do* j $\leftarrow$ j + 1
15. C[i,j] = min[C[i-1,j]+1 , C[i,j-1]+1 , C[i-1,j-1]+1-δ(x[i], y[j])]
16. *until* j = n

17. *until* i = m

18. *return* C[m, n]

19. *end*

1.5 Receiver Operating Characteristics Graph [14] [13]

A receiver operating characteristics (ROC) graph is a technique for visualizing classifier performance. ROC graphs have long been used in signal detection theory to depict the tradeoff between hit rates and false alarm rates of classifiers, while they are currently widely used in medical applications.

Assuming a classification problem using only two classes, each instance I is mapped to one class of the set ***p,n*** of positive and negative class labels. A classifier is a mapping from instances to predicted classes. We label the predicted class using the capital letters ***P,N***.

Given a classifier and an instance, there are four possible outcomes. If the instance is positive and it is classified as positive, it is counted as a ***true positive***; if it is classified as negative, it is counted as a ***false negative***. If the instance is negative and it is classified as negative, it is counted as a ***true negative***; if it is classified as positive, it is counted as a ***false positive***.Given a classifier and a set of instances (the test set), a two-by-two confusion matrix (also called a contingency table) can be constructed representing the possible outcomes from that set of instances.

Figure (1.2) shows a confusion matrix. The numbers along the major diagonal represent the correct decisions made, and the numbers off this diagonal represent the confusions (errors) between the classes. This matrix is the basis for many common classifier performance metrics, like accuracy and recall.

ROC graphs are two-dimensional graphs in which true positive rate is plotted on the Y axis and false positive rate is plotted on the X axis (equivalently, true negative and false negative can be used). A ROC graph depicts relative tradeoffs between benefits (true positives) and costs (false positives).

		True Class: p	True Class: n
Hypothesized Class	P	True Positive	False Positive
	N	False Negative	True Negative

Figure 1.2: Confusion Matrix

An example of a ROC graph can be seen in figure (1.3), where the points produced for different parameters of the same classifier have been connected by a line.

The lower left point (0%, 0%) in ROC space represents the strategy of never issuing a positive classification; such a classifier commits no false positive errors but also gains no true positives. The opposite strategy, of unconditionally issuing positive classifications, is represented by the upper right point (100%, 100%). The point (0%, 100%) represents perfect classification.

Informally, one point in ROC space is better than another if it is to the northwest (tp rate is higher, fp rate is lower, or both) of the first. Classifiers appearing on the left-hand side of an ROC graph, near the X axis, may be thought of as conservative: they make positive classifications only with strong evidence so they make few false positive errors, but they often have low true positive rates as well. Classifiers on the upper right-hand side of an ROC graph may be thought of as liberal: they make positive classifications with weak evidence so they classify nearly all positives correctly, but they often have high false positive rates. The preferred type of classifier depends on the class skew of the domain.

The diagonal line y = x represents the strategy of randomly guessing a class. For example, if a classifier randomly guesses the positive class half the time, it can be expected to get half the positives and half the negatives correct; this yields the point (50%, 50%) in ROC space. Overall a random classifier will produce a

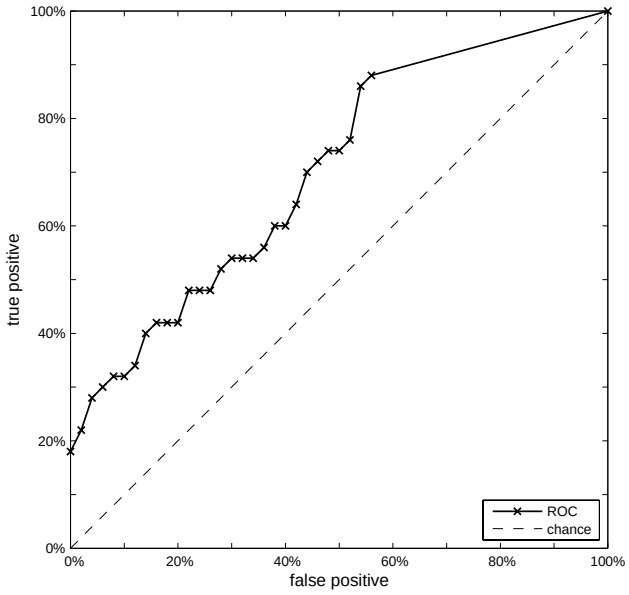

Figure 1.3: ROC graph example

ROC point that moves on the diagonal. For a classifier to appear in the upper triangle it needs to make use of the data.

Thus any classifier that appears in the lower right triangle performs worse than random guessing, so this triangle is usually empty in ROC graphs. Reversing such a classifier (inverting all it's decisions) results in it moving to the upper triangle.

Normally discrete classifiers (those outputting only a class label) define a single point in ROC space. That however can change when considering parametric systems. The obvious case and the one used in this thesis is a varying classification threshold, changing it produces multiple points in ROC space which in turn can define a curve. This curve can be used to fine-tune the classifier for a specific usage, like picking an appropriate threshold for a conservative classifier (low number of false positives).

Comparing different parametric classifiers through their ROC curves can be done in many ways depending on our demands. For example we could compare the total Area Under the Curve (AUC) if interested in overall performance or

there might be a Region Of Interest (ROI) within which we are limited. Further consideration should be given to the fact that ROC graphs don't take into account class skew (one class could be much more likely than the other) or error weight (mistakenly diagnosing cancer has a much lower cost than mistakenly giving a clean bill of health), factors like these usually help us define a region of interest in ROC space.

Chapter 2

Our approach

2.1 "Bayesian formulation of the speech recognition problem" [9]

The general formulation of the speech recognition problem using the bayesian formulation is

$$\hat{W} = arg\max_{W} P(W|O) = arg\max_{W} P(O|W)P(W) \tag{2.1.1}$$

Where O is the acoustic observation vector, W is any word sequence and $\hat{W}$ is the recognized (most probable) word sequence.
The probability of the recognized word sequence can also be calculated as

$$P_{LA}(W_r) = \sum_{W_c} P(W_r|W_c)P(W_c) \tag{2.1.2}$$

Where $P(W_r|W_c)$ is the word misclassification probability, the probability that W_r will be recognized given the speaker said W_c, $P_{LA}(W_r)$ is the probability of recognizing W_r and $P(W_c)$ is the probability that the speaker says W_c.
To simplify matters we assume that only word substitution errors can happen, no insertions or deletions. Then we can compute the word unigram and bigram probabilities for recognized word sequence W_r from n-gram models of correct word sequences. For unigrams we have

$$P_{LA}(w_r^1) = \sum_{w_c^1} P(w_r^1|w_c^1)P(w_c^1) \tag{2.1.3}$$

and for bigrams

$$P_{LA}(w_r^2|w_r^1) = \sum_{w_c^1,w_c^2} P(w_r^1, w_r^2|w_c^1, w_c^2)P(w_c^2|w_c^1)\frac{P(w_c^1)}{P_{LA}(w_r^1)} \tag{2.1.4}$$

These can be simplified further by assuming independence among word misrecognitions

$$P(w_r^1, w_r^2|w_c^1, w_c^2) = P(w_r^1|w_c^1)P(w_r^2|w_c^2) \tag{2.1.5}$$

2.2 Misrecognition models

The quantity needed in the above equations is $P(W_r|W_c)$, the misrecognition probability. This section discusses some methods of calculating it.

There are three main categories of models :

1. Assume that W_r and W_c are independent $\rightarrow$ $P(W_r|W_c) = P(W_r) \rightarrow P_{LA}(W_r) = P(W_r)$
2. Assume that $P(W_r|W_c)$ is a constant for any $W_r \neq W_r \rightarrow$ all words are equally confusable with any word in the corpus
3. Assume that $P(W_r|W_c)$ is a function of both W_r and W_c

Some model examples

$$P(W_r|W_c) = \begin{cases} p & W_c = W_r \\ \frac{1-p}{N-1} & W_c \neq W_r \end{cases} \tag{2.2.1}$$

$$P(W_r|W_c) = \begin{cases} p & W_c = W_r \\ \frac{1-p}{N(W_c)-1} & W_c \neq W_r \& L(W_c) = L(W_r) \\ 0 & else \end{cases} \tag{2.2.2}$$

$$P(W_r|W_c) = \begin{cases} p + (1-p)P(W_c)^a & W_c = W_r \\ (1-p)P(W_r)^a & W_c \neq W_r \& L(W_c) = L(W_r) \\ 0 & else \end{cases} \tag{2.2.3}$$

2.3 Simplified form

Given a probabilistic lexicon with alternate pronunciations for each word and with U_r and U_c being the phone sequences corresponding to W_r and W_c, we can express $P(W_r|W_c)$ as

$$\begin{aligned} P(W_r|W_c) &= \sum_{U_c,U_r} P(U_c,U_r,W_r|W_c) = \\ &= \sum_{U_c,U_r} P(W_r|U_c,U_r,W_c)P(U_r|U_c,W_c)P(U_c|W_c) = \\ &= \sum_{U_c,U_r} P(W_r|U_r)P(U_r|U_c)P(U_c|W_c) = \\ &= \sum_{U_c,U_r} P(U_r|W_r)P(W_r)\frac{P(U_r|U_c)}{P(U_r)}P(U_c|W_c) \end{aligned} \tag{2.3.1}$$

assuming $P(W_r|U_c,U_r,W_c) = P(W_r|U_r)$ and $P(U_r|U_c,W_c) = P(U_r|U_c)$ which is equivalent to a bayesian network of $W_c \rightarrow U_c \rightarrow U_r \rightarrow W_r$. In the equation, $P(U_r|W_r)$ and $P(U_c|W_c)$ are the probabilities of a sequence of phones given a sequence of words, which are pronunciation probabilities directly retrievable from a probabilistic pronunciation lexicon. $P(U_r|U_c)$ is the phone misrecognition probability, retrievable from a phone misclassification model. So in total we can get all required probabilities using : a phone misclassification model, a language model and a probabilistic pronunciation lexicon.

2.4 Baseline system

For the implementation we approximate Eq. (2.3.1) with

$$P(W_r|W_c) = \max_{U_r,U_c}[P(U_c|W_c)P(U_r|U_c)P(U_r|W_r)P(W_r)] \tag{2.4.1}$$

with U_r, U_c being the acoustic observation vectors and W_r, W_c being the word sequences. We are assuming that all phones are equiprobable and that U_r has the same number of phones for each U_r corresponding to word sequence W_r (the

later is true when we allow only for substitutions in the phone misclassification model). We also substitute the sum with the max, which should be a good approximation provided most of the probability mass is confined to a small amount of assumptions.

The recognition result will be the W_r assumption with the highest probability

$$\hat{W_r} = \arg\max_{W_r}[P(W_r|W_c)] = \arg\max_{W_r}[\max_{U_r,U_c}[P(U_c|W_c)P(U_r|U_c)P(U_r|W_r)P(W_r)]] \tag{2.4.2}$$

Which corresponds to the best path of a composition of a probabilistic pronunciation lexicon (L), it's inverse (L^{-1}), a phone misclassification model (M) and a language model (W). So our system will correspond to ;

$$L \circ M \circ L^{-1} \circ W \tag{2.4.3}$$

In a similar fashion we can get the W_c estimate given the recognized word sequence.

$$\hat{W_c} = \arg\max_{W_c}[P(W_c|W_r)] = \arg\max_{W_c}[\max_{U_r,U_c}[P(U_c|W_c)P(U_c|U_r)P(U_r|W_r)P(W_c)]] \tag{2.4.4}$$

Which can be represented by the same system, only using a different misclassification model.

The schematic representation is presented in figure (2.1).

The first component is a pronunciation lexicon representing $P(U_c|W_c)$, it's output is the combination of all possible pronunciations a speaker may use to utter the phrase W_c. That then passes through a misclassification model representing $P(U_r|U_c)$, mapping each phone to all possible confusions. Then all resulting phone sequences pass through an inverse pronunciation lexicon providing $P(U_r|W_r)$ which transforms them into all possible recognized word sequences W_r. Finally these word sequences go through a language model representing the $P(W_r)$ term of the equation, which adds grammar weight.

Intuitively, the first lexicon represents the speaker, the misclassification model and second lexicon are the phonetic part of the recognition process and the language model is the linguistic part. The system is implemented via fsm toolkit, by

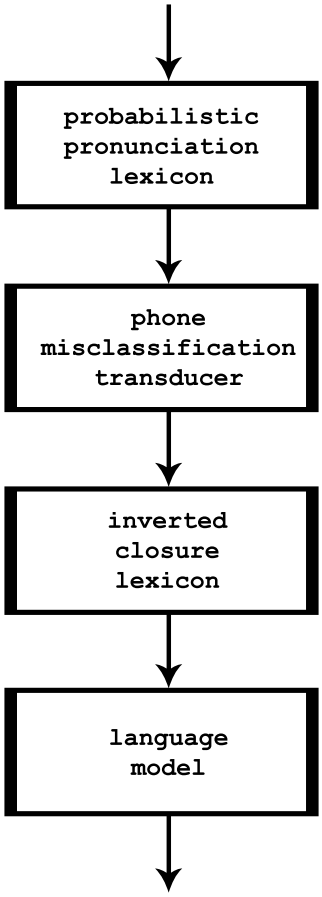

Figure 2.1: System diagram

composing the 4 finite state machines. Initially all automata were trained using the TIMIT dataset.

Each component of the system is implemented in multiple ways, following are their descriptions.

2.4.1 Lexica

The 2 lexica are the same, with one of course being inverted. Two alternatives were used

1. a classic pronunciation lexicon containing all alternates available in the

corpus with a cost of $-100ln(P(U_c|W_c))$ where $P(U_c|W_c) = \frac{count(U_c,W_c)}{count(W_c)}$.

2. a version contains only one pronunciation per word, the most likely one of the previous lexicon and is costless.

2.4.2 Misclassification

The misclassification model is a 1:1 mapping of all possible phone substitutions. It is trained by taking the minimum edit paths between a transcription and a phone recognition of the same text. It's costs are $-100ln(P(U_r^i|U_c^i))$, calculated by simply counting the instances where the transcription phone U_c^i is substituted by U_r^i in the recognizer output (a deletion is a substitution with ϵ, an insertion is a substitution of ϵ). Across the entire phone sequence U the cost is $-100ln(\prod_i P(U_r^i|U_c^i))$, where U_r^i is the $i-th$ element of phone sequence U. We assume phone misrecognition independence, the result of one phone recognition has no effect on following ones. For the purposes of this thesis we consider only phone substitution errors, insertions and deletions are ignored.

Part of the misclassification matrix can be seen in figure (2.2), the first column being the phones in the transcription result versus the first line which are the recognition result. For example, "aa" appears 846 times in the transcription, 137 of these correspond to "EPS" (a deletion) in the recognition. These numbers converted to probabilities (normalized by the total number of "aa" occurences in the trascription) create our model.

We also train a second misclassification model, used for $P(W_c|W_r)$ calculation. For the second misclassification model we follow the same training procedure, but map from recognition to transcription.

Finally we also create a uniform error probability model as the one illustrated in Eq. (2.2.1).

2.4.3 Language model

The language model is task-dependent, based on the one from the system we are trying to approximate. It's costs are $-100ln(P(W_c))$. We create unigram,

	EPS	aa	ae	ah	ao	aw	ay	b	ch	d	dh	eh	er	ey	f	g
EPS	0	19	23	57	22	18	19	15	30	52	18	18	19	21	14	16
aa	137	430	12	62	90	35	26	0	0	2	1	2	6	0	2	1
ae	86	9	462	25	1	21	23	0	1	1	0	96	2	8	1	0
ah	479	45	25	1132	34	27	27	6	0	4	2	62	48	2	1	1
ao	126	78	1	16	433	12	14	1	0	1	0	0	2	1	0	1
aw	13	15	13	12	7	136	1	1	0	1	0	2	0	0	0	0
ay	49	30	17	31	4	6	501	1	0	1	0	8	0	13	0	0
b	90	2	0	1	1	2	0	667	1	33	27	3	1	0	0	6
ch	32	0	0	0	0	0	0	0	178	5	0	1	0	0	0	0
d	378	3	9	7	1	3	9	23	1	1295	52	5	12	4	0	29
dh	215	0	0	1	2	0	1	34	0	51	437	0	0	1	2	3
eh	293	3	111	109	3	16	5	2	7	7	5	459	17	37	3	0
er	125	8	4	28	13	1	5	0	0	1	3	22	1220	5	2	0
ey	42	0	6	2	0	1	15	0	0	0	0	16	2	590	0	0
f	40	0	0	0	1	0	0	0	3	0	1	3	1	0	772	0
g	50	3	1	2	1	0	1	6	0	33	1	1	2	5	0	352

Figure 2.2: Misclassification Matrix

bigram and zerogram versions.

2.5 AURORA4 ; The real-life counterpart

AURORA4 is the name of the automatic speech recognition system used as a reference point. It is based on HMMs and uses tied-state triphone models. Specifically the iterative training process goes as follows ;

R1 → Initial 1-gaussian/3-state monophone models
hmm0

hmm1

...

hmm8

R2 → Cross-Word initial triphones 1-gaussian/3-state

hmm0

hmm1

hmm2

hmm3

R3 → Tied-state triphones

hmm10

...

hmm13 → 1-gaussian

hmm20

...

hmm23 → 2-gaussian

...

hmm60

...

hmm63 → 6-gaussian

It is trained and tested using the wall street journal dataset.
For the purposes of this thesis we train the system using clean (noise-less) data and use a 166 sentence testing dataset, again using noise-less data, for testing. The model used is the tied-state 1-gaussian one.

Chapter 3

Experimental Procedure

3.1 Adaptation

Adapting the system was required for it to become compatible with the testing data of AURORA4. This is the procedure followed.

3.1.1 Phone set

The phone set used by TIMIT and therefore the one used by our lexica and misclassification models is different than the one used by AURORA4. We used the maximum common subset of those 2 : $AUR4 \cap TIMIT$. Conversion from the old sets to the new one happens according to figure (3.1). The mappings were decided by looking at the previous misclassification matrix (phones become their most common misclassification) and by comparing the pronunciations of words between the lexica of the 2 systems. The phones "q","sp" and "sil" are used in the 2 systems to signify silence, but in different ways, so we remove them completely.

The phones in the lexica of both systems and the training files for the misclassification model are adapted to the new phone set.

3.1.2 Lexicon

The lexicon we had trained with TIMIT data didn't include too many words from the one used by AURORA4. Furthermore, we couldn't just use the AURORA4

AUR	-	converted to
sp	-	NULL
sil	-	NULL
zh	-	z

TIMIT	-	converted to
ax	-	ah
axr	-	er
dx	-	d
el	-	l
en	-	n
ix	-	ih
nx	-	n
q	-	NULL

AUR + TIMIT
EPS
aa
ae
ah
ao
aw
ay
b
ch
d
dh
eh
er
ey
f
g
hh
ih
iy
jh
k
l
m
n
ng
ow
oy
p
r
s
sh
t
th
uh
uw
v
w
y
z

Figure 3.1: Table of phone conversions

lexicon since it isn't probabilistic and we didn't have access to it's training corpus. So we went for a middle ground solution. We created a probabilistic lexicon based on AURORA4's with all pronunciations being equiprobable, then we combined that with the TIMIT-trained lexicon as

$$lex = lex_{TIMIT} \cup (lex_{AUR4} - lex_{TIMIT})$$

All words already existing in the TIMIT lexicon are left untouched, then the new words from AUR4 (and their equiprobable pronunciations) are added.

3.1.3 Language model

The language model used by AURORA4 is a smoothed back-off bigram in the .arpa format, with costs being $-100log_{10}(P(W_c))$. We convert it to a form compatible with fsm-toolkit and recalculate the costs. A unigram and a zerogram (uniform unigram) model are also created from the same file.

Language model weight was a recurring problem. We used multiple heuristic methods trying to get an appropriate value, one was comparing the value $average(\frac{linguistic\,cost}{acoustic\,cost})$ of the 2 systems, another was comparing the standard deviation of the fraction of costs for the two systems $stdev(\frac{linguistic\,cost}{acoustic\,cost})$ however the resulting weights didn't produce the expected results. The values we use in our experiments are the ones that produced the best results, but not necessarily the optimal ones. Specifically we use a language model weight of 1 for unigram experiments and a weight of 0.5 for bigram and zerogram models. Intuitively the weight should increase as language model class increases, but experimentally that wasn't the case.

3.2 Operation

The system accepts as input a file of word sequences and returns the results of N-best recognition for each sequence, along with the lingo-acoustic, linguistic and acoustic costs. A sample output file given the correct phrase 'in a fundamental' can be seen in figure (3.2).

in a fundamental	(2052.845142)	(1065.496735)	(987.34848)
and a fundamental	(2153.82909)	(1126.185426)	(1027.643723)
no fundamental	(2278.721944)	(1347.947937)	(930.773972)
any fundamental	(2287.354074)	(1353.540711)	(933.813401)
in the fundamental	(2290.035474)	(1329.834442)	(960.201004)

5-Best recognition given the correct phrase 'in a fundamental'.
The output is ; recognized phrase, lingo-acoustic cost, acoustic cost, linguistic cost.

Figure 3.2: Sample output of N-best recognition

Initially the input sentence is read and through the lexicon the transducer containing all pronunciations of it is produced. Figure (3.3) shows the pronunciation fsm for input phrase "hot water".

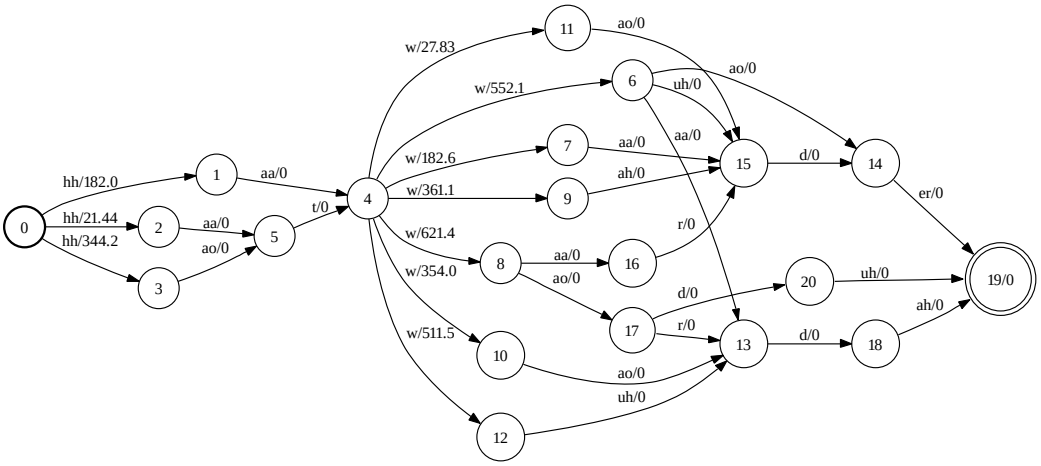

Figure 3.3: Pronunciations example

That is then composed with the misclassification model, giving us the total number of possible ways the phrase can be recognized by a front-end. Figure (3.4) shows part of the fsm for this stage, given the phrase "hot water".

The inverted lexicon converts those phones back into words and the language model adds grammar weight. Figure (3.5) shows part of the fsm for this stage, given the phrase "hot water".

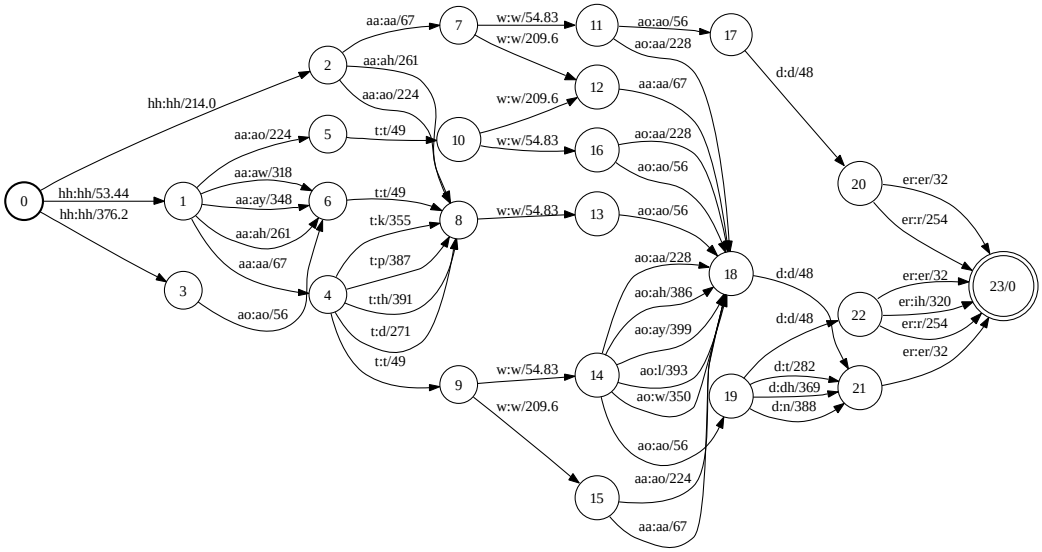

Figure 3.4: Misclassification example

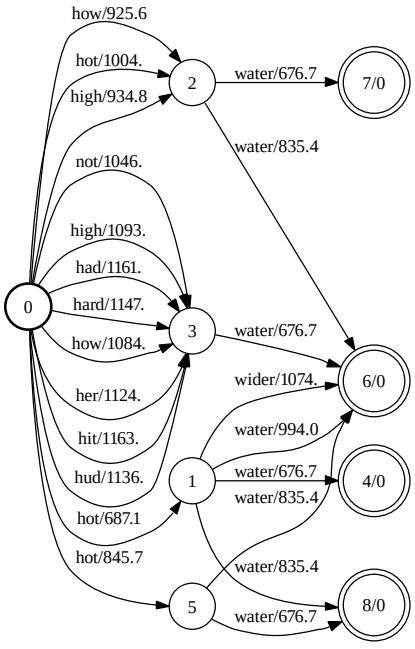

Figure 3.5: Output fsm example

From that fsm we extract the N best paths using Viterbi decoding.

3.2.1 Processing results

One of our targets is comparing the results of the real and virtual recognition systems given the same test data, another is using the virtual system to predict the behavior of the real one. In any case we need a method of comparing the output of the two systems, so we create a superscript that reads the testing dataset of AURORA4 and feeds the virtual system with the appropriate data. Then the results are processed ; we add a normalized probability for each path assuming there are no other valid paths besides those N (the assumption doesn't hurt us that much, since almost all probability mass really is spread across very few paths). A sample can be seen in figure (3.6). The total probabilities of correct/error are produced by taking the minimum edit paths between each candidate path and the correct/recognized words. Note that the probabilities of correct/error are based on the second word of the trigram (see appendix B), for example the minimum edit path between "in a fundamental" and "and a fundamental" is "S-C-C" which is identified as correct since the second word ("a") is correct and it's probability is added to $P(correct)$.
Most of our tests follow this procedure ;

1. Run a 166 phrase test in AURORA4
2. Get the minimum edit paths from the correct to the recognized phrases of AURORA4
3. From the minimum edit paths identify an appropriate amount of trigrams
4. Give the correct or recognized trigrams as input to the virtual system and run a 5-best or 10-best recognition

The output is multiple text files of the form seen in figure (3.6). Unless mentioned otherwise, we use an input of 1639 correct-correct-correct and 92 correct-substitution-correct trigrams for the unigram and zerogram tests, 50 correct-correct-correct and 50 correct-substitution-correct trigrams for the bigram tests.

C-S-C
CORRECT : in a fundamental
AURORA4: in the fundamental

PHR	C(LA)	C(A)	C(L)	P(LA)	
in a fundamental	(2052.845142)	(1065.496735)	(987.34848)	0.603173349186083	correct
and a fundamental	(2153.82909)	(1126.185426)	(1027.643723)	0.219722448762161	correct
no fundamental	(2278.721944)	(1347.947937)	(930.773972)	0.0630190217484311	
any fundamental	(2287.354074)	(1353.540711)	(933.813401)	0.0578073143392496	
in the fundamental	(2290.035474)	(1329.834442)	(960.201004)	0.0562778659640748	recognized

P(correct) = 0.822895797948244
P(error) = 0.177104202051756
P(recognized) = 0.0562778659640748

5-Best recognition given the correct phrase 'in a fundamental' and the AURORA4 recognized 'in the fundamental'.
The output is ; recognized phrase, lingo-acoustic cost, acoustic cost, linguistic cost, lingo-acoustic probability.

Figure 3.6: Sample output of N-best recognition, comparative

3.2.2 Classification

Given the output files of the previous stage and the probabilities contained in them, we can classify the results. For example we can set a $P(error)$ threshold of E and classify any result with a probability of error higher than that threshold as wrongly identified. To create the ROC graphs used for our evaluation we

1. Pick a feature to act as the criterion ($P(error)$,$P(correct)$,$P(substitution)$ etc)

2. Classify all results using all N samples as thresholds T, so a total of N classifications per experiment

3. Place all points (true reject, false reject) in ROC space creating a curve

Chapter 4

Evaluation

4.1 Behavior

Initially we want to see if the output of the virtual system matches that of the real recognition system using the probability of error against some often used metrics. Specifically we use $P(error|wordlengthinletters)$, $P(error|wordlengthinphones)$ and $P(error|number\,of\,alternate\,pronunciations\,of\,word)$ or rather the average values of those expressions across all available conditions. The comparison is against the probability of error of AURORA4 across it's entire test set. The diagrams can be seen in figures (4.1), (4.2) and (4.3).

All models provide a reasonably good approximation given word length in letters and phones, at least in the range of values for which we have a lot of samples (4-8 phones, very short and very long words are much less common). None of the models does really well given the number of alternate pronunciations, however that is reasonable since the 2 systems use different lexica and therefore have different numbers of alternate pronunciations for the same words (the number of pronunciations in the virtual system's lexicon was used for the comparison).

Distributions

For the purpose of classifying samples we create distributions of the output, $P(P(error)|category)$ and $P(category|P(error))$. An example can be seen in figures (4.4) and (4.4) with the distributions from a unigram test.

The concept of $P(P(error)|category)$ or $P(P(error))$ in general may seem

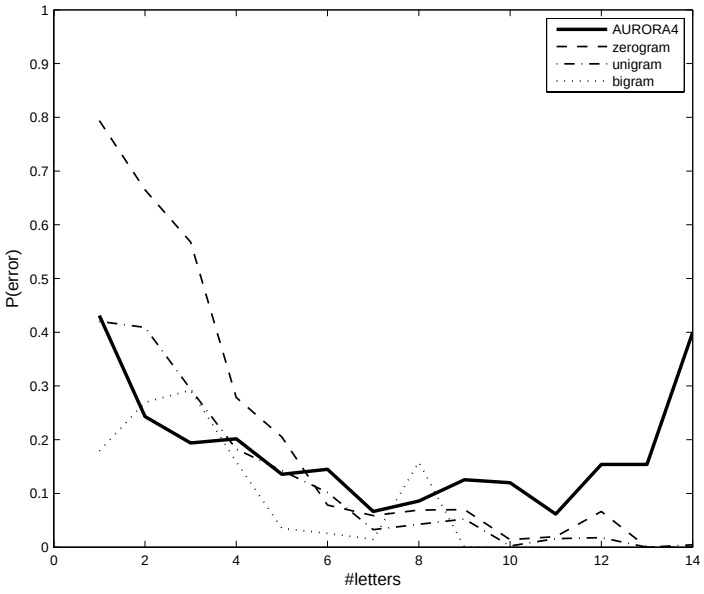

Figure 4.1: P(error) given word length in letters

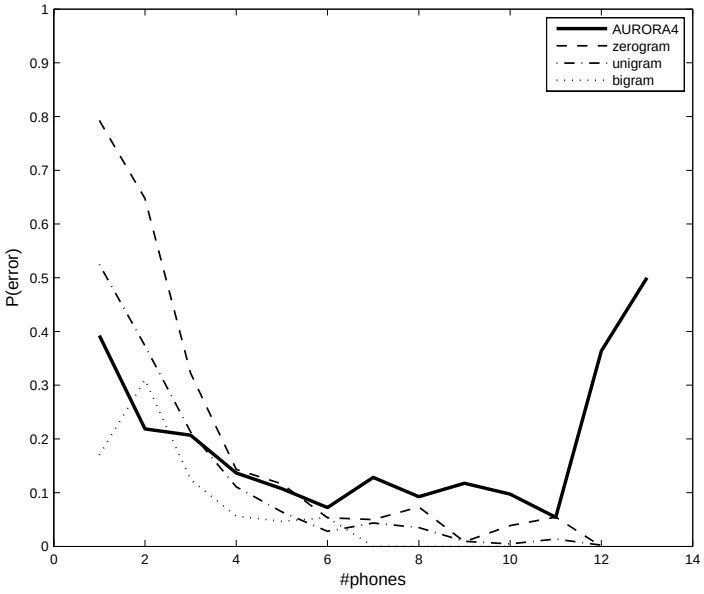

Figure 4.2: P(error) given word length in phones

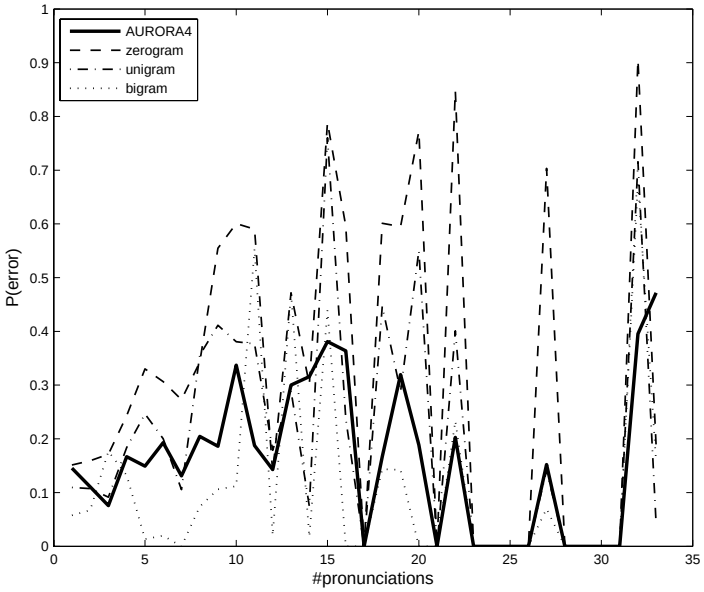

Figure 4.3: P(error) given the number of alternate pronunciations of the word

confusing at first. As an example, in figure (4.4), about 60% of all correct word samples from the recognizer are assigned an error probability of around zero (the samples are quantized), while about 0% are assigned an error probability around 1. Similarly, 30% of all recognition error samples are assigned a probability of error around zero, while about 5% are given a probability around 1.

The $P(P(error)|category)$ diagram shows that misrecognitions are assigned a higher average probability of error, with a larger probability mass spread across the higher $P(error)$ values than correct samples. However a lot of mass is gathered around the zero mark, for both correct and wrong samples. This indicates that the system underestimates confusability, perhaps an artifact of the simple phone misrecognition model (no insertions or deletions) and places a limitation to any classification efforts. It should be noted that corresponding graphs for all our tests follow the same form, including the probability mass around zero, though the exact values are different in each case.

The $P(category|P(error))$ diagram is here mostly for illustrative purposes,

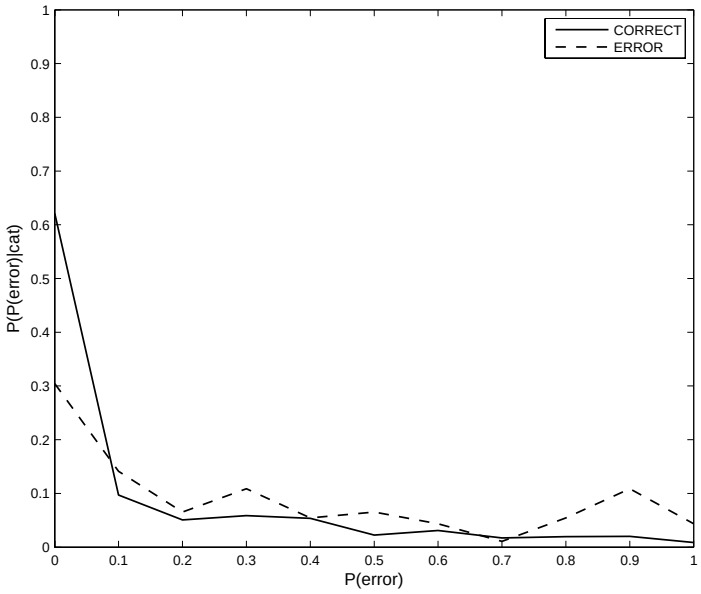

Figure 4.4: $P(P(error)|category)$ distributions

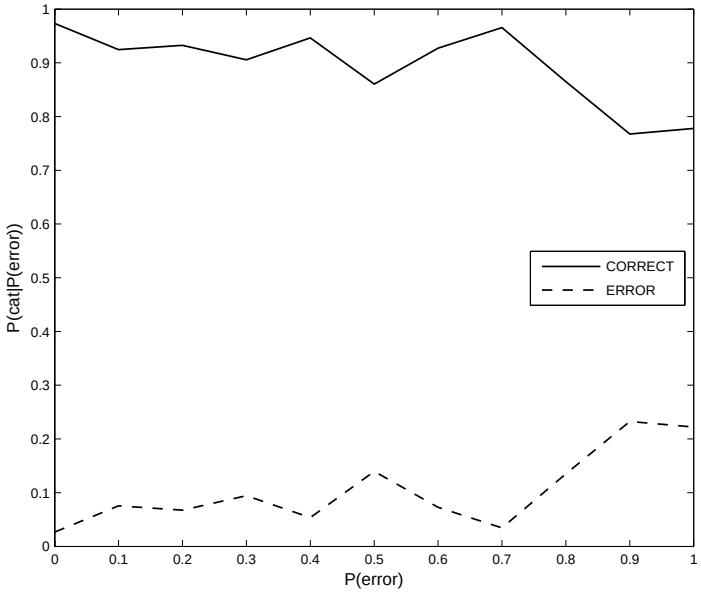

Figure 4.5: $P(category|P(error))$ distributions

the much larger a-priori probability of a correct recognition means that a correct recognition is always the much more likely outcome. This suggests that if all errors (false accept and false reject) have equal weights then we should just classify all samples as correct. However the weight of false accept (wrong word accepted as correct) is typically much higher than that of false reject (correct word rejected as wrong) and the varying weight of these errors is what pointed as to ROC analysis.

4.2 Predictive power

As a post process all samples of the output are classified as correct (accepted) or wrong (rejected), trying to predict the output of the real speech recognition system. The criterion used is the probability of error of each sample (see figure (3.6), samples with a probability of error higher than the threshold are rejected, others are accepted. We define ROC space as true reject (errors correctly identified as errors) over false reject (correct samples wrongly classified as errors).

Varying the classification threshold from 0^- to 1^+ we get multiple points in ROC space, which give us a complete view of the classifier's potential. For presentation purposes we connect those points creating ROC curves, though our classifier is discrete and so doesn't produce a continuous ROC curve. Following are comparative ROC diagrams, illustrating the factors that affect prediction accuracy. Unless mentioned otherwise, the diagrams refer to classification using the total probability of error as the criterion.

It should be noted that the graphs are not directly comparable to each other, since each corresponds to different common parameters.

4.2.1 Prediction

Initially we give the system the correct trigram and ask it to predict if the second word will be correctly identified by the speech recognition system.

Language Model

We compare the three available language models using the normal misclassification model and the single pronunciation lexicon (a necessity to accommodate the bigram model). Figure (4.6) shows the results. As expected, the higher the class of the language model used the more accurate the classification. Experimentally this seems to be the most important factor.

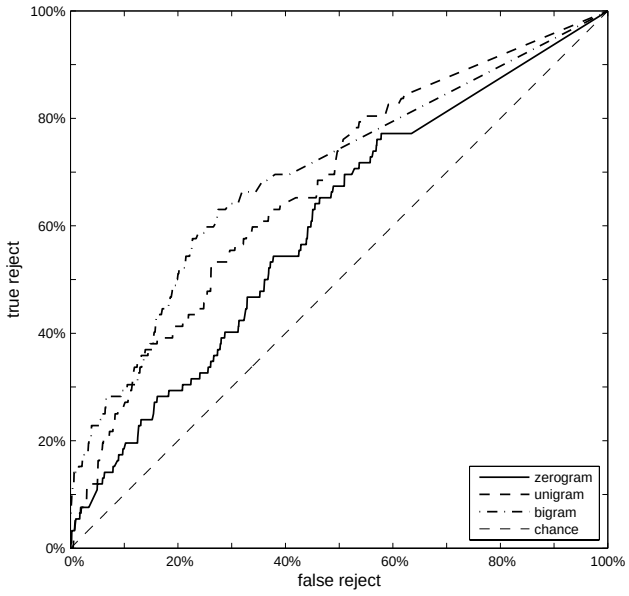

Figure 4.6: ROC using different class language models, prediction

Misclassification

Another major factor is the misclassification model used. For comparison we use 2 models as illustrated in section (2.4.2), one being the "normal" (default) one, the other using a uniform probability of error. Figures (4.7), (4.8), (4.9) show the effect of using the two misclassification models combined with the three language models, all tests utilize the probabilistic pronunciation lexicon. All 3 diagrams show minor differences. In this case we would expect more of an improvement by using the "normal" model.

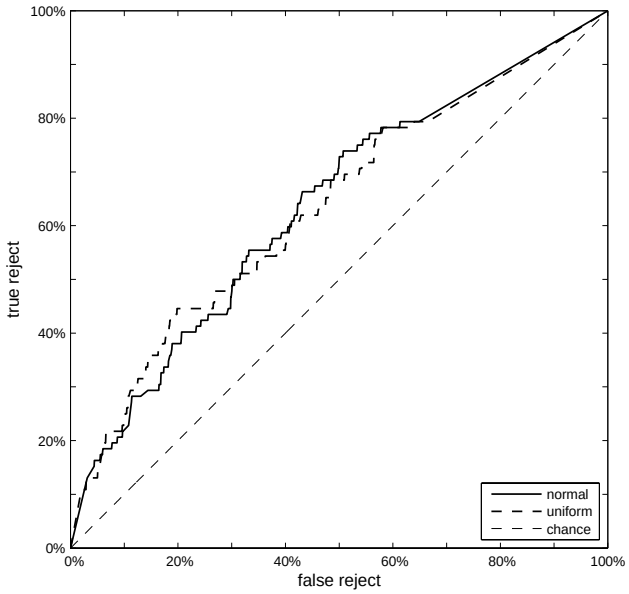

Figure 4.7: ROC using different misclassification models, zerogram, prediction

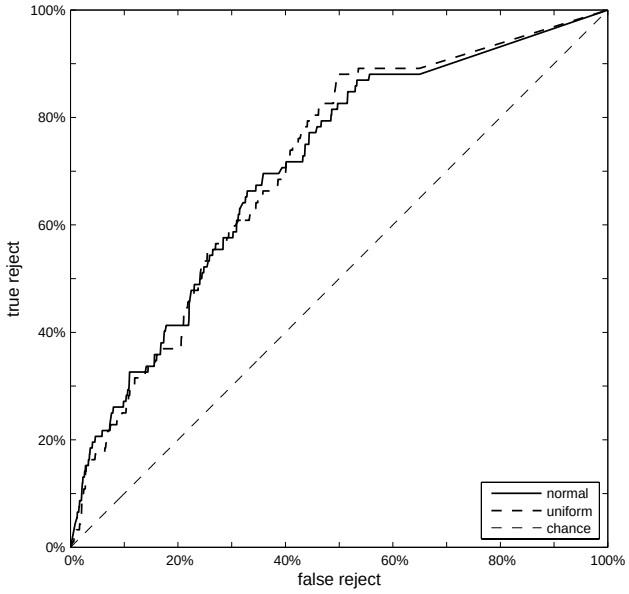

Figure 4.8: ROC using different misclassification models, unigram, prediction

Classification Criteria

Classification accuracy is affected by the feature we use as the criterion. By default we use the total probability of error (substitution, insertion and deletion), for comparison we also create ROCs using as criteria ;
the probability of substitution error
the difference of log probabilities between the most likely correct and the most likely wrong path.
Figures (4.10), (4.11), (4.12) show the effect of using the three criteria combined with the three language models. As seen in the figures, the substitution error accuracy improves comparatively to the total error accuracy as the language model class increases. It is much worse in the zerogram case, but comparable in the unigram and bigram cases. This is expected, accuracy deteriorates as more non-substitution errors occur and the zerogram model has a natural tendency towards deletions.

$P(error)$ and $log(P)difference$ are more competitive, the latter usually performing better in the lower left and upper right quadrants of the diagram, the former inbetween. The latter produces a larger area under the curve, but the former produces higher peaks. The advantage of the $log(P)$ criterion seems to be that the concentrated probability mass (see figure (4.4)) is distributed better.

4.2.2 Diagnosis

In the more realistic case, we have the recognized word and want to decide whether it is correct or not. So we present the recognized trigram to the system and expect an estimate of which correct word produced the recognized second word, if the estimated correct word is identical to the recognizer output then we assume the result to be correct.

Language Model

As previously, figure (4.13) shows the effect of using different class language models. And again using a language model of a higher class produces a more accurate classification.

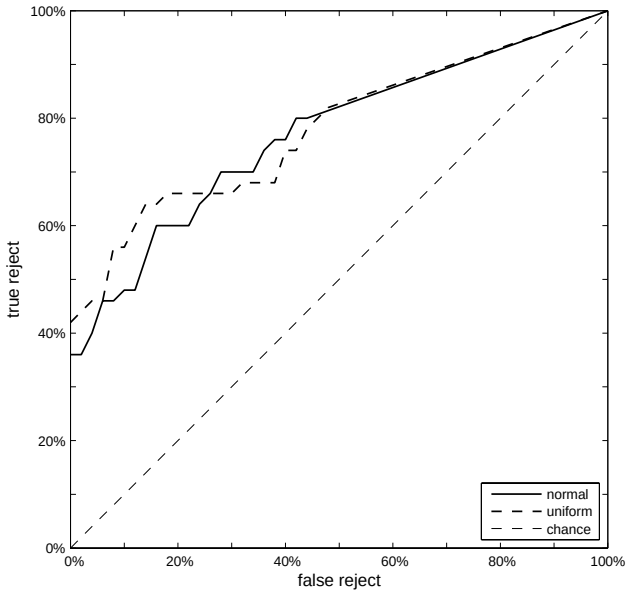

Figure 4.9: ROC using different misclassification models, bigram, prediction

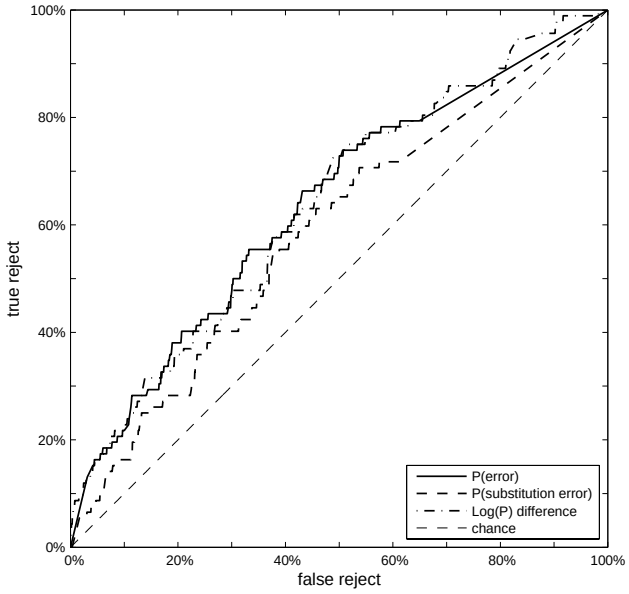

Figure 4.10: ROC using different criteria, zerogram, prediction

Misclassification

As previously, figures (4.14), (4.15), (4.16) show the effect of using the two misclassification models combined with the three language models. Again all models show minor differences.

Classification Criteria

Figures (4.17), (4.18), (4.19) show the effect of using the three criteria combined with the three language models. Again substitution error proves a poor criterion, while the other two are competitive.

4.2.3 Prediction VS Diagnosis

Since we assume only substitution errors and the test data we use only produces substitution errors in the real speech recognition system, we expect error prediction accuracy to be similar whether given the correct or the recognized word (trigram). Figures (4.20), (4.21), (4.22) show prediction accuracy using the 2 methods with the three language models. The zerogram model works noticeably better in diagnosis, the unigram model shows little difference while the bigram model works significantly better in prediction. Overall the prediction performance improves comparatively to diagnosis word performance as the language model class increases.

It seems the zerogram language model provides a different angle, giving it the edge in diagnosis. By it's nature it takes into account things like the higher probability of a small word in the context of a large word being misrecognized. Things that the higher order models don't do.

4.2.4 Acoustic VS Linguistic Influence

Previous tests have shown that changing the class of the language model has an important effect in the system's performance, while changing the phone misclassification model doesn't. To further explore this we compare 2 systems, one having a strong linguistic but weak acoustic component and one being the opposite.

The "linguistic" system is composed of a bigram language model, a single pronunciation lexicon and a uniform error probability phone misclassification model. The "acoustic" system is composed of a zerogram language model, a probabilistic pronunciation lexicon and the normal phone misclassification model. Figure (4.23) shows the result of the prediction experiment.

It serves as a confirmation of what we've seen before, the linguisticically stronger system performs much better.

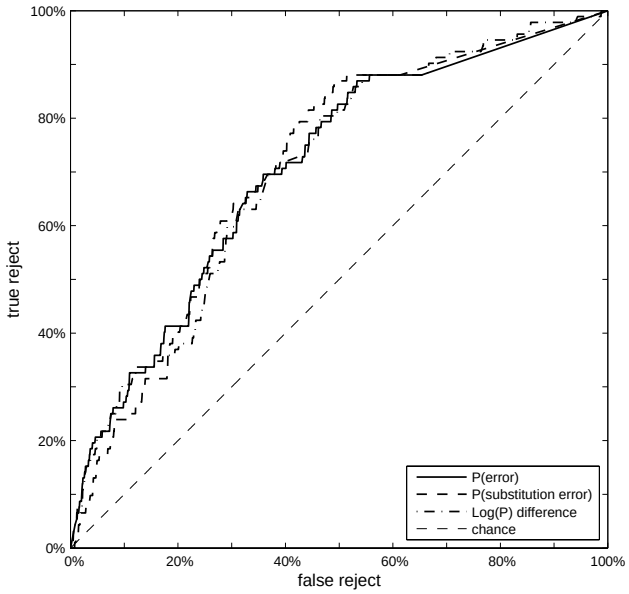

Figure 4.11: ROC using different criteria, unigram, prediction

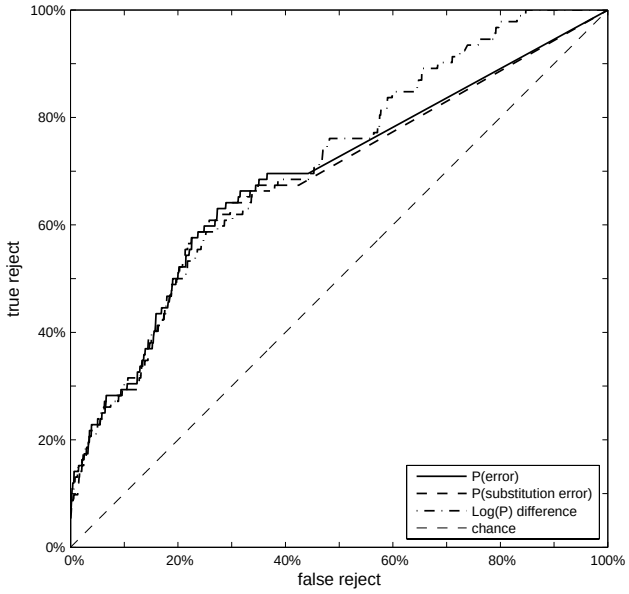

Figure 4.12: ROC using different criteria, bigram, prediction

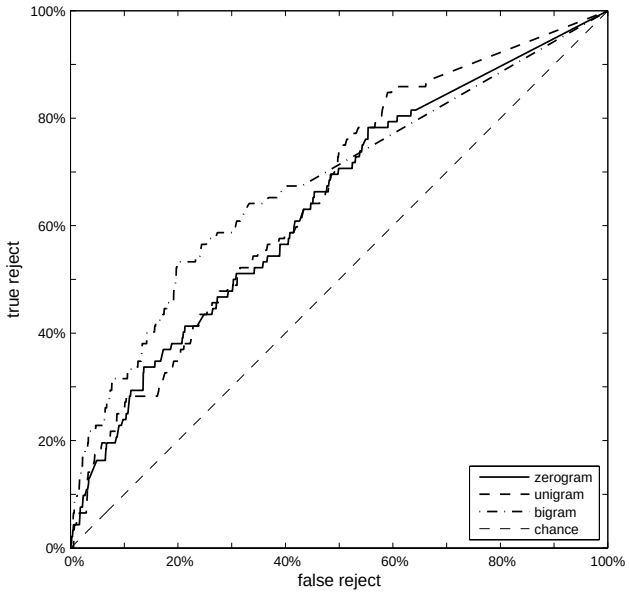

Figure 4.13: ROC using different class language models, diagnosis

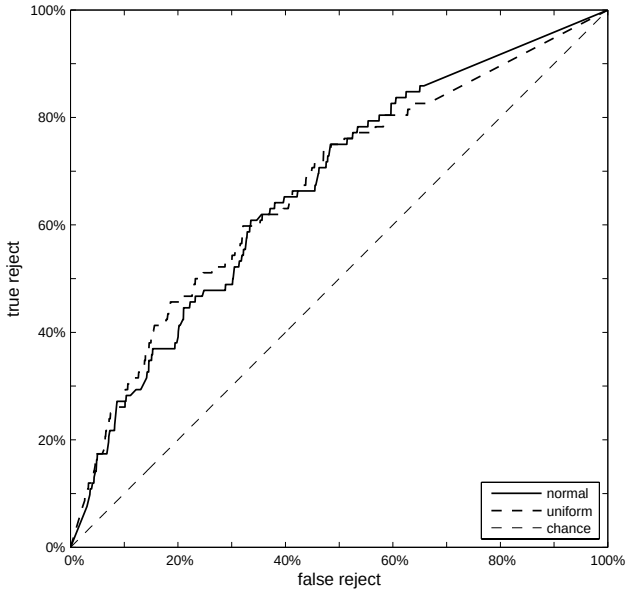

Figure 4.14: ROC using different misclassification models, zerogram, diagnosis

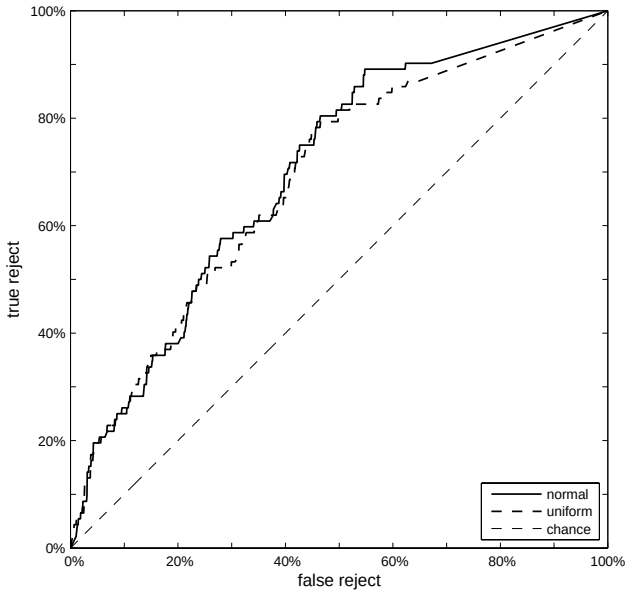

Figure 4.15: ROC using different misclassification models, unigram, diagnosis

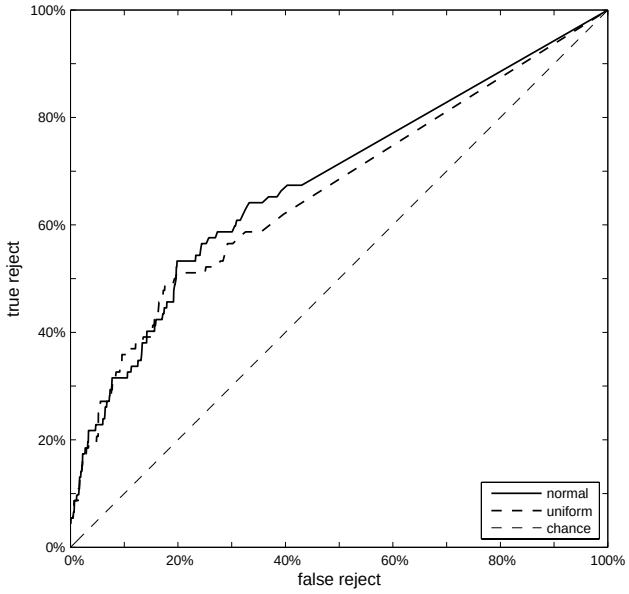

Figure 4.16: ROC using different misclassification models, bigram, diagnosis

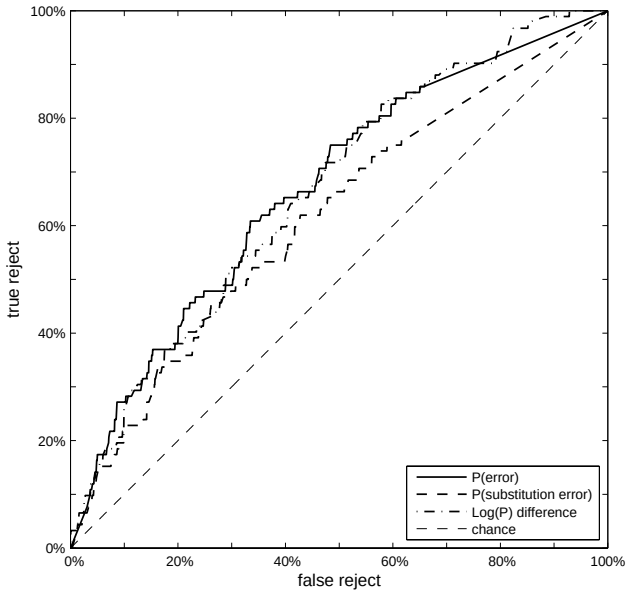

Figure 4.17: ROC using different criteria, zerogram, diagnosis

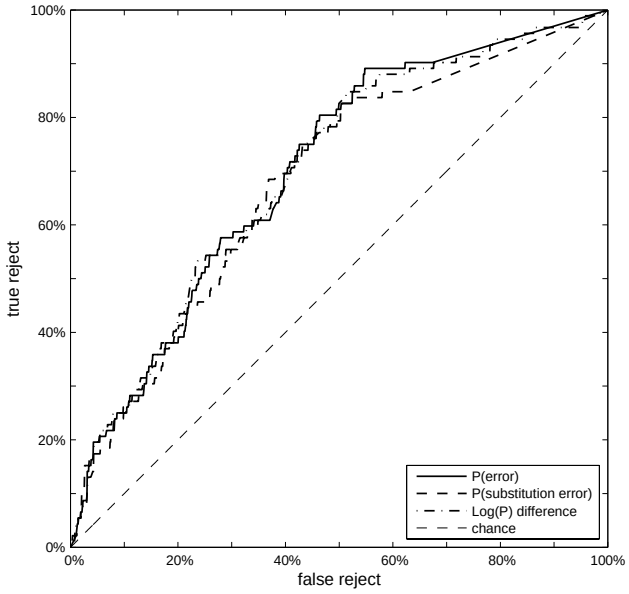

Figure 4.18: ROC using different criteria, unigram, diagnosis

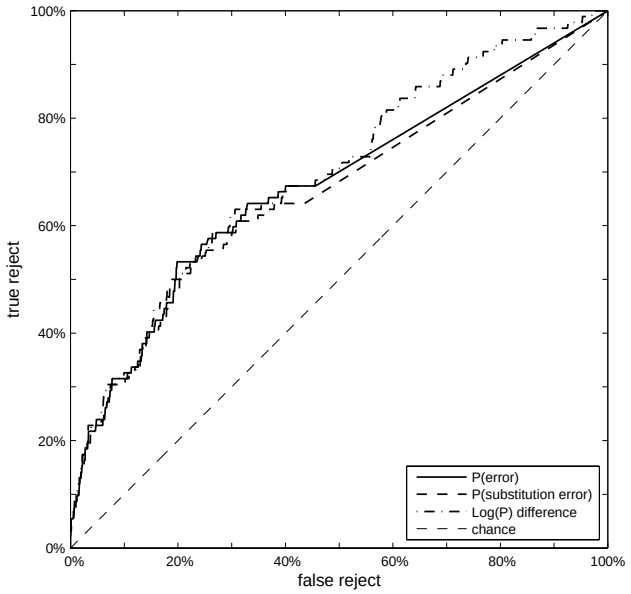

Figure 4.19: ROC using different criteria, bigram, diagnosis

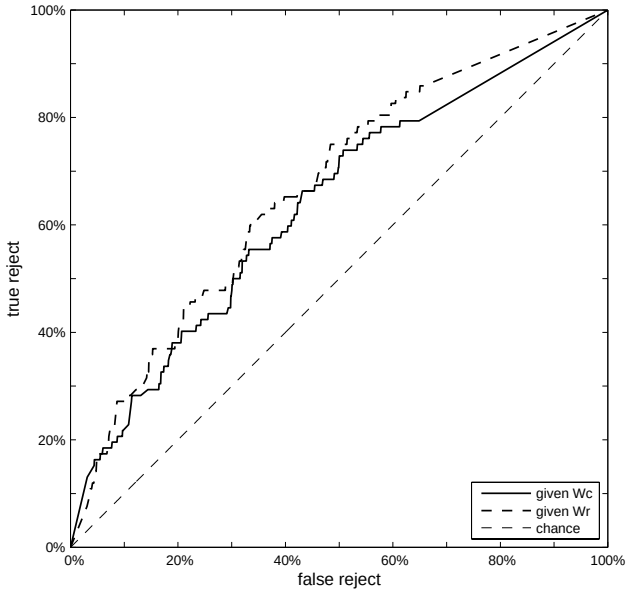

Figure 4.20: ROC using Wc vs Wr, zerogram

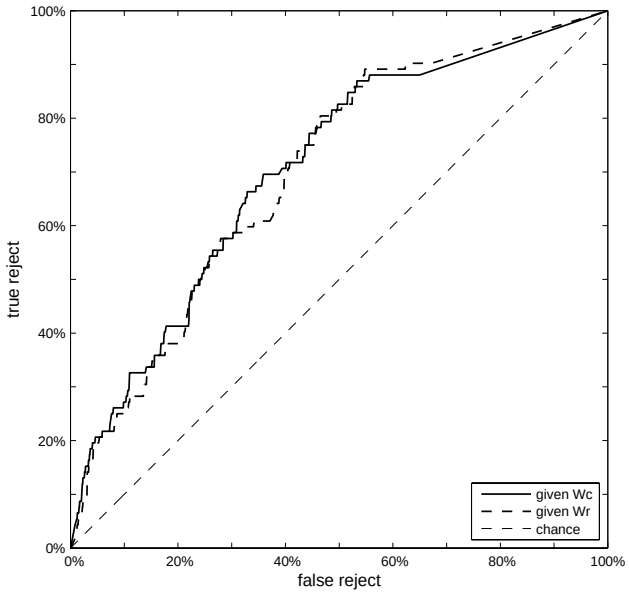

Figure 4.21: ROC using Wc vs Wr, unigram

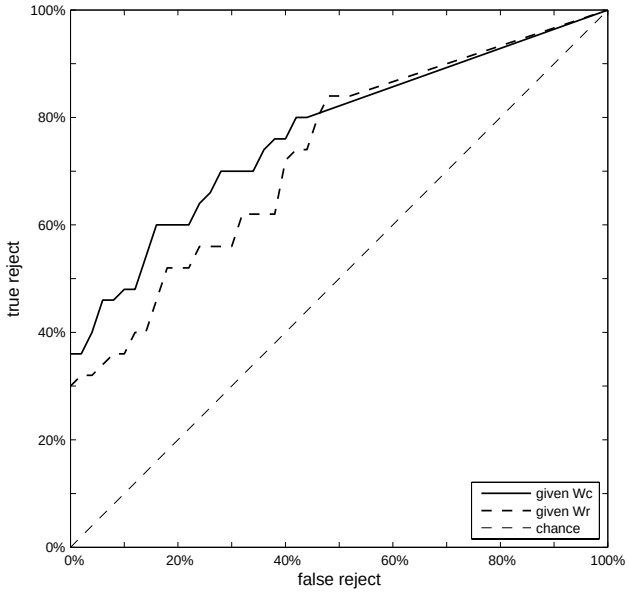

Figure 4.22: ROC using Wc vs Wr, bigram

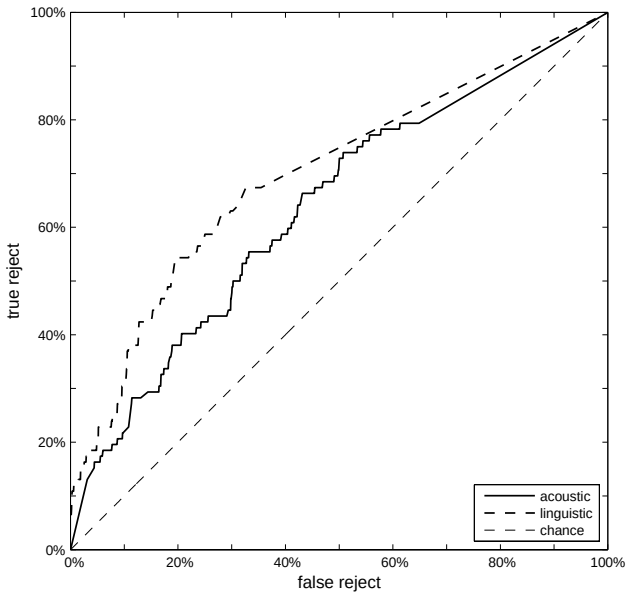

Figure 4.23: Influence of acoustic and linguistic components

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Overall the classifier has fair but not spectacular performance, yet shows potential. As it is, it can perform well as a conservative classifier, indicated by fairly good results in the lower left quadrant of ROC space. The best results provided by the composition of the most complex models (probabilistic lexicon, normal misclassification, bigram language model) are inconclusive due to the limited amount of samples used in that scenario, but still encouraging.

The language model is the main factor in all our tests. As exhibited, the effect of the language model's class is more important than all other factors combined. One obvious reason for this is that the language model used is identical to the one used by the actual ASR system while the acoustic components are not.

Alternatively the acoustic components used are too weak. The phone misclassification model doesn't feature insertions or deletions, the phone set conversion required for compatibility inserts noise, while the lexicon features a lot of words with equiprobable pronunciations. Furthermore the system has no concept of word boundaries, like silence between words. Improvements in these sections should enhance the overall performance of the system noticeably.

5.2 Future Work

This being preliminary work and one plagued by technical limitations (see Appendix (B)), there are many possible improvements.

An obvious one would be increasing the complexity of the phone misclassification model. Including phone insertion and deletion errors was omitted from our tests because the results produced were not particularly good (unclear why) but it is the next logical step. Further improvement should come from incorporating phonetic context, since phone misrecognition does depend on the neighboring phones.

The lexicon could be altered to include word boundaries (perhaps in the form of silence phones). Presently the system has no idea of where each word ends when they are converted to phones (see figures (3.3) and (3.4)), the problem being confounded by the absence of any type of duration data. This would not necessarily improve results.

Discarding information, like the least likely paths (pruning), during the stages of the process would both lessen the computational perplexity of the system and better approximate the operation of a speech recognition system decoder.

Further experiments can be done in the classification stage using different classification criteria. It should be noted however, that the form of the probability distributions produced by this system limit what we can expect from such efforts.

Electing to substitute a sum for a max (see Eq. (2.4.1)) is a necessity from an implementation standpoint. Assuming execution can be accelerated enough, it should be be corrected. It should be noted that while the system extracts paths using that formula, the sum of probabilities we use in the result processing stage (see (3.2.1)) alleviates the problem somewhat.

Appendix A

FSM Toolkit [7]

A.1 Overview

The AT&T FSM library is a set of general-purpose software tools available for Unix, for building, combining, optimizing, and searching weighted finite-state acceptors and transducers. Finite-state transducers are automata for which each transition has an output label in addition to the more familiar input label. Weighted acceptors or transducers are acceptors or transducers in which each transition has a weight as well as the input or input and output labels.
The original goal of the AT&T FSM library was to provide algorithms and representations for phonetic, lexical, and language-modeling components of large-vocabulary speech recognition systems.
This imposed the following requirements:

1. Generality: to support the representation and use of the various information sources in speech recognition

2. Modularity: to allow rapid experimentation with different representations

3. Efficiency: to support competitive large-vocabulary recognition using automata of more than 10 million states and transitions.

The mathematical foundation of the library is the theory of rational power series, which supplies the semantics for the objects and operations and creates opportunity for optimizations such as determinization and minimization.
System Components:

1. AT&T FSM library: includes about 30 stand-alone commands to construct, combine, determinize, minimize, and search weighted finite-state machines (FSMs). These commands manipulate FSMs by reading from and writing to files or pipelines.

2. Dot and Dotty: programs used by the FSM library to visualize graph representations of FSMs (Graphviz).

A.2 Commands

1. **FSM COMPILATION AND DISPLAY**

   fsmcompile takes input representing an FSM from file file or standard input, and sends to standard output its binary encoding. The input should be the textual representation of an FSM. FSM states, input symbols and output symbols are represented in the input by non-negative numbers, unless the options -s symbols, -i symbols, -o symbols are used. These options allow state, input symbols and output symbols, respectively, to be given textual names, where symbols files give the translation from those names and numbers . The input should be an acceptor, unless the -t option is given, in which case it should be a transducer.

   fsmprint prints the input FSM on standard output using same textual format as fsmcompile accepts as input. States, input symbols and output symbols are printed in numeric form, unless the options -s symbols, -i symbols, -o symbols are used to provide textual names for states, input symbols and output symbols, respectively.

   fsmdraw sends to standard output a dot(1) graph representation of the input FSM (The command dot -Tps can be used to convert from dot format to PostScript.) States, input symbols and output symbols are displayed in numeric form, unless the options -s symbols, -i symbols, -o symbols are used

to provide textual names for states, input symbols and output symbols, respectively. The options -w x and -h x set the page width and height (in inches), -f fontname sets the font name (default is Times-Roman), -F n sets the font size (in points), -p use portrait mode (default is landscape), and -v displays vertically (i.e., top-to-bottom; default is left-to-right).

2. **FSM CONSTRUCTION AND COMBINATION**

fsmunion returns the union of one or more input FSMs.

fsmconcat returns the concatenation of one or more input FSMs, in the order specified by the command-line arguments.

fsmclosure returns the Kleene closure of the input FSM. With the -p option, the empty string is not added, that is, Kleene "+" is used instead of Kleene "*".

fsmrmepsilon returns an equivalent FSM with no epsilon transitions. The input FSM must have no negative cost epsilon cycles.

fsmintersect returns the intersection of two or more acceptors. Each input FSM accepts string s iff the output FSM accepts s with the costs combined by the EXTEND operation.

fsmcompose returns the relational composition of the input FSMs, in the order given in the command line. With two input FSMs, for example, if the first machine transduces string s1 to s2 and the second machine transduces s2 to s3, then the output machine will transduce s1 to s3 with the two costs combined by the EXTEND operation. If an input machine is an acceptor, it is treated as a transducer from the language it accepts to itself.

fsmdifference returns the intersection of the acceptor fsm1 with the complement of the costless, deterministic, epsilon-free acceptor fsm2.

3. **FSM MINIMIZATION AND EQUIVALENCE**

fsmconnect returns an FSM from which any states and arcs in the input that do not lie on a path from the start state to a final state have been removed. With the -t option, it returns exit status 1 if the output has no states, which is useful for testing the input for emptiness.

fsmdeterminize determinizes the input FSM, which must be determinizable. Epsilon arcs are treated the same as other symbols.

fsmminimize returns the minimal deterministic FSM equivalent to the input FSM, which must be a deterministic acceptor. Epsilon arcs are treated the same as other symbols.

fsmarccollect COLLECTS costs on identically-labelled arcs between the same source and destination states.

fsmcompact uses a heuristic procedure to return an FSM equivalent to fsm. but possibly smaller. It works for arbitrary FSMs.

fsmequiv exits with zero status if fsm1 and fsm2 are equivalent. The inputs must be deterministic, epsilon-free acceptors.

4. **FSM SEARCH**

fsmbestpath returns the lowest-cost path from the start state of the input FSM to a final state. The path is encoded as a (single path) FSM. With the -n nbest option, the nbest lowest-cost paths are returned. The output

is encoded as an FSM that is the union of the individual paths in increasing cost order. With the -c cthresh and -N nthresh options, the input FSM is pruned as in fsmprune, limiting the nbest search. With the -u option, all paths returned will be distinct strings.

fsmprune returns those states and arcs that lie on paths whose total path cost in fsm is within cthresh of the lowest cost path and at most the nthresh best such states. input epsilons. In each case, if there is a cycle with respect to the sorting criterion, fsmtopsort returns the input FSM unsorted.

fsminfo sends to standard output the following information about the input FSM – its FSM class and whether it is an acceptor or transducer. With the "-n" option, various numeric information is printed, including the number of states, number of transitions, final states, epsilon transitions, strongly-connected components, accessible states, and co-accessible states. With the "-p" option, FSMProps is called on the input, which will return pre-computed information about the FSM, such as whether it is cyclic, costless, non-negative, or deterministic. If pre-computed information about a property is not supported by the FSM class, a "?" is printed for it. With the "-t" option, values for all FSM properties are printed (by explicit tests run on the FSM if needed). See "fsmprops.h" for the set of defined FSM properties. With the "-c" option, the FSM class properties are printed, which include the FSM operations supported by that class. See "fsmprops.h" for the set of defined FSM class properties. With the "-q w" option, quantiles in intervals of width w are printed for various data including state in-degree, state out-degree, input label, output label, and arc cost. With the "-b q1" option, the quantiles begin at q1 (default: 0.0), and with the "-e q2" option, the quantiles end at p2 (default: 1.0). The "-v" option is equivalent to "-tcn -q4".

Appendix B

Practical Limitations

Our system is produced by composing 4 finite state machines as shown in figure (2.1). After the first step, we have a machine with all the possible combinations of word pronunciations, so the number of paths in the automaton will be

$$\prod_{i=1}^{n}(\#\, pronunciations\, of\, w_i) \tag{B.0.1}$$

The second step involves the misclassification matrix, which will multiply the total number of paths by

$$(\#\, of\, phones) avg(\#\, of\, misclassifications\, per\, phone) \tag{B.0.2}$$

The third step involves converting phone sequences back into words, the increase in path numbers is hard to compute, but it is proportional to the number of phones in each path (more phones will produce more possible outputs).
Finally the language model increases the total number of paths by adding backoff arcs (each word sequence can be produced in a lot of different ways), the number of arcs added is obviously proportional to the number of words in each sequence.

Overall, on average, the number of arcs in the final stage is proportional to the input sequence's length in words and phones and to the product of alternate pronunciations for all words in the sequence. These factors however are not independent, a phrase containing more words is likely to also have a longer phonetic length and it will probably have a higher product of alternate pronunciations. As also verified experimentally the system's demands in processing time and space increase exponentially as phrase length increases.

That makes it impractical or even impossible to use large phrases in conjunction with the bigram language model. As an example the composite machine given the correct trigram 'companies with experience' has a size of 2.4 Gigabytes, 273727 states, 159374132 arcs and requires over 7 Gigabytes of virtual memory to produce and process (obtain best paths etc). The size of the machines produced even for very small word sequences means that our system requires a 64-bit addressing space to work (so a 64-bit processor and operating system). Using a 'normal' sized sentence of 10 or so words would require impossible amounts of time and storage space to complete.

Similar problems are cited in related work on the use of transducer composition [11][1][12][17] and is mostly countered by simplifying the system. Proposed solutions include the use of incremental language modeling [3] and on-the-fly transduction [6].

In this thesis we pick the option of making concessions. For all our tests we used trigrams (an example seen in figure (3.6)), focusing on the second word. Keeping the words before and after the one we are interested in means we keep a linguistic context and since each of those words is composed of at least one phone we also maintain phonetic context. One problem this introduces is that insertion errors have no real meaning. We give a trigram as input to the system and then get it's minimum edit paths with the output N-best paths to see what happens to the second input word. However that second word already exists in the correct trigram, it can not be inserted, it can only be correct, substituted or deleted. Furthermore we use specific trigrams from AURORA4's test set, namely those where the first and third word are recognized correctly, as an effort to keep as much context as possible. As can be seen in figure (3.6), the result is labeled 'C-S-C', which is the result of AURORA4's recognition ; the first word is correct, the second one is substituted, the third one is correct.

It should be noted that only the bigram model has such a problem, using a unigram or zerogram language model allows us to ignore this for the most part. However, since the bigram implementation is our main focus and to keep results comparable between the systems we used trigrams for all tests.

Bibliography

[1] Beng T. Tan, Yong Gu, Trevor Thomas, "*Word Confusability Measures For Vocabulary Selection In Speech Recognition.*" in Proc. IEEE Automatic Speech Recognition and Understanding Workshop, 1999.

[2] Francis, W. and Kucera, H., "*The Brown Corpus (Revised and Amplified).*" Brown University, 1979.

[3] Hans J.G.A. Dolfing, I.Lee Hetherington, "*Incremental Language Models For Speech Recognition Using Finite State Transducers.*" in Proc. IEEE Automatic Speech Recognition and Understanding Workshop, 2001.

[4] Jurafski, Martin, "*Speech and Language Processing.*" Prentice Hall , p 174-191, 2000.

[5] M. Mohri and F. Pereira and M. Riley, "*Weighted Finite State Transducers in Speech Recognition.*" in Proc. ISCA ITRW Automatic Speech Recognition: Challenges for the Millenium, pages 97-106, 2000.

[6] M. Mohri, M. Riley, D. Hindle, A. Ljolje, and F. Pereira, "*Full expansion of context-dependent networks in large vocabulary speech recognition.*" in Proc. ICASSP, 1998.

[7] Mehryar Mohri, Fernando C. N. Pereira and Michael D. Riley, "*AT&T FSM Library documentation.*" http://www.research.att.com/~fsmtools/fsm/, 1998.

[8] Moffat, A., "*Implement the PPM data compression scheme.*" IEEE Transaction on Communications, 38(11): 1917-1921, 1990.

[9] Potamianos A., "*Lingo-Acoustic Perplexity Measures for Automatic Speech Recognition.*" Technical University of Crete , Personal Notes, 2007.

[10] R. Duda, D. Stork, P. Hart, "*Pattern Classification.*" John Wiley & Sons, 2000.

[11] T. Hain, "*Implicit pronunciation modelling in ASR.*" in Proc. ISCA Pronunciation Modeling Workshop, 2002.

[12] Timothy J. Hazen, I. Lee Hetherington, Han Shu, and Karen Livescu, "*Pronunciation Modeling Using A Finite-State Transducer Representation.*" in Proc. ISCA Pronunciation Modeling Workshop, 2002.

[13] Tom Fawcett, "*Using Rule Sets to Maximize ROC Performance.*" ICDM , p 131-138, 2001.

[14] Tom Fawcett, "*An introduction to ROC analysis.*" Pattern Recognition Letters 27, 2006.

[15] V. Digalakis, "*Introduction to Speech Processing Ch11.*" Technical University of Crete , Class Notes.

[16] Witten, I.H. and Bell, T.C., "*The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression.*" IEEE Transactions on Information Theory, vol. 37(4), 1991.

[17] X. Mou, S. Seneff, and V. Zue, "*Context-dependent probabilistic hierarchical sub-lexical modelling using finite state transducers.*" in Proc. Eurospeech, 2001.