Speech Recognition for Chatino

Vijay John

LIN 679HB

Special Honors in the Department of Linguistics

The University of Texas at Austin

May 2009

Anthony C. Woodbury

Department of Linguistics

Supervising Professor

Table of Contents

Acknowledgments

There are several people I would like to thank for making this work possible. First, I wish to thank Mr. Bernard Rapoport and Prof. Robert King for generously awarding me the Rapoport-King Scholarship. The report below follows directly from the proposal I had sent for the scholarship. In supporting me with their contribution, Mr. Rapoport and Prof. King have helped to provide a valuable resource for the Chatino Language Documentation Project (CLDP) here at the University of Texas.

I would also like to thank Hilaria Cruz, a member of the CLDP and native-speaker of the SJQ variety of Chatino. She has always expressed her enthusiastic support for this work ever since it was proposed. More importantly, though, she was also very instrumental in the creation of this work. All of the sound files used in this work were created by her and with her help. In addition, she always provided me with an analyzed transcript of each of the texts used to create the sound files.

I offer my sincere thanks to Prof. Tony Woodbury, who was my advisor for this work. When I first brought up the idea behind this work and proposed submitting it as a proposal with his recommendation, he agreed immediately. He has made a number of useful suggestions in addition to taking great interest in this work.

In addition to Prof. Woodbury, I would like to thank several professors in the Linguistics Department who have taught me topics that have directly or indirectly contributed to my understanding of the subjects that were applied to this thesis, specifically: Prof. Scott Myers, Prof. Megan Crowhurst, Prof. Nora England, Prof. Katrin Erk, and Prof. Ian Hancock.

Finally, thanks go to Prof. Richard Meier and Prof. Stephen Wechsler for supporting me in applying to various awards and scholarships. Thanks also to Mr. Ben Rapstine for suggesting my name for various opportunities.

Chapter One: Introduction

This report describes the steps we used to create a speech recognizer for Chatino. Chatino is a Zapotecan language of the Otomanguean language family spoken in Oaxaca, southern Mexico. San Juan Quiahije Chatino (SJQ) is one variety spoken in the village of San Juan Quiahije in the district of Juquila. A speech recognizer for SJQ must deal with certain complications involved in documenting Chatino. For example, Chatino is a tonal language with extensive tone sandhi, so finding the pronunciation of words in isolation (i.e. the "lexical forms") often involves inference based on sandhi rules. Also, Chatino is spoken by a small population (relative to other indigenous languages such as Nahuatl and Zapotec), so generating interest in the documentation project is an issue.

Developing Sphinx to recognize continuous speech in SJQ will help generate interest. Sphinx is a voice recognition system (developed at Carnegie Mellon University) that has been applied primarily to English; however, applications to a few other languages have been proposed as well. Applying Sphinx to SJQ will have three main advantages for documentary linguistics on this variety of Chatino. The first advantage is the creation of surface phonemic mappings for one variety of Chatino (i.e. SJQ). The second is that transcription of SJQ could be automated and that ultimately, lexical forms could be generated by a computer. The third and perhaps most important benefit is that a speech recognition system would substantially increase appreciation among Chatino-speakers for the fact that they speak a language in its own right and not merely an indigenous "dialect." The voice recognition system will have two technological advantages as well: application of voice recognition to lesser-studied languages spoken by populations that are small in size and use in intercultural technology adoption by Chatino-speakers.

In order to apply Sphinx to SJQ, a phonetic dictionary first needed to be developed. Such a dictionary helps a computer to recognize spoken words in SJQ by relating the acoustic data to the words. The next component that is necessary is an acoustic corpus of approximately 10,000 words. From this corpus, sounds are broken up into triphones. After breaking the sounds up in this manner, the data is integrated with Sphinx, and Sphinx's parameters will be fine-tuned. It may also be necessary to modify Sphinx, but there are no plans to modify the statistical algorithm used in the voice recognition system.

Sphinx is trained using acoustic models developed from transcribed audio corpora. Part of the acoustic model is a dictionary that decomposes words to be recognized by the speech recognizer into phones. Sphinx recognizes these phones in new speech by relating each phone with some probability; this is used along with more information about the language to recognize this speech. Carnegie Mellon has developed a tool, called lmtool, for creating, among other things, phonetic decompositions of words. However, this tool seems to use English-based pronunciations from a large pronunciation dictionary called cmudict. (Incidentally, our conclusion is based on publicly accessible versions of the lmtool script, such as Simple_LM available through SourceForge.) This does not result in the proper phonemic decomposition for words in SJQ. A further complication is that SJQ uses tones that are relevant in proper recognition of the spoken words.

While it is possible to develop a pronunciation dictionary for SJQ, this is a time-consuming process. Instead, we have developed a method that allows us to realize phonetic decomposition through a two-stage process. In the first stage, we use rules based on linguistic properties of SJQ to create a decomposition of written words into sound units. The next stage decomposes these sound units into sets of phones. By breaking down words into sound units, the problem of phonetic decomposition is reduced to a problem of mapping a finite number of sounds into phones. This list is considerably smaller than a pronunciation dictionary such as Carnegie Mellon's cmudict (CMU 2009).

In creating a speech recognizer for SJQ, it is important to consider what type of recognizer is to be constructed. There are two types of recognizers that can be created using Sphinx. One is an "isolated word recognizer," which would be able to recognize only one word at a time. The other is a "continuous speech recognizer," which would be able to recognize an entire sentence at a time if fully and properly developed. Both types of recognizers require several instances of "phones," as they are called in Sphinx. However, the definition of a "phone" depends on the type of recognizer being used. For an isolated word recognizer, a "phone" means an entire word. For a continuous speech recognizer, a "phone" means a triphone, i.e. a set of three sounds (or "phones" as defined in linguistics).

The type of recordings required for an isolated word recognizer is different from the recordings required for a continuous speech recognizer. For an isolated word recognizer, it is necessary to create several (5-10) recordings of the same word. This may be done by making several recordings of about 10 words with regular pauses in between. Then each recording may be divided into about 10 recordings consisting of one word each, yielding 50-100 recordings in all of isolated words. By contrast, a continuous speech recognizer only requires several instances of triphones. This is more convenient, since several instances of triphones are more likely to occur in recordings than several instances of an entire word.

To create a speech recognizer using Sphinx, it was necessary to follow several steps, as shown in Figure 1:

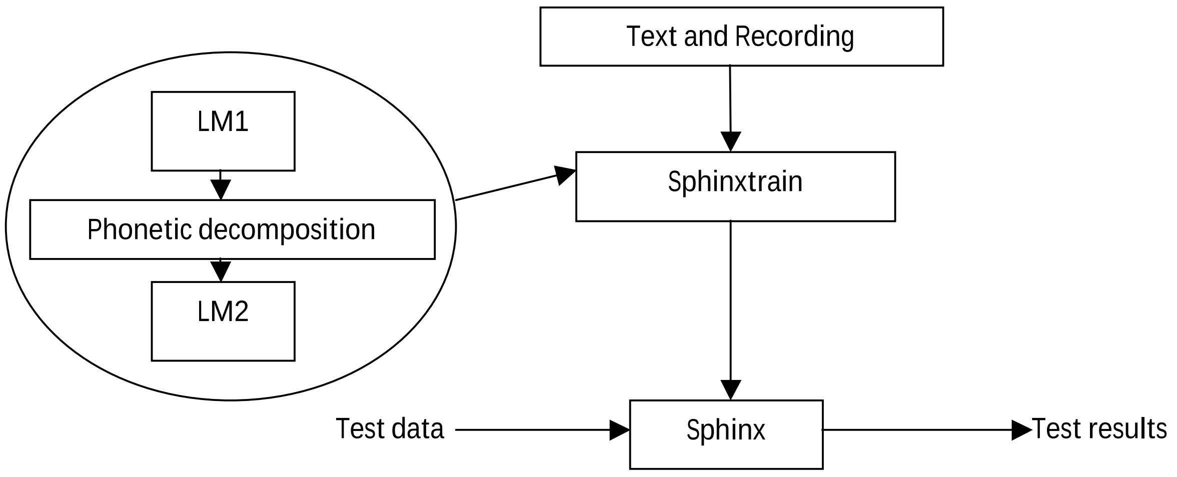

Figure 1. Diagram of basic steps involved in creating a voice recognizer with Sphinx. Sphinxtrain is part of the Sphinx application that takes training data as input and transfers it to Sphinx, i.e. the speech recognizer. The training data includes (a) recordings and their transcripts as well as (b) a Language Model (LM). The Language Model is a set of programs, one of which is a method of phonetic decomposition to break up words into phones. "LM1" and "LM2" refer to the rest of the Language Model; "LM1" precedes the program for phonetic decomposition and "LM2" follows it. Sphinx then uses the testing data as input; using the training data, it generates results.

Thus, to create a speech recognizer using Sphinx, it is always necessary to have recordings with their transcription. The recordings and transcription are always included as training data along with Sphinx's Language Model (LM). This training data is then used as input in Sphinx. When Sphinx is tested with testing data, it uses the training data to return what it heard. The information that Sphinx returns is what is called the "testing results" in the above diagram.

We will explain Sphinx in further detail later in this report. However, before we can look at the speech recognizer in detail, it would also be useful to have some information of the language to which is being applied, because we will be looking at specific examples in that language. In this case, that means that we must understand the phonemic system of SJQ, which is detailed in the next chapter.

Chapter Two: Phonemic System of SJQ

The orthography used in this report is based on previous work by E. Cruz (2004), Rasch (2002), Cruz, Cruz, and Stark (2007), and H. Cruz (2008), among others. The phonemic system of SJQ is outlined in the tables below, with the phonemes written in the same orthography used throughout this report. SJQ has 34 phonemic consonants, as shown in Table 1:

	Bilabial	Apico-dental	Laminal-alveolar	Palatal	Velar	Labialized velar	Glottal
Stops	p	t	ty	ky	k	kw	7
Prenasalized stops	mb	nd	ndy	ngy	ng	ngw	
Affricates		ts	ch				
Prenasalized affricates		ndz	ndzy				
Nasals	m	n	ny				
Preglottalized nasals & glides		7n	7ny	7y		7w	
Fricatives		s	x	jy		jw	j
Glides				y		w	
Laterals		l	ly				
Flaps		r					

Table 1. Consonants

and five oral vowels, as shown in Table 2:

	Front	Back
High	i	u
Mid	e	o
Low	a	

Table 2. Oral vowels

There are also four nasal vowels in SJQ. Note in Table 3 that *u* becomes *on* when nasalized; thus, there is no phoneme **un*.

	Front	Back
High	in	
Mid	en	on
Low	an	

Table 3. Nasal vowels

Both oral and nasal vowels can be glottalized in SJQ:

	Front	Back
High	i7	u7
Mid	e7	o7
Low	a7	

Table 4. Glottalized oral vowels

	Front	Back
High	in7	
Mid	en7	on7
Low	an7	

Table 5. Glottalized nasal vowels

SJQ has ten tones that are contrastive in isolation, i.e. four level tones and six contour tones. As explained in (Cruz and Woodbury, 2006), there are minimal pairs between words that differ only in tone, e.g. *skwa1* "soup," *skwa3* "swam," and *skwa4* "chayote." These contrastive tones are shown in Table 6 below:

Level	1, 2, 3, 4
Rising	20, 32, 42, 40
Falling	14, 24

Table 6. Tones

The highest level tone is represented by the number "0," and the lowest is represented by "4." Contour tones are represented by numbers in which the first digit represents the pitch at which the contour starts and the last digit represents the pitch at which the contour ends. (For example, tone "24" begins at tone "2" and falls to tone "4").

Every syllable in SJQ must include at least one consonant, one vowel, and one tone. More specifically, the syllabic structure of SJQ is (N)(C)CV(n)(7)T, where "N" is a nasal, "C" is a consonant, "V" is a vowel, and "T" is a tone. Thus, all syllables in SJQ must have an onset and tone, but they cannot have a coda except for the glottal stop "7." Note that SJQ is an isolating language, so all morphological roots are monosyllabic, as (Cruz and Woodbury, 2006) point out.

In addition to the ten tones listed above, four other tones also occur in SJQ: 0 (a level tone), 10 (a rising tone), 04 (a falling tone), and 140 (a falling-rising tone, i.e. a tone that falls and then rises). H. Cruz (2008) provides the following examples of tones 10 and 04 occurring in isolated words: *sya10* "heart" and *tyu04* "cute." However, these four tones do not contrast with other tones in isolation. All four of these tones may occur in the context of *tone sandhi*. This means that they may occur when one word occurs next to another word. This type of situation is explained in the next chapter, which includes an example of each of these additional four tones.

Chapter Three: Tone Sandhi in SJQ

In a tonal language, when two words are juxtaposed (each word having its own tone), the tone of one word may change under the influence of the other. This process is known as *tone sandhi*, and there are rules indicating how tones change when they occur next to each other. It is an important feature of the phonology of SJQ, because the meaning of an individual word in continuous speech cannot be determined without taking tone sandhi into account.

One example from (Cruz and Woodbury, 2006) involves the words in SJQ for "soup" and "Easter." Both words are pronounced *skwa1* in isolation. However, when they are followed by the word *kan742* "that," their tones are different. "That soup" is pronounced *skwa1 kan742*, but "that Easter" is pronounced *skwa10 kan742*. For this reason, (Cruz and Woodbury, 2006) analyze the two words as having different underlying tones: The word for "soup" has the underlying tone "1," whereas the word for "Easter" has the underlying tone "1+0," i.e. a floating tone. They note that when a syllable carrying a floating tone 1+0 is followed by another syllable with tone 42, the first syllable (with underlying tone 1+0) undergoes tone sandhi. Thus, its tone changes from 1 (as an isolated word) to 10 (before tone 42).

Another example in which the tone of the first syllable undergoes tone sandhi due to the presence of a floating tone is the following: The word meaning "we will pull out" in SJQ is pronounced *ston14* in isolation, but when it is followed by the word *ti3* "string," it is pronounced *ston140*. Thus, "we will pull out string" is pronounced *ston140 ti3* and not **ston14 ti3*. This is because the underlying tone of the word meaning "we will pull out" is 14+0.

Tone sandhi is also demonstrated when *skwa1* is followed by the adjective *ko1* "big." "Big soup" is pronounced *skwa1 ko1*, but "big Easter" is pronounced *skwa1 ko0*. In this case, tone sandhi affects the pronounciation of the second syllable (underlyingly "*ko1*"), changing tone 1 to tone 0 after an underlying tone "1+0."

We have now seen three examples of floating tones in SJQ. In addition, all words in SJQ that are pronounced with tone 4 or tone 24 in isolation have an underlying "strong" or "weak" tone. Weak tones tend to undergo tone sandhi more often than strong tones. For instance, the word for "tortilla" is pronounced *yja4* in isolation, but "you ground tortillas" is pronounced *yo1 yja24*. Thus, the tone 4 in *yja4* changes to a tone 24 after tone 1. However, if the word *jya4* "sugarcane" is preceded by the word *yo1* "you ground," its tone does not change due to sandhi. "You ground sugarcane" would be pronounced *yo1 jya4*.

For this reason, a distinction between the two types of words with tone 4 must be observed. Thus, *yja4* is analyzed as having an underlying weak tone 4, whereas *jya4* has an underlying strong tone 4. In underlying forms or representations, to distinguish between strong and weak tone 4, tone is indicated only on words with a strong tone 4 (Woodbury 2008). Thus, the word for "tortilla" is written (underlyingly) *yja* (i.e. with a weak tone 4) while the word for "sugarcane" is written *jya4*.

Strong and weak tones can also be found for tone 24. The word for “twenty” is pronounced *kla24* in isolation, but in the expression *nya1 kla04 ti3* “he is making twenty strings,” the tone 24 in *kla24* changes to a tone 04, being preceded by a tone 1. By contrast, the tone 24 in the word *ti24* ‘ten’ does not change in the expression *nya1 ti24 ti3* “he is making ten strings.” The explanation for this is that *kla24* has an underlying weak tone 24, while *ti24* has an underlying strong tone 24. Weak tone 24 is indicated as “24W” while strong tone 24 is indicated as “24S.” (“W” and “S” stand for “Weak” and “Strong,” respectively). The underlying forms for the words “twenty” and “ten” in SJQ would be written *kla24W* and *ti24S*, respectively.

When using transcripts of continuous speech in SJQ, it is important to make sure that the transcript contains no mistakes in the marking of tones. There is considerable speaker variation in SJQ, and this may affect transcripts of the spoken language. Often, there are mistakes in transcripts of SJQ, and the tones with which words are marked are inconsistent with the lexical forms of those words. In the interests of creating a continuous speech recognizer that is to be (ultimately) speaker-independent, words should be transcribed consistently. If we check the transcription to make sure that words follow the sandhi rules for SJQ, we will be able to ensure greater consistency.

However, tone sandhi is not a problem in an isolated word recognizer, because an isolated word recognizer cannot deal with continuous speech. It requires regular pauses between words, and this means that words are pronounced in isolation. In Chapter Four, the details of how an isolated word recognizer works is described. In later chapters, we will focus on the steps involved in creating a continuous speech recognizer and compare continuous speech recognizers to isolated word recognizers.

Chapter Four: Isolated Word Recognizer

In an isolated word recognizer, there is no need for phonetic decomposition because only one word needs to be recognized at a time. An isolated word recognizer uses as input a series of words. These words are recorded with regular pauses after each word. If the recognizer is then tested with any one of the recorded words, it should be able to return that word.

I recorded the same set of ten SJQ words over five large sound files. These words were separated into different sound files, obtaining fifty sound files. These sound files were used to train Sphinx. The words are spoken separately and Sphinx recognizes each word without influence from other words, thus the recognizer can be considered as an isolated word recognizer. The main steps involved in creating an isolated word recognizer are shown in Figure 2.

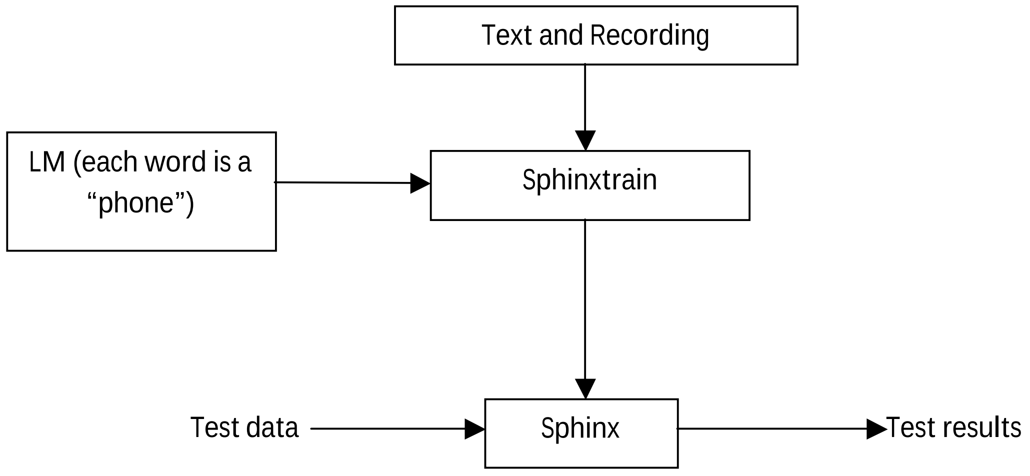

Figure 2. Diagram of basic steps involved in creating an isolated voice recognizer. Sphinxtrain is part of the Sphinx application that takes training data as input and transfers it to Sphinx, i.e. the speech recognizer. The training data includes (a) recordings and their transcripts as well as (b) a Language Model (LM). The Language Model is a set of programs. In these programs, each word in the training data is treated as a "phone."

In an isolated word recognizer, Sphinx treats each word as a separate "phone." These are not traditional phones in the sense this is used in phonetics; in the context of Sphinx, a "phone" is simply something that can be recognized as a pattern. Speech recognition in Sphinx uses patterns built from analysis of the different instances of a "phone." Thus Sphinx has to be "trained" on instances of each word. Data from this training is later used to recognize words.

Sphinx Installation and Use

Sphinx is usually synonymous with the voice recognizer. The recognizer can use patterns obtained from another system called SphinxTrain. The recognizer and the trainer are separate in the sense that patterns created by SphinxTrain can be used in new ways by Sphinx (Recognizer). For example, there is a large audio corpus called TIDIGITS that consists of transcribed sets of numbers one to ten spoken by many people. SphinxTrain can be used on this corpus to obtain patterns which is used in a simple Sphinx demonstration called HelloDigits. But this training data can also be used in an application, such as one for entering price information into a database.

Installing Sphinx is fairly straightforward. Sphinx comes with several examples that are useful in creating a recognizer for SJQ words. Note that the Sphinx source files are needed to create our recognizer; Sphinx also comes in a non-source version.

However, it took some time to set up and use SphinxTrain. SphinxTrain has some documentation, such as the manual in (Singh). Based on (Googlecode, 2008), we developed a set of instructions to setup and use SphinxTrain on a Windows PC. Aside from the fact that this tries to use a version of the Microsoft C++ compiler that SphinxTrain does not expect, this document helped get SphinxTrain running properly.

In addition to the steps mentioned in (Googlecode, 2008), I also needed to construct a language model and various other files expected by Sphinx and SphinxTrain. These are explained in a Wiki page (Wiki), but no explanation of how to run SphinxTrain is available there.

Sphinx Training

SphinxTrain needs sound files and transcriptions for training. The sound files have to be named in the transcription file. I used the audio recording program Audacity to record sound files. The next section describes the sound file creation in greater detail.

I created fifty sound files. Each file included ten words in SJQ, i.e. one conjunction (*7o1* 'and'), one adjective (*jlyu2* 'big'), one adverb (*jan742* 'then'), and seven nouns. The words I recorded were the following:

k7ya2 'mountain'

sti4 'father'

skwa1 'soup'

7o1 'and'

kcha42 'sun'

sne1 'toad'

jlyu2 'big'

kwen2-tu10 'story'

ko73	'moon'
jan742	'then'

The transcription gives information about the various sound files. The transcription file is one single file; its contents are shown below:

```
<s> K7YA2 </s> (one-a)

<s> STI4 </s> (one-b)

<s> SKWA1 </s> (one-c)

<s> 7O1 </s> (one-d)

<s> KCHA42 </s> (one-e)

<s> SNE1 </s> (one-f)

<s> JLYU2 </s> (one-g)

<s> KWEN2TU10 </s> (one-h)

<s> KO73 </s> (one-i)

<s> JAN742 </s> (one-j)

<s> K7YA2 </s> (two-a)

<s> STI4 </s> (two-b)

<s> SKWA1 </s> (two-c)

<s> 7O1 </s> (two-d)

<s> KCHA42 </s> (two-e)

<s> SNE1 </s> (two-f)

<s> JLYU2 </s> (two-g)

<s> KWEN2TU10 </s> (two-h)

<s> KO73 </s> (two-i)
```

```
<s> JAN742 </s> (two-j)
<s> K7YA2 </s> (tri-a)
<s> STI4 </s> (tri-b)
<s> SKWA1 </s> (tri-c)
<s> 7O1 </s> (tri-d)
<s> KCHA42 </s> (tri-e)
<s> SNE1 </s> (tri-f)
<s> JLYU2 </s> (tri-g)
<s> KWEN2TU10 </s> (tri-h)
<s> KO73 </s> (tri-i)
<s> JAN742 </s> (tri-j)
<s> K7YA2 </s> (for-a)
<s> STI4 </s> (for-b)
<s> SKWA1 </s> (for-c)
<s> 7O1 </s> (for-d)
<s> KCHA42 </s> (for-e)
<s> SNE1 </s> (for-f)
<s> JLYU2 </s> (for-g)
<s> KWEN2TU10 </s> (for-h)
<s> KO73 </s> (for-i)
<s> JAN742 </s> (for-j)
```

(SphinxTrain seems to like the uppercase for the transcription. Incidentally, SphinxTrain is very sensitive to simple variations, such as blank lines in input files.)

In the transcription shown here, KWEN2TU10 is *kwen2-tu10* 'story.' The <s> and </s> indicate that the word has some silence in front of the word and some silence after the word. The words in parenthesis are corresponding file names. For example, <s> KO73 </s> (for-i) means that the word *ko73* is recorded (with the silence before and after the word) in the file for-i.wav.

There are many files needed for training. These are described in the SphinxTrain documentation given in the references. After all the input files are carefully prepared, doing the training involves running one Perl program that calls another set of Perl programs which calls yet another set of Perl programs which trigger various C++ programs. For this set, the training is completed in a few seconds.

Data Preparation

I recorded five sound files using Audacity, an open source voice recording and analysis tool. The recordings were done using a LogiTech microphone attached to a Windows PC. In each of the files, which I named one.wav, two.wav, three.wav, four.wav and five.wav, I recorded the ten distinct SJQ words that appear in the transcription.

I recorded the words with a pause of about one second between words.

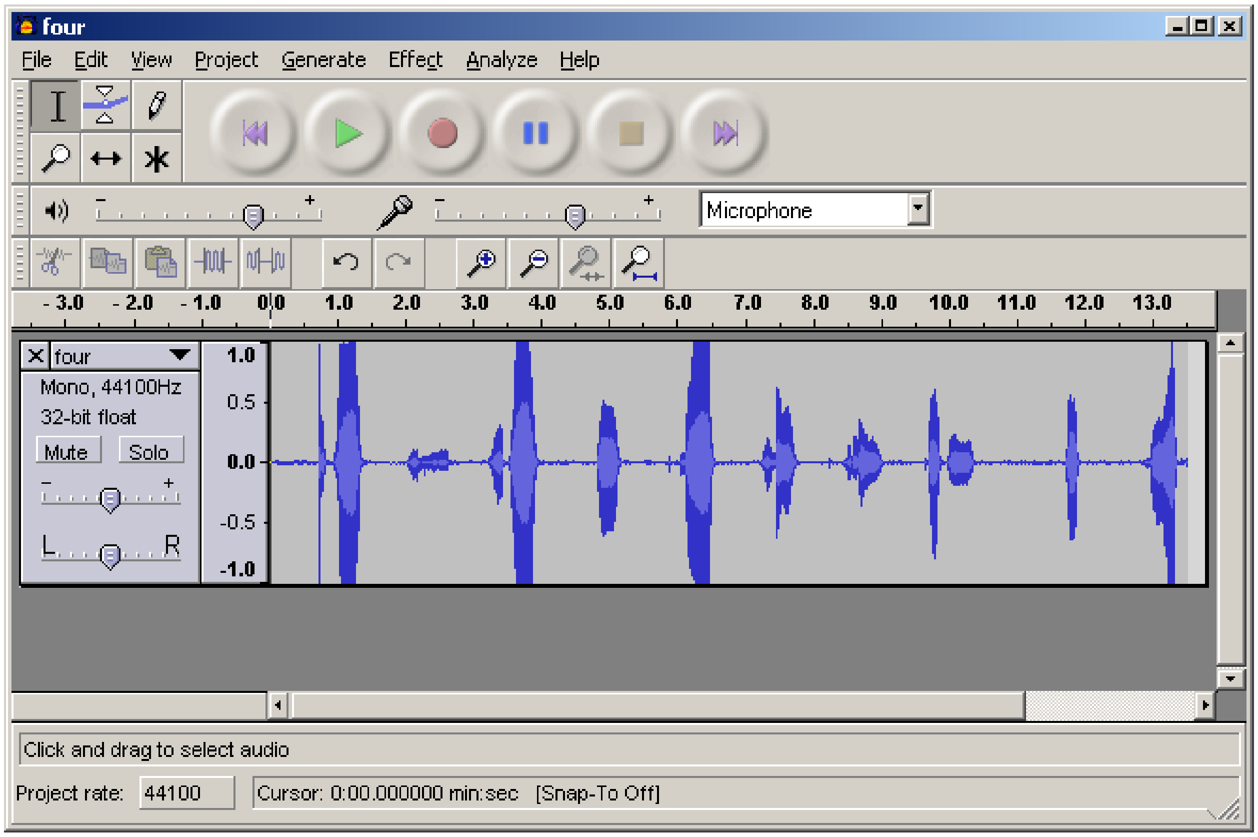

Figure 3. Audacity showing four.wav

The recording produces a file containing many words. I used Praat, the tool for “doing phonetics by computer” (Boersma and Weenink, 2009), to analyze and divide each file into ten individual files. There are some Praat scripts available for doing this automatically; however, some manual operations are needed to create ten files from each larger file. Using a combination of Praat scripts and manual arrangement, I created fifty wav sound files from the original five sound files. These are named as indicated below, i.e.

one.wav became one-a.wav, one-b.wav... one-j.wav

two.wav became two-a.wav, two-b.wav... two-j.wav

three.wav became tri-a.wav, tri-b.wav... tri-j.wav

four.wav became for-a.wav, for-b.wav... for-j.wav

five.wav became fiv-a.wav, fiv-b.wav... fiv-j.wav

The fifty wav files were used to train Sphinx.

Live Testing Program

I developed a program to test SJQ word recognition by copying and modifying a Sphinx demo program called HelloDigits (CMU, 2009). HelloDigits uses the TIDIGITS audio corpus and files trained from that. I do not have access to the original TIDIGITS corpus (which seems to be available only from the Linguistic Data Consortium for a fee). But I was able to get files from SphinxTrain that seemed to be similar to the trained pattern files from TIDIGITS.

The testing program is a Java program. This is packaged into a single “jar” file (“jar” is the format for packaged information for Java programs, similar to the Unix “tar” file for packaging multiple files into a single file). Further, all the trained audio information is packaged into another “jar” file. The Java testing program loads information from this audio jar file. After loading the information, the testing program waits for input from the computer's microphone. It matches the sound heard from the microphone with patterns learned during the training (and stored in the audio jar file).

During testing, I found fairly poor matching with my spoken words. The system recognized k7ya2 correctly almost all the time and recognized sti4 most of the time. However, it produced erratic results on the rest of the words.

Batch Testing Program

I suspected that the poor results from the live testing program were due to differences between my testing speech and the speech I used during training. To verify this, I modified another demo program, wavfile, in the Sphinx system to create a batch, i.e. non-interactive, program to test SphinxTrain.

To do this test, I trained SphinxTrain using only the first forty sound files, i.e. one-a.wav, ..., one-j.wav, two-a.wav, ..., two-j.wav, tri-a.wav, ..., tri-j.wav and for-a.wav, ..., for-j.wav.

After training SphinxTrain, I used the remaining ten sound files, i.e. fiv-a.wav, five-b.wav... fiv-j.wav to test the system.

Here the results were much more encouraging:

fiv-a.wav kcha42
fiv-b.wav sti4
fiv-c.wav skwa1
fiv-d.wav 7o1
fiv-e.wav kcha42
fiv-f.wav sne1
fiv-g.wav jlyu2
fiv-h.wav kwen2tu10
fiv-i.wav ko73
fiv-j.wav jan742

Only the first file was recognized incorrectly in this test.

Discussion of Results

There are several possible explanations for the variation between live testing and batch testing:

1. I was speaking differently when testing.
2. There were artifacts in the sound files I created using Praat.

If the second situation is the explanation, then I would expect to see some difference between the original sound file like four.wav and the smaller files like for-a.wav that I used in training.

This does not appear to be the case, at least looking at the spectrogram of the files in question:

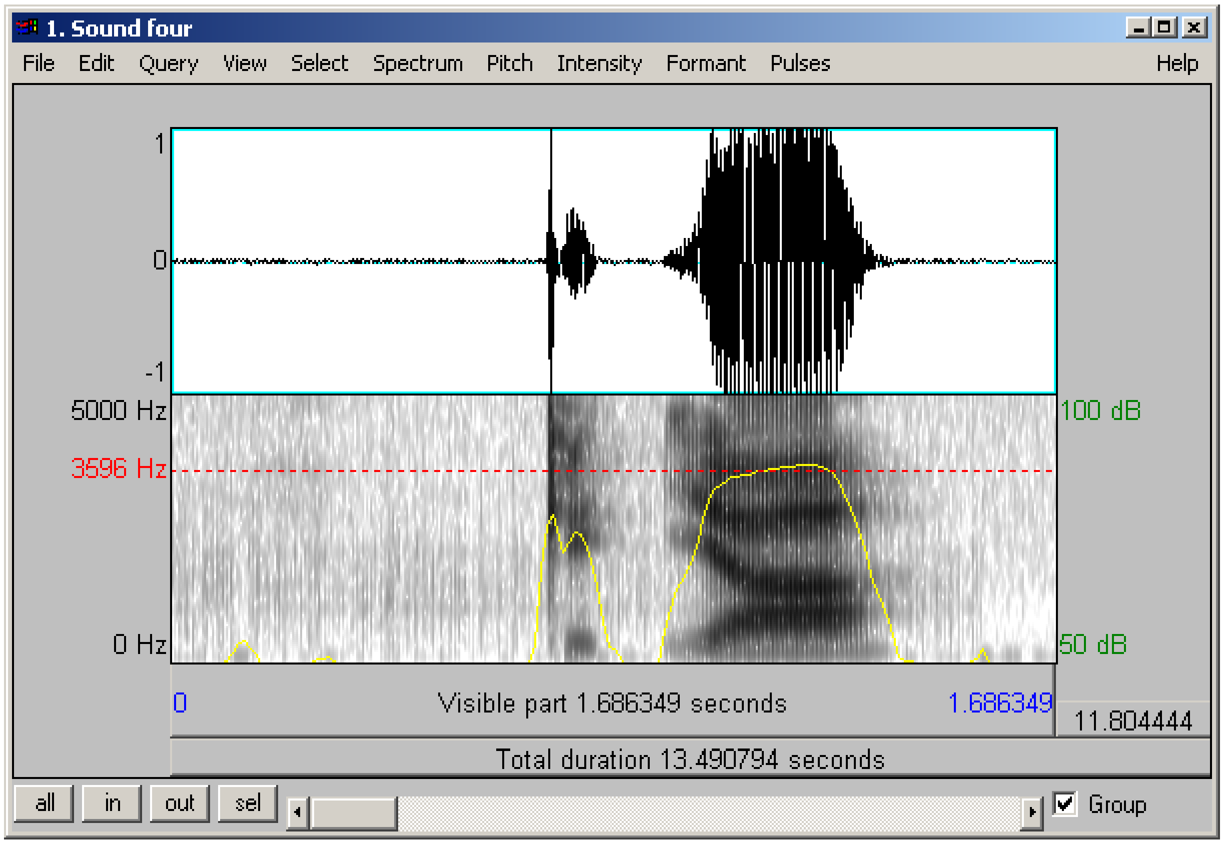
1. Sound four
File Edit Query View Select Spectrum Pitch Intensity Formant Pulses Help
1
0
-1
5000 Hz
3596 Hz
0 Hz
100 dB
50 dB
0
Visible part 1.686349 seconds
1.686349
11.804444
Total duration 13.490794 seconds
all in out sel
Group

Figure 4. Spectrum of start of four.wav. The spectrum above is the word *k7ya2* ‘mountain.’

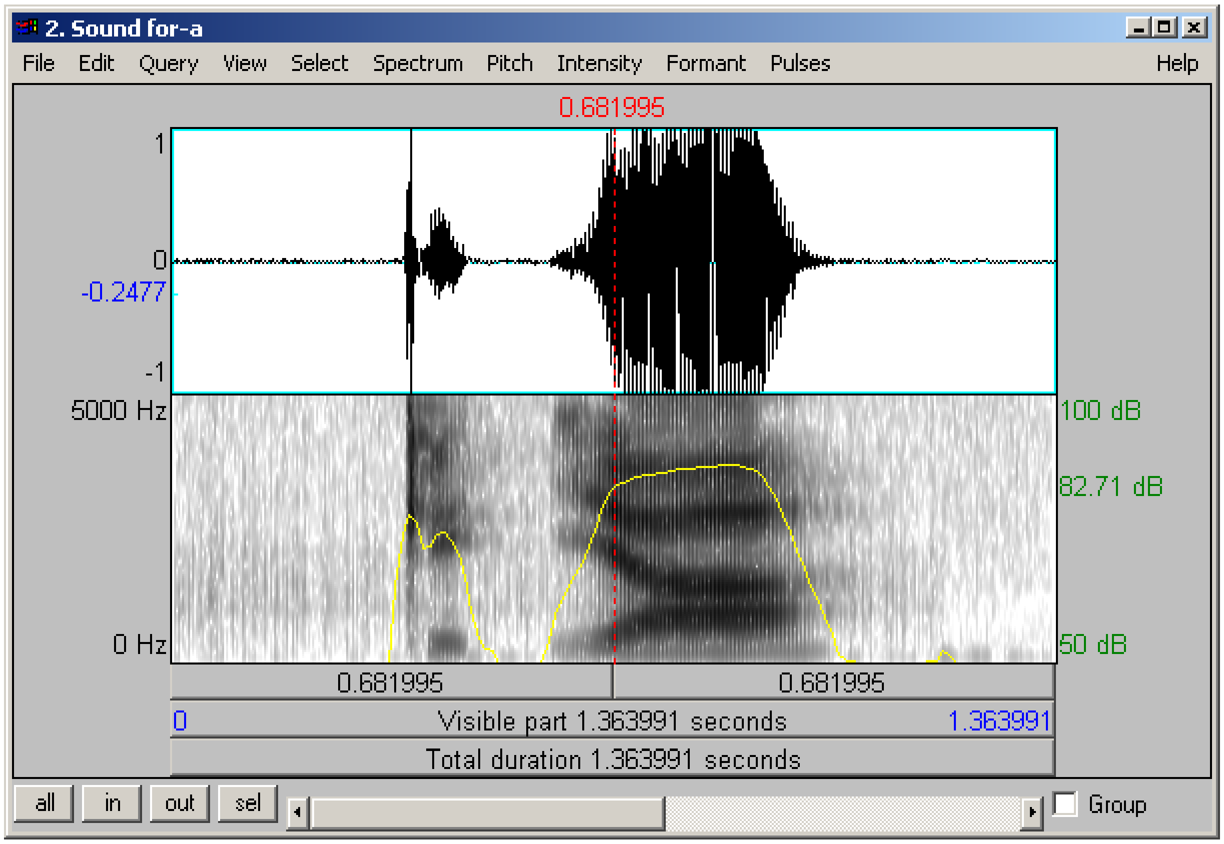

Figure 5. Spectrum of start of for-a.wav (i.e. *k7ya2* 'mountain').

Thus the first explanation seems to be correct. This indicates that I need much more extensive training data to make a proper recognition system that can create patterns that will match more situations and perhaps different speakers; Sphinx is expected to work as a speaker-independent system.

This concludes our discussion of the creation of an isolated word recognizer for SJQ. In the next chapter, we will discuss a continuous speech recognizer. We will often refer back to the isolated word recognizer and discuss the differences between the two types of speech recognizers.

Chapter Five: Continuous Speech Recognizer

A continuous speech recognizer does not need several instances of the same word. Instead, it requires recurring triphones. A *triphone* is any sequence of three phones. In continuous speech, it is more likely that three sounds will occur in sequence than it is that three words will occur in sequence. The fact that a continuous speech recognizer can recognize triphones rather than entire words makes it better suited than an isolated word recognizer to naturally occurring speech, which generally does not include regular pauses between every two words.

As shown in the figure below, a continuous speech recognizer involves the same two stages as an isolated word recognizer: training and testing. Training includes two parts, namely transcribed recordings (recordings and transcripts thereof) and a Language Model. However, since continuous speech recognizers depend on phones rather than words, part of the Language Model must be changed. In particular, it is necessary to go from words in a transcript to individual phones. In other words, there must be some method of *phonetic decomposition*—breaking words up into individual phones. This will be described in greater detail in Chapter Seven.

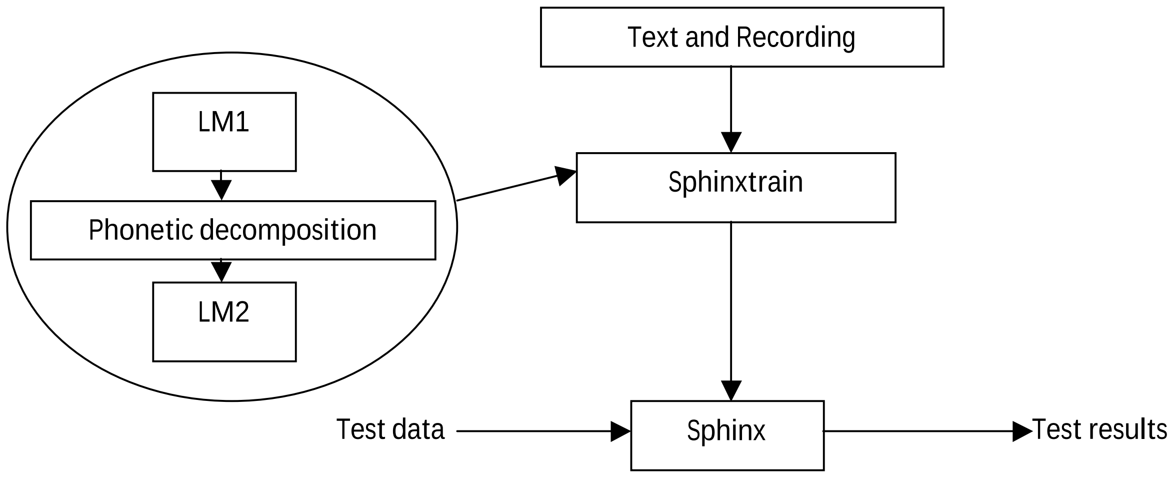

Figure 6. Diagram of basic steps involved in creating a continuous speech recognizer with Sphinx

The main restriction on recordings is that they must be clean and not contain any noise, since Sphinx is very sensitive to noise. Care should be taken to ensure that the transcript does not contain any words that may cause problems for the phonetic decomposition in the Language Model. However, even if not all of the words are correctly decomposed, it is possible to use the recordings and transcription as training data, as we will see later. For the purposes of working with SJQ, as noted in Chapter 3, it may be necessary to modify transcripts based on tone sandhi rules.

It should be noted that Sphinx works best with short sound files, generally no longer than 30 seconds (i.e. roughly one sentence). This means that recordings of entire passages must be broken up into individual sound files, with each sound file mapped to a corresponding line in the transcript. Because recordings must be broken up, it is necessary that there are pauses at which it is possible to split the recording. As we noted before, an isolated word recognizer requires regular pauses; by contrast, the pauses in a continuous speech recognizer do not have to be regular.

Parts of audio files are used as test data entered into Sphinx. The test results are Sphinx's attempted transcription of the test data. In generating its test results, Sphinx uses the input it has received, i.e. any recordings it can use, the transcripts corresponding to those recordings, and its Language Model complete with phonetic decomposition method. The conditions that apply to recordings in the training data apply to test data as well. When there are not many recordings used for training, the test data must be fairly similar to the training data. If Sphinx is not trained on many transcribed recordings but is tested with data completely different from these recordings, then it is likely to generate bad test results.

In our application of Sphinx to SJQ, the transcribed re-recordings of two political speeches were used. The last four lines of each speech were used as test data, and the remainder was used as training data along with the transcript (of the recordings used for training) and Language Model. Additionally, a transcribed recording of 12 (not necessarily distinct) lines from one of the speeches was used as testing data.

The audio data was recorded on a tape recorder in stereo and uploaded onto a computer (not used for processing the recordings). From there, it could be transferred to other computers (used for processing) via online file-sharing. It was then converted to a mono recording in Audacity. Finally, the recordings of continuous speech were split up into individual audio files in Praat, like the audio recording of words in the isolated word recognizer described in the previous chapter.

From this general description of a continuous speech recognizer, it should be clear that the first step is to collect transcribed audio data. Although it is less complicated to collect recordings for a continuous speech recognizer than for an isolated word recognizer, breaking up the recordings for use in a continuous speech recognizer tends to take more time. This is partly because the transcript needs to be organized so that each line of the transcript corresponds to an audio file (i.e. to a specific part of the original recording that can be isolated from the rest of the recording). In the next chapter, we describe the process of recording and assembling the re-recordings of political speeches in SJQ.

Chapter Six: Preparing Data

Sphinx can never function as a speech recognizer of any sort unless it has data to use for training and testing. At least one transcribed recording must be prepared for a speech recognizer to work; test data does not have to be prepared beforehand. The more data Sphinx has available, the more effective it will be in speech recognition. In this chapter, we will discuss the steps used to integrate transcribed recordings of SJQ into Sphinx.

Currently, we have collected data from two political speeches recorded between December 31, 2004 and January 1, 2005 in SJQ. The original speeches were given by two high-ranking politicians in the municipial government of San Juan Quiahije village. One was performed by Ricardo Cruz Cruz (a.k.a. "Mende"), the former mayor of San Juan Quiahije. The other was performed by Remigio Apolonio ("Ligio"), a member of the local elders' council. Both were re-recorded by Hilaria Cruz, a native-speaker of SJQ and graduate student at the University of Texas who had collected and analyzed the original speeches. It is her re-recordings of these speeches that have been used in the creation of a speech recognizer for SJQ.

Collecting the audio recordings itself was a straightforward process. Hilaria read out both speeches from transcripts she herself had prepared, each of which contained a transcription of the original SJQ side by side with its translation in English. We used a tape recorder from which audio recordings could be transferred onto a computer. Each of the two speeches was recorded separately, on different days. Both were recorded in stereo and uploaded onto a computer immediately after recording.

We always uploaded at least one transcript along with the recording. After recording Mende's speech, we uploaded two transcripts along with the recording: One was the transcript from which Hilaria was reading, and the other was an analyzed version of the same text, containing the underlying forms of each word, word-by-word glosses, and free translations in Spanish along with the original Chatino words and free English translation. After recording Ligio's speech, we uploaded only an analyzed version of the text (not the version from which Hilaria was reading directly). These analyzed versions could be used to check the tone sandhi in the transcription. In all of the transcripts, the speech was divided into lines.

It is the processing of the recordings that is a more complicated matter. First, each recording needed to be converted from stereo to mono using Audacity. This was done by splitting the stereo track, deleting the Right track, and changing the remaining Left track to mono. At this point, the mono recording could be opened up in Praat. Before any further work could be done on the recording, however, it would be necessary to create a file containing the transcription as well.

The transcription file could contain only the words that were in the file (i.e. only SJQ words, no English translation). It could not include any punctuation, so all punctuation marks in the SJQ transcription had to be removed. Each line was included by typing in the line number (e.g. "1" for the first line of the transcript), followed by a tab and the SJQ transcription for that line. For example, the first line of Mende in Hilaria's transcripts was "Cha73 no24 t7wi:::24 cha73 tlyu2 ri72 7wan1" ("May you have forgiveness in your heart"). This was entered as follows:

"1 cha73 no24 t7wii24 cha73 tlyu2 ri72 7wan1"

From the recording in Praat, we would try to extract the part of the sound file that corresponded to this line in the recording. We would then save this part of the sound file as a separate .wav file. However, it was not always possible to extract parts of the sound file that corresponded to specific lines in Hilaria's original transcripts. This was because there was not always a pause between lines.

It was sometimes necessary to extract parts of the sound file that corresponded e.g. to two adjacent lines in the original transcripts. It is very important that the final transcript used in the continuous speech recognizer correspond to the audio files, so the number of lines in the final transcript of each speech may not be the same as the number of lines in the original transcript. This was true of both of the texts used for making the speech recognizer. The original 177 lines of Mende were reduced to 136 lines in the new transcript, and the 128 lines of Ligio were reduced to 122 lines.

An additional problem is that the recordings do not always correspond perfectly to the original transcripts. For example, Hilaria would sometimes repeat words when reading from one of her transcripts. This meant that the transcript had to be modified if parts of each newly created audio file was to be effectively mapped with the corresponding transcription. Because of these issues regarding the transcription, creating the audio files and transcript was time-consuming, though relatively uncomplicated.

A more important issue is tone sandhi and the use of lexical vowel length. "Lexical vowel length" simply refers to the fact that long vowels necessarily occur in SJQ only in certain grammatical contexts (specifically, if a first person plural inclusive is involved, e.g. *steen7242* 'our father (inclusive).' In the speeches, vowels that are normally short were sometimes elongated for prosodic reasons having to do with oratory in SJQ. We did not check the transcript of Mende's speech for mistakes involving tone sandhi, and we kept all prosodically long vowels long. (For instance, in the first line from Mende's speech above, we can see that the word written *t7wi:::24* containing prosodic vowel lengthening was rewritten as *t7wii24* rather than *t7wi24*, which is its lexical form). In Ligio's speech, however, we did check for tone sandhi and eliminated prosodic lengthening. This process of checking tone sandhi (and eliminating unnecessarily long vowels) seems to have made the task of creating the sound files and transcription even more time-consuming than otherwise. However, it was a necessary step.

Once a recording has been split into several sound files and the transcript has been reorganized and modified as necessary, the sound files and transcript are ready. The only other part of a continuous speech recognizer that is used as training data is the Language Model, which can be created once and applies equally to all sound files and transcripts. In the next chapter, we will describe the Language Model in more detail, explaining how we modified it for the purpose of creating a continuous speech recognizer for SJQ.

Chapter Seven: Language Model

From the audiodata and transcript, Sphinx generates a Language Model, which includes a group of acoustic models. The Language Model provides information that the recognizer can use to relate the audio files to the transcript. For example, it assigns each phone to a probability value based on how frequently the phone occurs in the transcription. Sphinx also uses the Language Model to generate results when tested with data.

In an isolated word recognizer, the smallest unit is the word, and words cannot be broken down into sounds. This means that the Language Model in an isolated word recognizer is simpler than that of a continuous word recognizer. In a continuous speech recognizer, the Language Model must include a way to divide up a transcript into individual phones, so that patterns of triphones can be found in the transcript.

Carnegie Mellon has developed a tool called “lmtool” as part of its Language Model (CMU 2009). One of the uses of lmtool is phonetic decomposition. It carries out the phonetic decomposition using a phonetic dictionary called cmudict. Unfortunately, this dictionary has been developed specifically for English and cannot be effectively applied to other languages in general. It is time-consuming, though possible, to develop such a dictionary for SJQ. Thus, instead of using cmudict, we have developed an alternative method of phonetic decomposition.

We use a two-stage method for phonetic decomposition involving the use of a table. The method is mainly an application of an algorithm described in (John, 2006) (where it was originally applied to searching in transliterated Mandarin). The table contains sound units (written in the orthography described in Chapter 2) in one column and their equivalents in a phonetic alphabet modeled on Arpabet (which in turn is based on IPA). Any given sound unit may contain one or more phones.

The first stage is breaking down words into sound units. This is done using a right-to-left approach. The second stage is simply converting these sound units into phones. The whole process could be summarized as follows for any given word in SJQ:

1. See whether the word is included in the first column of the table.
2. If not, subtract one character from the *end* of the word, and see whether this new word is included in the first column.
3. Continue step 2 until what remains of the term is found to be in the first column.
4. Next, look for the rest of the term that has been discarded in the list.
5. If necessary, continue step 4, storing terms that are found and performing step 4 with what remains. Stop when the part that remains is an available transcription.
6. Find the phonetic equivalents of the parts of the term that are found. If necessary, divide a part of the term into more than one phone.

Here is an example of how this approach works with an actual word in SJQ. Suppose we want to convert the word *jwe4-sa10* ‘strength’ (which occurs in Mende, in sandhi context) into the phonetic alphabet we use. We would do the following:

1. See whether *jwe4-sa10* is included in the first column of the table.
2. Since it is not, delete the “0” at the end of the word and search instead for *jwe4-sa1*.
3. Continue deleting one letter from the end at a time and searching until what remains of the word is found in the first column. In this case, that means deleting one letter from the end at a time until all that is left is the first two letters *jw*. Store *jw* as a sound unit to be converted into the phonetic alphabet.
4. Look for *e4-sa10*.
5. Since it is not there, delete the “0” again. Then delete the “1,” the “a,” etc. until you are left with only *e*. (In other words, do with *e4-sa10* what was done earlier with the whole word *jwe4-sa10*, then do the same thing with *4-sa10*, etc.). Ultimately, this results in the parts *jw*, *e*, *4*, *s*, *a*, *1*, *0*.
6. These correspond to the phones “hh w,” “e,” “4,” “s,” “a,” “1,” “0” respectively. (Note that *jw* is broken up into two sounds, i.e. “hh” and “w.”) Thus, *jwe4-sa10* has been broken up into individual phones as“hh w e 4 s a 1 0.”

This method has a number of advantages over other existing approaches to phonetic decomposition. It is a finite state machine but avoids order dependence, which is frequently a problem in finite state machine implementations. This gives the method we use an advantage over other finite state machines that are applied to linguistic problems.

The method also has an advantage over the phonetic dictionaries normally used in Sphinx-based speech recognizers. By breaking down words into sound units, the problem of phonetic decomposition is reduced to a problem of mapping a finite number of sounds into phones. This list is considerably smaller than a pronunciation dictionary such as cmudict.

The Language Model, combined with audio data and transcripts, is used as training data for a speech recognizer in Sphinx. In a continuous speech recognizer, the use of a two-stage phonetic decomposition method facilitates the process of breaking down the transcripts into individual sounds in an (ultimately) IPA-based phonetic alphabet. Although there are several texts that should be used as input for the continuous speech recognizer, one Language Model is sufficient for dealing with all such texts. In the next chapter, we will look at the results generated by Sphinx using the combination of our modified Language Model with two transcribed audio files.

Chapter Eight: Recognition Performance

In the last three chapters, we focused on the training data for a continuous speech recognizer. We saw what the three parts of the training data were (audio, transcriptions, and the Language Model), and we also discussed the way in which we created such training data for an SJQ speech recognizer. In this chapter, we will explain how we tested Sphinx's effectiveness and assess the results of our tests.

At first, we used only the audio files corresponding to the last four lines (133-136) of Mende's speech as test data. The other files (#1-132) were used as training data along with the transcript of those lines. We conducted two tests with this data. The first test was used to see whether Sphinx could recognize the first word in a file. The second test was used to see whether Sphinx could recognize all of the words in a file.

During testing, Java (which Sphinx uses) ran out of memory. This was a minor problem that could be solved by obtaining more memory. The testing then ran successfully. If the training data were too small for Sphinx to use at all, Sphinx would not have been able to produce any results. This test shows that Sphinx could at least generate results.

The results of the first test (in which Sphinx was required to recognize only the first word in each test file) are shown below, beginning with the line number, the actual transcription, and then (in the line beginning with "RESULT:") Sphinx's test output:

```
133 KWI724 STEN14 JYAAN1 7NE42 JYA73 SA4 SKA32 CHA73 IN20 7O1
RESULT: kwi724

134 KAN742 CHA73 A14 NO32 TI2 NX7YA20 WA42 7WAN4 IN20
RESULT: kan742

135 SA20 TI2 CHIN720 TI14 CHA73 TI2 TYKWI1 WA42 7O1 WAN24 IN42
RESULT: kwi714

136 7O20 CHA73
RESULT: cha73
```

In each of the first two test files, Sphinx recognized the first word correctly. However, in file #135, it generated as output a word that did not occur in the test file at all. The cause of this mistake is unknown, but it may have had to do with the repeated occurrence of the syllable *ti* in this file (and the fact that more than half of the words end in *i* or *in*), suggesting that the most likely candidate for the first word would be a word that ends in *i*. In file #136, Sphinx generated as output the word *cha73*, which is the second (not the first) word in that file. The reason for this may have been that the word *7o20* contains only a glottal stop followed by a vowel with a tone "20," and thus not much material for Sphinx to recognize. Note that all of the words generated as results in this test are common words in SJQ that occur numerous times in the transcribed audio recordings.

The results of the second test (in which Sphinx's recognition of all the words in each line was tested) are as follows:

133 KWI724 STEN14 JYAAN1 7NE42 JYA73 SA4 SKA32 CHA73 IN20 7O1
RESULT: kwi724 kan742 cha73

134 KAN742 CHA73 A14 NO32 TI2 NX7YA20 WA42 7WAN4 IN20
RESULT: kan742 kan742 cha73

135 SA20 TI2 CHIN720 TI14 CHA73 TI2 TYKWI1 WA42 7O1 WAN24 IN42
RESULT: cha73 kwi714

136 7O20 CHA73
RESULT: cha73 ndya32

In this test, Sphinx recognized no more than three words at a time. In files #133-134, it recognized the first word and the word "cha73" but misheard "kan742" in between. In the other two files, "cha73" was the only word that was correctly recognized. Again, most of these words occurred very often in the transcribed recordings.

Although the testing ran successfully, the results generated were not very good. We then took some modest steps to improve the recognition. One step was to modify the decomposition table; Sphinx had trouble processing punctuation marks that were originally used in the phonetic alphabet (i.e. in the second column of the table). These were replaced with lowercase alphanumeric characters. The other more important step was to use more transcribed audio recordings. It was at this point that we included Ligio's speech along with Mende's as training data.

We also had various lines from Mende's speech reread. This gave us more data with which to test Sphinx. From the recording of lines from Mende's speech, we generated twelve files that were used as test data. (This re-recording of the twelve lines was made separately from the re-recording of the entire speech). We also used eight audio files as test data: in addition to the four audio files from Mende used as test data in the last two tests, we included four audio files corresponding to the last four lines of Ligio. Aside from these eight lines, all other lines from each text were used as training data.

More specifically, in addition to audio files #1-132 of Mende, we used the files corresponding to lines 1-118 of Ligio's speech as training data. Thus, lines 133-136 of Mende and lines 119-122 of Ligio were used as testing data. Additionally, 12 sentences were used as testing data. These 12 sentences all occur in Mende's speech. However, they are not all distinct sentences and do not always correspond to only one line. (Specifically, the 12 sentences are lines 21, 42, 146, 54, 85, 108, 108, 119, 119, 124-5, 135, and 165).

This time, the results were much more promising. These were the results generated for lines 133-136 of Mende:

133 KWI724 STEN14 JYAAN1 7NE42 JYA73 SA4 SKA32 CHA73 IN20 7O1

RESULT: kwi24 sti24 kan720 7ne42 jya73 s sa4-na10 ska32 cha73in20 7o1

134 KAN742 CHA73 A14 NO32 TI2 NX7YA20 WA42 7WAN4 IN20

RESULT: kan720 cha73 no32 ti4 nt7en4 7ya1 wa42 kwan20 in20

135 SA20 TI2 CHIN720 TI14 CHA73 TI2 TYKWI1 WA42 7O1 WAN24 IN42

RESULT: t'kwa1 ska1 ti24 chin32 tii2 cha73 ti4 ri72 tykwi71 wa2 7o1 wan24 in20

136 7O20 CHA73

RESULT: ka2 cha73

In file #133 of Mende, Sphinx had minor problems in recognizing only the first three words and a slightly more serious problem in recognizing the word *sa4-ska32*. In the latter case, Sphinx produced more phones than necessary in its output. (Sphinx also combined *cha73* and *in20* into one word). In file #134, the word *kan742* was recognized as **kan720*, with tone “20” instead of tone “42”; this may be due to differences in pitch range between the test recording of sentences and the recording of Mende. The word *a14* was not recognized; it may have been considered part of the previous word *cha73*. *Ti2 nx7ya20* was misheard as **ti4 nt7en4 7ya1*, and *7wan4* was misheard as **kwan20*. In these last two cases, all of the tones in the test results were incorrect, and some of the phones were mistaken as well (*x* for **t7en4* and, more understandably, *7* for **k*).

In file #135, Sphinx transcribed the first four words *sa20 ti2 chin720 ti14* as *t’kwa1 ska1 ti24 chin32 tii2*. The transcription **t’kwa1* contains an apostrophe; this is an error resulting from a mis-transcription of the word *tkwa24* ‘two’ in the data (note that the tone also is mistaken). For the first word *sa20*, Sphinx returned **t’kwa1 ska1*, inserting an extra word as well as an extra **k*. The main problem with the other words is mistakes in tone, although the glottal stop in *chin720* was omitted and *ti14* was analyzed as having a long vowel. It also transcribed *ti2 tykwi1 wa42* as *ti4 ri72 tykwi71 wa2*, changing the tones on two words and inserting an extra word *ri72* (as well as a glottal stop in *tykwi1*, yielding **tykwi71*). The word *in42* was transcribed as *in20*, once again replacing tone “42” with tone “20”; this may have to do with how much more frequently *in20* occurs in the training data than *in42* does. Finally, in file #136, the word *7o20* was recognized as *ka2*.

The results generated for lines 119-122 of Ligio were as follows:

119 ndyo14-si0 s7wa24 ya72

RESULT: ndyo24-si1 tsan24 ri72 t'kwa1 ska1 ya1

120 ndyo14-si0 s7wa24 tykwa24 7na42 7o1 wan24 ne2

RESULT: ndyo24-si1 sti24 ska24 wa1 7wan1 wa1 yna42 ty7o20 ne2 in24

121 ti2 ti2 ku1-cha73 wan24 ti2 chin720 ke2 re2

RESULT: sti4 ti24 in24 nt7an32 cha1 7wan1 ti24 chin1 in24 7en4 7ne24

122 nde2 no1 ti2 tsa24 7o20 wa42 re2 ne72 lya42 cha73 no24 wa2 lyan242 wan4 ne2

RESULT: nt7en4 no4-nga24in20 ti24 tsa24 7o1 wa1 ne1 7an1 ne1 ya4 cha71 no24 wa1 7wan24 ne2 in24

In file #119, none of the tones were correct, and Sphinx's output includes the words **tsan24 ri72 t'kwa1 ska1* instead of the word *s7wa24*. (Also, the glottal stop in *ya72* was omitted). The word *tsan24* includes the correct tone "24," the words *t'kwa1* and *ska1* each bear some phonetic similarities to *s7wa24*, and *ri72* seems to be an extraneous word. In file #120, once again the tones in *ndyo14-si0* were recognized incorrectly, which is understandable (given the fact that tones "14" and "0" are only slightly higher than "24" and "1," respectively). But with the exception of the first and last words, none of the other words in the line were recognized, and *in24* was added to the end of the result. Perhaps these particular words (*s7wa24 tykwa24 7na42 7o1 wan24*) pose problems for Sphinx's capability of recognition.

In file #121, the words *ti2 ti2* were recognized as **sti4 ti24* (or perhaps **sti4 ti24 in24*); this is simply a matter of omitting an "s," making errors on the tones, and perhaps adding an extra word. Similar problems occurred with the words *wan24 ti2 chin720*. The presence of **(in24?) nt7an32* instead of *ku1-* is more difficult to explain. The last two words *ke2 re2* were recognized as **7en4 7ne24*, with mistakes in tones as well as word-initial glottal stops. This may be partly due to the fact that the words *7en4* and *7ne24* both occur more frequently than *ke2* and *re2*, respectively (in fact, the word *ke2* does not even occur in the training data).

In file #122, most of the tones were recognized incorrectly; the only words that were recognized correctly were *tsa24*, *no24*, and *ne2*. Additionally, the word *re2* in this line was completely ignored. *nde2 no1 ti2* was recognized as **nt7en4 no4-nga24in20 ti24*; *re2 ne72 lya42* was recognized as **ne1 7an1 ne1 ya4*; and the last three words are **7wan24 ne2 in24* instead of *lyan242 wan4 ne2*. None of these results are especially surprising.

From the last set of files, i.e. the test files generated from assorted lines of Mende, Sphinx came up with the following results:

m001 yaan42, 7aan242, in20

RESULT: ngan24 ya1 wan14 ntyka2 in24

m002 kwi724 wan04 ti2 tsa14 te20 lo14, in20

RESULT: kwi724 wan1 ti4 ndya4 t'kwa1 ska1 t'kwa1 wan1 in24

m003   kan742 cha73 no1 wa2 tya24 ra1 nty7on20 ti3 7a1, in20

RESULT: kan742-cha73 cha73 no1 wa1 t'kwa1 sya1 tykwi4 ti24 7a1in20

m004   lo24 wan32 jnya3 7in24 yu24

m005   ndya32 ra0 ja4 ngwa2 7a1 cha73 7ne14 jnya3 7na42

RESULT: lo14 wan32 jnya3 t'kwa1 ti4 yu1

m006   7ne14 cha73 na3 xnyi24 yu24

RESULT: 7ne24 cha73 na2 xnya3 yu4

m007   7ne14 cha73 na3 xnyi24 yu24

RESULT: n7ne1 cha73 na3 x7i2 yu4

m008   xka32 jnya3 wa2 ka14 wan32

RESULT: xka32 t'kwa1 jnya3 wa2 ka24 wan14 in20

m009   ka32 jnya3 wa2 ka14 wan32

RESULT: xka32 t'kwa1 jnya3 wan1 ka2 wan24

m010   ja4 sne1 cha71 jlan14 no1 nga24 7an32 tynya3 re2 o20 cha73

RESULT: ja4-ne1 skan1 ne1 cha71 ngan24 no4-nga24in20 7a24 7an32 jnya3 ne2 kwa32 cha73

m011   ka2 ta20 ndye73 ri72 yu1 t7a42 yu4

RESULT: ka2 cha1 ndya4 yu24 7a1 yu1

m012   nt7en42 cha73 7en4

RESULT: nt7en32 cha1 7en242

In file m001, none of the words were recognized correctly. This may be because the only one of those words that occurs very often in the training data is *in20* (which was recognized correctly except for its tone). It is also worth noting that the words in this file were pronounced at a higher pitch than in the training data.

In file m002, the first word *kwi724* was recognized correctly, and the next two words *wan04 ti2* and the last word *in20* were also recognized correctly except for tone. The other words in the test result for this file are probably more common (i.e. occur more frequently) than the ones in the original file.

The results for file m003 were more successful than those of the two previous files. Apart from *tya24 ra1 nty7on20* being transcribed as **t'kwa1 sya1 tykwi4*, the only errors in this file were mistakes in identifying tones. The words *tya24 ra1 nty7on20* were probably replaced by **t'kwa1 sya1 tykwi4* because they were phonetically similar (e.g. *nty7on20* and *tykwi4* are some of the (relatively few) words in the texts containing the phoneme *ty*) and, again, because the words in the test results may occur more frequently in the training data than the words in the correct transcription.

There appears to have been no distinction between files m004 and m005. Most of the words in m005 were omitted in the result for these two files. Instead, all of the words in m004 (and only m004) were included, except *7in24* which was replaced by the words **t'kwa1 ti4*. All remaining instances of tone 24 in this file were misidentified; *lo24* was identified as **lo14*, and *yu24* as **yu1*.

Files m006 and m007 were repetitions of the same sentence and are supposed to be transcribed identically. However, there were some differences in Sphinx's transcription of these two files. The word *cha73* was transcribed correctly both times, and the word *yu24* was transcribed both times as **yu4*. The word *na3* was also transcribed correctly both times, with the exception of having been assigned tone 2 in file m006. However, *7ne14* was transcribed variously as **7ne24* and **n7ne1*, and *xnyi24* was transcribed as **xnya3* and **x7i2*.

Files m008 and m009 were supposed to be identical. In both files, the first two words *xka32 jnya3* were always recognized correctly (although Sphinx always added **t'kwa1* between the two words), and the last two words *ka14 wan32* were always recognized incorrectly. The word *wa2* was recognized correctly in m008 but not in m009. The last three words were recognized as **wa2 ka24 wan14 in20* in m008 but as **wan1 ka2 wan24* in m009. File m008 is fairly similar to the actual transcription *wa2 ka14 wan32*, although the word **in20* is added at the end; file m009 differs more from the actual transcription, including nasalization (and mistaken tone) in the word *wa2* and a level tone (2) instead of an inflecting tone (14) in the word *ka14*.

In file m010, the "s" at the beginning of the second word *sne1* was not recognized, and two extraneous words **skan1 ne1* were added in. The word *jlan14* was replaced by the similar-sounding *ngan24*, and more extraneous words *–in20 7a24* were added in. (A mistake was also made in the tone of *no1*, and **no4-nga24in20* was written as one word rather than two words *no1 nga24*). The *ty* at the beginning of *tynya3* was mistaken for a **j*, and the *r* at the beginning of *re2* was mistaken for an **n*. Finally, the word *o20* was mistranscribed as *kwa32*.

In file m011, only the first word *ka2* was recognized correctly; the tones on the words *yu1* and *yu4* were different in the transcription generated by Sphinx, but the phones were not. The word *ta20* was mistaken for **cha1*; the word *ndye73* was mistaken for **ndya4*; and the word *t7a42* was mistaken for *7a1. Ironically, the word *ri72* was omitted in the results, though it was included in sound files where it did not appear.

Finally, in m012, only the tones were mistaken (if one ignores the lack of a glottal stop in the transcription of *cha73*). Sphinx transcribed *7en4* as *7en242. This may be due to the fact that only *7en242* occurs in the training data at the end of a sound file; *7en4* occurs in the middle of a sound file in the training data.

Overall, the speech recognizer we have developed for SJQ generates surprisingly accurate results, despite having only 18 minutes of spoken data to work with. It has some problems with recognizing phones and more problems with tones. In some cases, it also inserts a word or fails to produce results for some of the test data. Often, it appears to replace unfamiliar words with words that occur more frequently in the training data. Most likely, it is possible to improve the results; for example, the transcript for Mende may need to be re-edited to account for tone sandhi and to remove prosodic lengthening. Nevertheless, the main resource needed for better results is more texts. Generally, Sphinx works best on training data consisting of about a million phones, which translates to about three thousand sentences. Since Sphinx generalizes from all available data, it is possible to have lower performance when Sphinx is trained on a very large corpus with large speaker variability.

Chapter Nine: Further Research

The recognizer we have created for SJQ has proven to be surprisingly effective. Nevertheless, it is incomplete, and there are several steps that we can take to improve it.

The most important step is to obtain more training data. One problem to consider in the process of obtaining more data is speaker variability in SJQ. There are often substantial differences between the speech of SJQ-speakers. Thus, it may be useful to first work with more data recorded by the same person (in this case, Hilaria Cruz). However, if the speech recognizer described in this report is to be speaker-independent, we will also need recordings of more speakers of SJQ. Many of these may be collected from audio files collected by the Chatino Language Documentation Project at the University of Texas, which have already been transcribed.

Another option for dealing with speaker variability could be creating several recognizers and choosing the recognizer generating the best results in the end. However, the usefulness of such a recognizer may be questionable. Presumably, a speech recognizer combining data from several speakers could be used by more speakers of SJQ than a speech recognizer with good results but using data from only a small group of speakers at most.

To improve Sphinx's recognition (and accuracy in transcription), it will also be necessary to revise transcripts that are used as training data. As previously noted, the transcript for Mende's speech used in this speech has not been corrected for mistakes. We must correct this transcript as well as any other transcripts that will be entered into Sphinx later.

In addition to using Sphinx, we could improve this recognizer by changing some things in Sphinx, especially in the Language Model. One thing we could modify is the tables used for decomposition. For example, the tables generally treat tones as units that cannot be split further (so e.g. "24" is kept as tone "24"), but currently there are some tones that are split into separate tones (e.g. "140" would be split into "14" and "0"). We may want to include such additional tones in the tables, or we may delete inflecting tones and force all inflecting tones to be split into individual level tones (so that e.g. "24" would be split into "2" and "4," and "140" would be split into "1," "4," and "0").

Another reason why we may want to change the tables is to deal with Spanish loanwords more accurately. Spanish loanwords are slightly problematic because they are often written in Spanish orthography (which is somewhat different from the orthography used here for writing Chatino) and because of phonological differences between Spanish and Chatino. One minor complication introduced by Spanish orthography is the presence of the letter *h* in transcriptions. This letter never occurs at the beginning of a Chatino word, but it may occur at the beginning of Spanish words present in Chatino texts (e.g. *hasta* 'until' occurs in line 32 of Ligio). Fortunately, we have dealt with this particular complication, but there are other problems that are still unsolved. For example, our recognizer treats the loan word *presente* as if there were no sound *n* present (because a combination such as *en* in the Chatino orthography indicates a nasalized vowel, not the presence of an actual nasal following the vowel), but it seems to be pronounced with an *n*.

It may be possible to improve Sphinx's recognition of tones. However, this requires an understanding of how Sphinx works internally. We do not yet know how exactly Sphinx recognizes tones, so understanding this and improving the recognition of tones is another area for further investigation.

Thus far, we have been focusing on phonetics and not on other branches of theoretical linguistics. However, one part of the Language Model also involves recognition of syntax. In Sphinx, syntax is modeled using trees. It may be possible to improve the syntactic recognition using newer models of syntax.

References

Boersma, Paul & Weenink, David (2009). Praat: doing phonetics by computer (Version 5.1.05) Computer program. Retrieved May 7, 2009, from http://www.praat.org/

CMU. Carnegie Mellon Speech Group. 2009. Sphinx Knowledge Base Tool http://www.speech.cs.cmu.edu/tools/lmtool.html and Simple LM from http://sourceforge.net/project/showfiles.php?group_id=1904.

Cruz, E; Cruz, H; Cruz, R; Smith Stark, TC. 2007. Complementación en el cha73 jn'a24 (Chatino) de kchin4 K7ya2 (San Juan Quiahije). To appear In Las memorias del Congreso de Idiomas Indígenas de Latinoamérica-II. Archive of the Indigenous Languages of Latin America, http://www.ailla.utexas.org/site/cilla2_toc.html [Original conference presentation October 27, 2007.]

Cruz, Emiliana. 2004. The Phonological Patterns and Orthography of San Juan Quiahije Chatino. University of Texas Masters Thesis. Austin.

Cruz, Emiliana, and Anthony C. Woodbury. 2006. "El sandhi de los tonos en el Chatino de Quiahije," Las memorias del Congreso de Idiomas Indígenas de Latinoamérica-II. Archive of the Indigenous Languages of Latin America. http://www.ailla.utexas.org/site/cilla2/ECruzWoodbury_CILLA2_sandhi.pdf.

Cruz, Hilaria. 2008. University of Texas Qualifying Paper. Austin.

Googlecode. 2008. Simple explanation for using SphinxTrain. http://emg-student-project.googlecode.com/svn-history/r20/trunk/Doc/Training/How%20to%20train.doc.

John, Vijay. A Method for Enhancing Search Using Transliteration of Mandarin Chinese. Texas Linguistics Society 10, 2006 University of Texas. http://uts.cc.utexas.edu/~tls/2006tls/papers/john_tlsx.pdf.

Rasch, Jeffrey. 2002. The basic morpho-syntax of Yaitepec Chatino. Houston: Rice University doctoral dissertation.

Singh, Rita. SphinxTrain manual. http://www2.cs.cmu.edu/~rsingh/Sphinxman/fr4.html.

Wiki page on Sphinx Training, http://Sphinx.subwiki.com/Sphinx/index.php/Hello_World_Decoder_QuickStart_Guide_Pt_2.

Woodbury, Anthony C. 2008. "Guide to data usage in Toolbox SJQProject_1.0." Chatino Language Documentation Project, University of Texas. Austin.