A Phonetic Summarizer for Sociolinguists: Concordancing by Phonetic Criteria

by

Justin V. Sperlein

(Under the direction of William A. Kretzschmar, Jr.)

Abstract

PhoSS, a Phonetic Summarizer for Sociolinguists, is a piece of software that accepts .wav and accompanying .txt transcripts for multiple informants, aligns the files using the Penn Phonetics Lab Forced Aligner, extracts formants with Praat, normalizes formant values using R, and summarizes the information in semi-prose output files with KWIC concordances based on phonetic criteria. PhoSS will compare individuals to a group, directly compare two individuals, or give descriptive output of just the individuals. Two test cases demonstrate PhoSS in action on informants from the Roswell Voices corpus. PhoSS makes large-scale analysis of speech corpora feasible to small research teams and is written using only free and open-source components.

Index words: Computational Linguistics, Forced Alignment, Formant Extraction, Linguistics, Open-Source Software, Phonetic Analysis, Sociolinguistics, Sociophonetics

A Phonetic Summarizer for Sociolinguists: Concordancing by Phonetic Criteria

by

Justin V. Sperlein

B.A., Brigham Young University, 2008

A Thesis Submitted to the Graduate Faculty
of The University of Georgia in Partial Fulfillment
of the
Requirements for the Degree

Master of Arts

Athens, Georgia

2011

A Phonetic Summarizer for Sociolinguists: Concordancing by Phonetic Criteria

by

Justin V. Sperlein

Approved:

Major Professor: William A. Kretzschmar, Jr.

Committee: Michael A. Covington
Lewis Chad Howe

Electronic Version Approved:

Maureen Grasso
Dean of the Graduate School
The University of Georgia
May 2011

Dedication

I dedicate this work first and foremost to my wife, Celeste. She has sacrificed so that I might finish. I also dedicate this to my children, Phoebe and Mark — to Phoebe, who simply knows that daddy goes off to school, and to Mark, who is too young to know even that. With all the love that I possess.

Acknowledgments

I first acknowledge the support of Dr. William Kretzschmar, who has struck the fine balance between insistence and distance. He has been more than helpful whenever I have come knocking on his door, and has otherwise left me to work out my project on my own. I also acknowledge Dr. Mark Liberman for suggesting the forced alignment project that grew into this thesis and introduced me to automated analysis of speech corpora. Thanks to Dr. Michael Covington for teaching me how to program and how to write clearly. I hope that I do his tutelage justice. Thanks to Dr. Chad Howe for being excitedly interested every time I have talked with him, and for always supplying me with sources to investigate. His direction shaped many of the key components of this project. Thanks most of all to my wife, children, and parents, who have done *all* they could to help me.

Table of Contents

Page

Chapter

1 An Introduction to PhoSS 1

2 Setup Information 6

3 Literature Review 16

4 Program Methodology Exhibited in a Comparison of the Vowel Space of Two Roswellian Peers 32

5 Future Work and Conclusions 50

Bibliography 53

Appendix

A Informant Output Files 57

B PhoSS Source Code 87

List of Figures

2.1 Tagging should be correct and consistent throughout. 11

2.2 All text in the transcript will be in the alignment. Pre-processing may be necessary. 13

4.1 Mixing Praat and Python to handle command line options 36

4.2 The .result file before context is inserted 37

4.3 Matching vowel tokens to context after sorting 39

Chapter 1

An Introduction to PhoSS

This thesis is the definition and implementation of a phonetic summarizer with sociolinguistic and language variation studies in mind. The Phonetic Summarizer for Sociolinguistics (PhoSS) analyzes speech corpora using less-structured input than most of the current speech analysis systems do. The design decisions for PhoSS are all based around the ideas of ease-of-use and cost-efficiency. The user need only input a .wav audio file with an accompanying .txt transcript, and PhoSS will align the text and audio, extract formants from a user-defined set of vowels, normalize if the researcher chooses it, and provide the user with both the raw data and data summaries. PhoSS incorporates an array of options to define the vowel measurement point and normalization options used at runtime. If informants have been previously processed, PhoSS recognizes old work and reprocesses only that which is necessary. Perhaps the greatest added value of PhoSS is its concordancing capability. Both phonetic and text concordance lines are provided for every vowel instance in the data. Information representation for phonetic data has traditionally been scatter plots, formant graphs, and spectrograms. While graphics are able to summarize information in a way words are not, words are the heart of all corpus studies, whether textual or phonetic.

Prior to PhoSS, there was no single tool to automate vowel summary, although there were tools that accomplished parts of the task separately from one another. PhoSS is intended in part to fill a need that Hay and Drager sum up in *Sociolinguistics* by stating that, "although variation studies are striving to investigate the social meaning of a variety of phonetic variables, we should hope to move beyond the focus on the variable and move toward studying the semantics of patterns of phonetic realization" (Hay and Drager 2007:98). Along a similar

vein, this concordance approach to phonetic realizations is what William Labov describes in an article about sociophonetics within the framework of sociolinguistics; "The question of interest is how information is stored and in what form it is retrieved. The proposal [in Labov's paper] is that it is stored in the remembered tokens of actual utterances, and extracted from those exemplars" (2006). Labov's statement corresponds with the output of PhoSS. It will show vowel tokens in both their phonetic and textual environments. Judging by the statements of these experts in the field, it seems that sociolinguistics will benefit from a complementary approach to ingesting data that is so often presented in charts and graphs.

There are, closeted away in our collective archives, reels, tapes, and digital recordings that are essentially "all dressed up with no place to go." Some of the reason for this has been the historical cost, in terms of tools, money, and time, of data processing. Until the 1980s the cost of hardware tools was prohibitive. With the rapid spread of desktop computing throughout the 1980s, and especially since the advent of more powerful desktop computers in the 1990s, it is within our grasp to process our stored data on adequate hardware. Software can be a limiting factor. For linguistic research, there is a plethora of analysis tools. Some of them are free; some of those are worth using. Most of these tools spawn from a gap in the cheap or free availability of software suited for linguistics analysis. Thus, over the last decade and a half, the software problem keeps shrinking. Time, however, is a perpetual limiting factor. Automation helps to answer some of the time problem, but introduces a whole new set of concerns and questions. Automation limits human interaction with the data, which feeds the tendency in some programs to be wary of using computer analysis too much. When it comes to answering sociolinguistic questions, automated analyses quickly become complex. Looking for answers often turns into solving multivariate statistical problems. Computers are well suited to run statistical analyses, but no matter how involved computers are in data handling, data interpretation is still a human task. Automation, where appropriate, should have a very specific purpose and specific range. The purpose of automation in PhoSS is description. Its statistical analyses are intended to be preliminary measures which can

confirm or reject initial suspicions without too much investment of time or resources. PhoSS does not attempt to draw conclusions in and of itself.

The goals for this thesis are mainly methodological. My first methodological goal is to use the Roswell Voices corpus as a test bed for the Penn Phonetics Lab Forced Aligner (P2FA). The first published papers that revolve around the performance of P2FA use the SCOTUS corpus — recordings from Supreme Court rulings (Yuan and Liberman 2008; Yuan, Isard and Liberman 2008; Evanini, Isard and Liberman 2009). The SCOTUS corpus straddles a middle ground between word list corpora and interview corpora. It is much more than word list recordings, but it is still a corpus of semi-scripted speech, and so not a true interview corpus. It may be viewed as an ideal interview corpus for speaker clarity. There are a very few speakers, recorded in a controlled environment on good equipment, and there are over fifty years worth of recordings.

Keelan Evanini, in his University of Pennsylvania Ph.D. dissertation, used the P2FA for analyzing the merger of /o/ and /oh/ around Erie, Pennsylvania, based on word list readings and recorded elicitation tasks. I am using the Roswell Voices corpus, which is part of the University of Georgia Linguistic Atlas Project. I chose an interview corpus over a word list corpus because successful application of this process to interview corpora will mean opening the door to automatically processing the material housed in the UGA Linguistic Atlas Project archives. This thesis is the extension of a pilot study that I conducted in the winter of 2010, during which I demonstrated that forced alignment of our Roswell Voices corpus was successful under certain conditions.

The second methodological goal is to join together the existing tools that automate the processes of forced alignment, formant extraction, vowel normalization and data summary, while allowing the user to select some program options at the beginning. By necessity, this becomes an attempt to apply best practices in sociolinguistics and acoustic phonetic analysis. I will have PhoSS default to what I can justify as the best choices based on published research. At the same time, I do not want my program to be so narrowly applicable as to

be useless to others. I will incorporate a robust array of options where is it feasible, which will allow the user to specify alternative methods to vowel formant sampling and application of normalization procedures. These options should make the program useful to researchers who have interests in speech recognition, or to sociolinguists who have very specific ideas regarding sampling and normalization for their data sets. For those who want more control still, the PhoSS code is commented throughout, but especially where a programming linguist can easily change program behavior.

My last goal, and where I will try to add the most value to current practices, is to show phonetic data in context of the original text. The current trend with analytical tools is to produce graphs, plots and tables. While pictures are valuable when summarizing data, they have limitations just like any other medium. "A picture is worth a thousand words," but if given a scatter plot of one thousand words, who can tell where they came from, or what their meaning is? Who can tell what company they keep? It is impossible without a supplement. For this reason, PhoSS will present its data summary in prose and semi-prose, accompanied by key word in context (KWIC) evidence.

Producing this evidence is a five step process. The first step is forced alignment, which assigns duration boundaries to a phonetic representation of a transcript. Second, I extract formant values for a user-defined subset of vowels. An optional third step will normalize the raw frequency measurements in order to reduce variance due to vocal-tract differences if the researcher desires normalization. The PhoSS default is no normalization. Fourth, the data are analyzed, summarized, and supported with KWIC examples. Fifth and finally, the summary is written to an output file.

Chapter 2 will cover all of the software and hardware requirements for implementing this system. It is a good reference for implementing quality forced alignment using only free tools. Chapter 3 will evaluate previous and current work in the areas of forced alignment, acoustic phonetic analysis, with focus on formant sampling practices, normalization procedures, and statistical methods in the context of sociolinguistics and existing corpus methods. Chapter

4 will explain parts of the PhoSS code, and will demonstrate PhoSS in action analyzing two speakers from the Roswell Voices corpus. Chapter 5 will discussion the results of the demonstration along with ways to improve and extend PhoSS, and offer suggestions for future research.

Chapter 2

Setup Information

Modern hardware will be well-equipped to run PhoSS, along with all of its components. I developed PhoSS on a netbook with a 1.66 GHz Atom N280 processor and 1 GB RAM, and on a desktop computer with a 2.4 GHz Intel Core 2 Quad processor and 8 GB RAM. The netbook was sufficient to run the P2FA, which is the most time-intensive portion of the process, although lengthy alignments (I define lengthy as any sound file over ten minutes in duration) take a very long time to process. Alignment of a twenty minute audio file ran for over three and a half hours on the netbook, and for two and a half hours on the desktop, but both ran to completion. Shorter sound files (five minutes or less) align significantly faster. The netbook completes a five minute alignment in twenty minutes, and the desktop in just over ten minutes. This scaling difference is likely due to the increased effort it takes the forced aligner to assign sound models to larger numbers of sound tokens. Variation in the tokens will mean lower probability scores for matching Hidden Markov Models (HMMs) to sound segments, and consequently more work for the aligner to sort out the variation. Evidently, any computer with a 1.6 GHz single-core processor and 1GB RAM will be sufficient to run PhoSS, but the alignment portion should be run on a faster computer when possible. If a faster computer is not accessible, it is a good idea to run the alignment overnight, or as an extended-break task.

All of the software components for PhoSS are free and open-source software (FOSS) available for internet download. PhoSS is designed to run only on Linux and Linux-like systems. I developed and tested PhoSS on Ubuntu 10.10, Maverick Meerkat, and in an Ubuntu 10.10

virtual machine on Windows Vista. PhoSS does use some Linux specific commands, and is not designed to run natively in Windows.

Cygwin, a Linux API for Windows, is also a viable alternative if all of the dependencies compile. I was not able to get RPy2 to install properly under Cygwin, although I have read that it is possible. It occasionally happens that Cygwin packages are not available through a regular Cygwin mirror. When a package is unavailable through Cygwin, it may be listed at http://cygwinports.org. Cygwinports uses a sourceware mirror where additional packages not bundled with the official Cygwin releases reside. There are special instructions to install packages through Cygwinports available on their main page. Working with Cygwin will necessitate downloading and compiling the Linux source code for some dependencies. Linux binaries will not run on Cygwin, but programs that are compiled from source within Cygwin will run just like any other native Linux program. After setting up a suitable environment, there are several software packages that are required to run PhoSS.

The first requirement is an installation of Python 2.6 or newer. I use some os.path commands that are new to Python 2.6, so attempts to use PhoSS with Python 2.5 or earlier will fail. Most of the Python modules that I include are in the basic Python installation. I use the os, sys, subprocess, fileinput, and fnmatch modules for the main functions, and the pprint module for some debug output. All of those modules are included as part of the base Python installation. The one exception is RPy2, which is a module that calls and manipulates R objects directly from a Python script.

PhoSS is written in Python since Python is very flexible, and since it has a shallow learning curve when compared with other computer programming languages. It has a large number of ready-to-use modules, and promotes programming in easy-to-read and easy-to-maintain code. Python also, and very importantly to this particular application, interfaces well with other programs. Through the use of some of its built-in modules, Python can execute and communicate with subprocesses — a capability that I will exploit to interface with both Praat and R. I have found that Python is especially kind for human language

processing tasks. It has a string data type that some other languages lack. The string data type is good for handling raw text, like transcripts. Python lists and classes, on the other hand, are good for defining phonetic sets and organizing informant information. Users who plan to make additions or changes to the PhoSS code will need to have some proficiency with Python.

The HTK toolkit (HTK 2009) is a dependency for the P2FA and, by extension, PhoSS. The HTK toolkit is a speech recognition toolkit that was used for building the Hidden Markov Models for the sounds included with the P2FA. Hidden Markov Models are probability functions for sequences of events, in this case sounds (Manning and Schütze 1999:317). The HMMs included with the P2FA use "GMM-based monophone HMM acoustic models with 32 mixture components on 39 PLP coefficients trained on 25.5 hours of speech from the SCOTUS corpus (Supreme Court oral arguments)" (Evanini 2009:52).

It is possible to use the HTK with existing aligned corpora to train new HMMs. This makes the HTK, and subsequently the P2FA, suitable for investigating languages other than English. The HTK is a free download, but users are required to register in order to download the source code. The version of the HTK that is specified in the P2FA documentation is not available as a direct download from the HTK website. In order to get version 3.4 instead of version 3.4.1, it will be necessary to copy the download link into a web browser's address bar, and delete the .1 in the address. The source will need to be compiled after downloading version 3.4.

The second dependency for P2FA is SoX, Sound Exchange, which is an easy install on Ubuntu, but will need to be compiled from source in Cygwin. SoX is an audio file manipulation tool. P2FA uses it to handle audio resampling, amongst other things. Unless specified otherwise at the command line, all .wav files are resampled to 11025 Hz prior to alignment. 11025 Hz is a low sampling rate. Audio CDs are sampled at 44100 Hz, and high definition DVD audio recordings are 96000 Hz. The Library of Congress stores archived audio from the UGA Linguistic Atlas Project sampled at 96000 Hz. With a low sample rate, there is

a potential for lost phonetic detail, since reducing the sampling rate reduces variation in the sound. Yuan and Liberman justify their design decision by testing seven sampling rates, from 2000 Hz to 44100 Hz. They report the highest alignment accuracy, 98.0%, for audio recordings sampled at 11025 Hz. Incrementally lower sampling negatively impacts alignment accuracy, but stepping up to higher sampling rates shows an even more dramatic negative effect than stepping down to lower sampling rates (Yuan and Liberman 2008). This means that the P2FA is likely to align telephony with higher accuracy than archival quality recordings. This may be due in part to the sound models the P2FA used for testing. Sound models built on high-sampled training data may show better prerformance aligning higher-sampled audio.

The P2FA comes with a distribution of the Carnegie Mellon University (CMU) pronunciation dictionary. The CMU pronunciation dictionary contains headword with phonetic pronunciation entries for over 125,000 words of American English. The pronunciations are transcribed in a variant of the ARPAbet, which is an ASCII representation of the IPA. The P2FA uses the HMMs built with the HTK along with the CMU phonetic pronunciation dictionary to:

1. Generate a phonetic transcription of a plain text transcript through dictionary look-up.

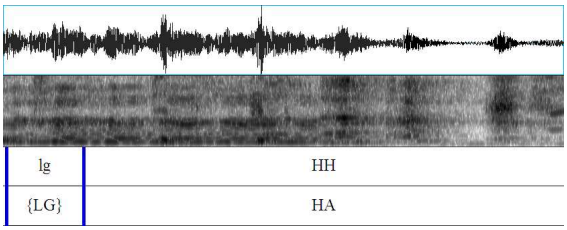
2. Use the resulting phonetic transcript, along with the sound models to probabilistically assign start and end times to the phones in the transcript.

3. Output a two-tier Praat TextGrid with start and end times that correspond to time in the audio file for each phone and word in the phonetic transcript.

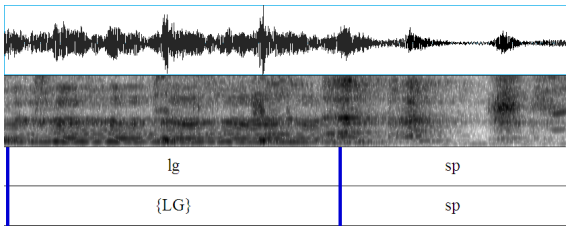
It will likely be necessary to run forced alignment more than once on a new file pair. When the P2FA encounters a word that is not in the dictionary, it gives a `SKIPPING WORD` warning. All of the skipped words should be reviewed and added to the dictionary, along with phonetic pronunciations. If they are not, then they are simply omitted from the phonetic transcript and from the TextGrid, resulting in a functionally useless alignment.

This is a good point at which to give a few words of warning regarding transcription for the purposes of forced alignment. Users should transcribe all words the speaker utters including disfluencies, self repairs, and verbal pauses. Users should tag instances of laughter, coughs, external noises and other non-linguistic sounds. Forced alignment is a probabilistic process. Every millisecond of every sound file will be classified as part of some phone. Whether a section of the audio gets correctly attributed to a phone or separated from that phone depends greatly on the detail of the transcription. Along with HMMs for the complete American English sound set, the P2FA comes with HMMs for laughter, cough, and short pause. The short pause is not of any concern for transcription. It gets automatically inserted at runtime. Why include the short pause, then? Yuan simply states in the P2FA README that, "The acoustic model includes a robust short-pause ('sp') HMM inserted optionally between words which greatly improves alignment accuracy." Laughter, coughs, and all other non-speech noise should be written into the transcriptions with the appropriate tags. Figure 2.1 shows the importance of properly annotating laughter in the transcription. The alignment in figure 2.1a is from a transcription that combines transliterating laughter ("HA HA") and tagging laughter using the P2FA {LG} tag. The result is an alignment that represents one sequence of laughter disjointedly between two segments. The sound model for the {LG} tag accounts for the erratic noise that is part of laughter. Transliteration will try to match the phones HH and AA1 to the sound wave, which it may only tenuously represent. Figure 2.1b is the same alignment using only one convention. The laughter is aligned, and the automatically inserted {sp} tag begins when the laughter ceases to fit the {LG} sound model.

XML encoding or otherwise tagging transcriptions is not discouraged, as long as they are preprocessed and P2FA tags are inserted. If they are not, then all of the non-speech sounds that are present in the audio will become part of some surrounding speech sound. Not taking extra-linguistic sounds into account will cause alignment accuracy to degrade. The P2FA will right itself eventually. That is, even with incorrect alignment through a portion of untranscribed non-speech, the alignment will right itself once more if it encounters another



(a) Misalignment from using transliterated laughter together with the {LG} tag



(b) Correct alignment using the {LG} tag

Figure 2.1: Tagging should be correct and consistent throughout.

stretch of continuous speech. However, for the portion where the P2FA is trying to match non-speech sounds to phones, the accuracy is severely penalized. Users should transcribe all words and parts of words and assign pronunciations to the parts of words in the lexicon. The headwords for phonetic dictionaries are not always recognizable whole words. Users should transcribe disfluencies, self repairs, verbal pauses, and tag all extra-linguistic sounds. This will produce the best alignment results for speech corpora.

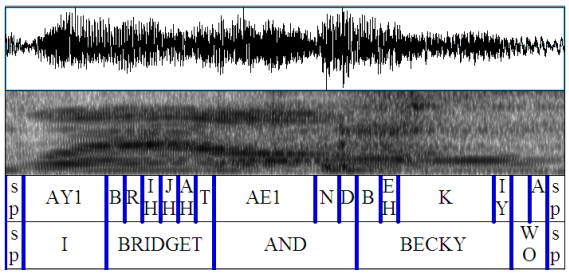
Special symbols, like punctuation, can be included in transcriptions. They are removed during file processing and do not affect performance. Users should, however, take care to transcribe disfluencies and audible pauses as best as possible. It is good practice to make transcription guides or format checkers part of large-scale transcription projects in addition to random quality control checks. Transcription is such an expensive process that transcripts should be designed to have a long life cycle.

The Roswell Voices transcription was carried out long before I began trying to perform forced alignment on the files. This required systematically reworking the transcripts to be consistent with one another and usable with the P2FA. All of our transcripts were typed using Microsoft Word, which uses Microsoft's smart punctuation. The forced aligner does not handle non-ASCII characters, including smart punctuation. I verified that extended ASCII characters are not supported by processing a French text with accented characters. When

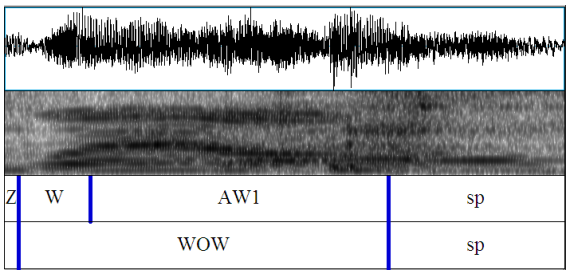
the P2FA processes unrecognizable characters, the skip list will display a list of all of the words with those characters in them. The initial skip lists were full of words like "you□re" and "don□t". Weeding through all of the entries to see which are misspellings and which are actual missing words will take human eyes. In addition to dealing with unrecognizable characters, numerals will appear in the list. It is possible to automatically respell numerals as words, but because of the many ways numerals may be spoken, it is not good practice. The best option is to consult the audio, find out how the informant spoke the numeral, and manually edit the transcript.

It may be the case that even after the dictionary is in order, the transcripts will still need work. Transcription is a purpose-driven task. For the Roswell Voices transcripts, lots of the transcript "problems" were just a result of the transcription being done without alignment in mind. When I looked at the first alignments with Praat, I saw that all of the speaker names were included in the alignment, and were throwing off the segmentation. The aligner does not have a mechanism for intelligently ignoring text or tags, so anything in the transcript files will become part of the alignment. I created a set of processed transcription files for the aligner that existed alongside the original transcription files.

I used our transcription guide to write a script that found and replaced our tags and markers with P2FA standard tags, or to remove them when there was no equivalent tag. What I found, however, was that transcriptionists took liberties with commenting, using whitespace, organizing header information, and marking speaker turns. The transcripts did not all fit into one mold, so I could not fix them in one pass. I wrote a normalization script that tested for smart punctuation and non-standard tags and rewrote them to their ASCII equivalents. Linux command line tools like grep and sed, or a powerful text editor like Vim or Emacs, may be invaluable for bringing order to the system, since they support robust regular expression find-and-replace. I used the results of the regular expression tests in the normalization script to output status messages, alerting me to the types of changes that had been made to each file. Repurposing our transcripts for alignment was, without a doubt, the



(a) An unprocessed transcript may produce erroneous alignment.



(b) The same segment is accurately aligned after processing.

Figure 2.2: All text in the transcript will be in the alignment. Pre-processing may be necessary.

most time consuming portion of the setup phase. Figure 2.2 demonstrates the dangers of using improperly formatted transcripts for alignment. Figure 2.2a is an alignment performed with an unprocessed transcript that had each turn labeled with the speaker name. Figure 2.2b is the same segment after pre-processing the transcript.

The next dependency is Praat, the well-known Phonetic Analysis tool developed by Paul Boersma and David Weenink at the University of Amsterdam. It is a freeware program, written in C, which is a robust phonetic analysis tool and a standard amongst linguists doing phonetics research. Its most prominent features are its LPC spectrum window, and formant and pitch trackers. With a .wav file, and TextGrid (with word and phone tiers), Praat displays a spectrogram with word and phone tiers underneath, making sounds easily isolable for hand-checking alignment results.

It will be necessary to have a knowledge of Praat's built-in scripting language in order to modify the formant extraction routine in PhoSS. Praat's subroutines are tied very closely to its GUI, including the spaces and ellipses in the subroutine names. This can make the Praat scripts hard to decipher and to write. When using Praat at the command line, scripts have to be called using "form" parameters, if there are any. It can be useful to base new Praat scripts on existing examples. The Praat script built by PhoSS is loosely based on Mietta

Lennes' formant extraction script (2003). It outputs a file that is ready to be interpreted by the NORM vowels.R package.

R is an open-source statistical and graphing package that has gained popularity over the last decade. The R project, which is patterned after the S statistical package, has developed into a very powerful terminal-based statistical tool. Harald Baayen gives an accessible introduction to processing linguistic data in R (2008). Jonathan Harrington also uses R and an EMU-R library in conjunction with a suite of phonetic corpora in *Phonetic Analysis of Speech Corpora* (2010). R is becoming a standard tool amongst quantitative linguists since it is much more powerful than the black-box routines available elsewhere. Pairing R with another programming language can produce very powerful tools. R can handle language data directly, but not always elegantly. Python, on the other hand, handles language well and benefits from being paired with a dedicated statistics package.

There is a steep learning curve to R, but in order just to use PhoSS, no proficiency with R is required. Changing the R routines that PhoSS uses will require some experience with R and RPy2. The syntaxes for Python and R are not very far from one another, and using RPy2 should be fairly intuitive for Python programmers. At the very least it should be decipherable. As with Python, there is no shortage of R tutorials and user-made libraries. Most of the R code in PhoSS is part of the vowels.R library.

For Cygwin environments, R is not available through the main Cygwin mirrors, but it is available through Cygwinports. Cygwinports uses the latest version of R currently available, version 2.12.1. Linux users should have no problem obtaining the latest releases of R.

The vowels.R library is a product of the Sociolinguistic Archive and Analysis Project (SLAAP), headquartered at North Carolina State University. Its authors are Tyler Kendall and Eric Thomas. Vowels.R is a well-put-together and well documented library of R functions for vowel normalization. The installation instructions for vowels.R are contained in the header of the file vowels.R. There are five main normalization routines, with options to control their variants. The vowels.R package is the back-end for the NORM project

web form. The web form accepts a formatted spreadsheet or tab-delimited text file of un-normalized formant frequency data, normalizes it, and plots a graph of the results. The results are downloadable as a .zip or .tar.gz file. Users may process their own data and take advantage of the NORM graphing functions by using the NORM web form at http://ncslaap.lib.ncsu.edu/tools/norm/norm1.php.

Chapter 3

Literature Review

PhoSS blends the results of research in text-based corpus linguistics, sociolinguistics, and computational phonetics. Within sociolinguistics, research tends to stress either social factors in language production or the phonetic particularities of speech (Hay and Drager 2007). Stress on the phonetic side has given rise to a sub-branch of sociolinguistics that has been dubbed sociophonetics. PhoSS strives to make quick work of a tedious but necessary step in most sociophonetic research — formant extraction and normalization — with the added value of extracting exemplars in context. This literature review will highlight the relevant research in each of these fields in an effort to clarify the position of PhoSS in a sociolinguistic venue.

Sociolinguistics, as a field, is a relatively young one. It began in earnest in the early 1960s with William Labov's famous "Martha's Vineyard" and "fourth floor" studies (Labov 1963, Labov 1966). From that time, social motivation for language variation has featured more prominently in the education of new linguists. One of the pervading methods for studying the rate of language change is applying the apparent time hypothesis to variation studies. The apparent time hypothesis, as summed up by Hay and Drager, asserts that "[an] individual's phonological systems and accents remain stable throughout their adulthood. For this reason, any observed differences between younger and older speakers recorded at the same time are generally considered to be indicative of changes in progress" (2007:91). This particular theory is interesting from a speech corpus perspective because diachronic speech corpus study can help to illuminate this question. Jonathan Harrington has tested the apparent time hypothesis through a diachronic corpus study of Queen Elizabeth II's yearly Christmas

addresses. Harrington demonstrated that the Queen's vowel space has been shifting away from typical RP norms and in the same direction as change within the general population. The evident vowel change that the Queen has exhibited throughout adulthood runs contrary to the apparent time hypothesis. The implication of the study is that language change in the adult population as a whole is greater than was postulated. This sort of generalization can be dangerous, but further research may prove that this trend holds, and that the apparent time hypothesis is incorrect. This would only mean that language change happens more quickly than the theory currently supposes. Having tools to perform vowel analysis on unstructured speech corpora makes more information such as this easily attainable.

Analyzing speech corpora for sociolinguistic research is a case for applying extended methods used in traditional corpus linguistics. All phonetic corpora, by necessity, have a textual component and fall under the umbrella of corpus linguistics. The textual component is one layer of the corpus, and text corpora have been studied for a much longer time than the audio component. The audio component is a newer investigative dimension for corpus linguistics in which the text becomes a means to an end — the analysis of sound quality that is corollary to the sorts of analyses text-based corpus linguists perform on words. Analysis of phonetic corpora benefits from the history of corpus linguistics, since there are established methods for handling text corpora.

Computer-based corpus linguistics began in the late 1940s with Father Roberto Busa, who created a concordance of the works of Saint Thomas Aquinas. His pilot endeavor proved that using computers for large scale text processing was possible. The 1960s and beyond showed that it was not only possible, but very promising. Francis and Kučera published the Brown corpus in 1963. The Brown corpus was the first "mega corpus," which was composed of just over one million words — five hundred two-thousand word samples in fifteen categories of material published in the United States in 1961. The Brown corpus was the first corpus that was really available for wide study. It became the focus of many early English language

studies and sparked the creation of the LOB corpus, which is a British parallel to the Brown corpus (Kennedy 1998).

The earliest corpus language studies revolved around word and sentence length. It was supposed that word and sentence length could be used as stylometric measures. This idea, however, proved simplistic and soon gave way to more complex studies of stylometry. The most famous of these is the 1964 paper by Mosteller and Wallace, "Inference in an Authorship Problem: A comparative study of discrimination methods applied to the authorship of the disputed Federalist papers." Mosteller and Wallace showed how authors' language varies at the subconscious level, and how that can be shown using proper statistical methods. Even more important than demonstrating the presence of seemingly unconscious function word variation amongst authors, they showed how important it is to apply appropriate statistics to language problems (Oakes 1998:208-212). The next twenty years in corpus linguistics became the time for methods development and solidification.

Through growing corpus study, it became apparent that words attract other words and kinds of words. J. R. Firth noted it best when he said, "You shall know a word by the company it keeps." Researchers like John Sinclair and Michael Stubbs took corpus linguistics through the 1970s and 1980s by studying and measuring collocational attraction, lexical profiles, and register variation. The Collins Cobuild dictionary, first published in 1987, was the first corpus-driven dictionary, built by defining words in terms of typical collocational contexts, rather than in contrived carrier sentences. Meanwhile, Biber was quantifying variation across speech and writing through the use of multivariate statistics (Biber 1988). The British National Corpus also appeared around this time. The BNC is a more modern corpus of 100 million words divided into five broad categories.

By the end of the 1990s Stubbs (1996), Biber (1998), and Kennedy (1998) had all written Corpus Linguistics handbooks which review many of the same methodological approaches to corpus study. Naturally, they cover the same major historical material in corpus linguistics. By the end of fifty years, corpus linguistics had gone from non-existence through its

experimental phase, into an era of established methodologies. All the while, linguists and computer scientists had been developing tools to automate text processing. Today, the focus of corpus linguistics is on size, since most words are very rare objects. The corpora of Mark Davies are the most widely available large corpora. The Corpus of Contemporary American English, patterned after the BNC, is presently at over 400 million words, with about twenty million words of text being added annually (Davies 2008). Garrison Bickerstaff, a recent UGA PhD, composed a six billion word bounded virtual corpus using LexisNexis (2010). Solid methodological foundations in text-based corpus linguistics have led to the ability to process text data on significantly larger scales than researchers not so very long ago.

Phonetic corpus linguistics is currently at a similar state to text corpus linguistics in the 1990s. In exactly the same way that text corpus linguistics has grown, phonetic corpus linguistics is growing. There has been foundational work. There are some tools available, and it is an area of growing interest. We are even starting to see some of the first dedicated handbooks for analyzing speech corpora (Harrington 2010). As it stands, however, there are not very many tools available for the processing of unstructured speech corpora. The current practice is to hop from tool to tool for each stage of the corpus analysis. Alignment is often carried out by hand using Praat, and formant extraction is performed with Praat, WaveSurfer or TF32. Data is often analyzed on spreadsheet tool or in R, and word investigation is done nearly exclusively in WordSmith Tools.

The Akustyk package (Plichta 2011) for Praat comes close to being an all-purpose tool for exploring speech corpora. It is an add-on that modifies the Praat executable, offering automated analysis of specific vowel sets along with graphing capabilities that are much more like what R offers. It offers four normalization methods but does not perform alignment. It can also measure vowel formant trajectories, voicing, intensity, nasalization, and voice quality, which are all features that PhoSS does not offer. Akustyk also offers discriminant analysis, which is good for separating a mass of values into two or three groups based on likeness measures.

There are some stand-alone tools, like the EMU speech database, which is a tool that interfaces with both Praat and R. It uses the EMU query language and is the tool used in Harrington 2010. Harrington demonstrates EMU on Australian and German speech corpora, but user corpora may be formatted for use with the EMU speech database. It incorporates all of the functionality of R and all of the speech-centric functionality of Praat, and looks very useful for users who are willing to spend the time to become proficient in the EMU query language. By far, the most cited corpus-independent tools for analyzing speech corpora are Praat and R, used separately.

While corpus-specific tools are usually the first types of corpus analysis tools to be produced, they are typically not the best tools to use. They invariably lack the ability to answer every question that researchers will ask of a corpus, and cannot always be cajoled into answering questions they were not designed to address. For this reason, PhoSS is designed to be corpus unspecific. All of the file formatting is handled within the program, so that the user has to spend as little time as possible structuring the input.

In general, there are vastly fewer speech corpora than text corpora that are publicly available. This is in part due to privacy agreements, and in part due to the cost of producing speech corpora. The process of recording, transcribing, and segmenting speech corpora is exceedingly labor intensive and very expensive. The Switchboard and TIMIT corpora are perhaps the best known available corpora for studying American English. There are sections of the ANAE (Labov, Ash, and Boberg 2006) that have been annotated, but the files are not available for public download. There are also a number of telephony corpora, but these are geared towards training speech recognition systems, and their audio quality is lower than ideal for phonetic analysis (Yuan and Liberman 2008). The quality of automated phonetic analysis begins with the quality of the alignment.

Forced Alignment is the process of computationally assigning time values on an audio track to the beginning and ending points for the segments of a phonemic-level transcription. Keelan Evanini wrote his 2009 University of Pennsylvania dissertation using the P2FA and

automated vowel analysis to analyze the vowel mergers around the area of Erie, Pennsylvania. I am employing many of the same practices that he uses since they are designed around research using the P2FA, and since they are mostly in accordance with popular practice. I will note deviations from Evanini's methods.

All of the phonetic analysis is dependent upon the accuracy of the forced alignment. Attention to detail in the transcription is the one major user-controllable variable for improving alignment accuracy. Even with the best transcription, the alignment process is prone to some error. For this reason forced alignment will have to perform well in order to be useful as a sociolinguistc tool.

To make sure that the P2FA is worth using, Evanini offers research into the performance of the P2FA. Evanini discusses the performance of the P2FA in comparison with human aligned corpus segments of his dissertation corpus. In that sample of 324 primary stressed vowels, the forced aligner marks the onset for two-thirds of the vowels within a twenty millisecond absolute difference of hand measurements for the same set. Nearly half of the vowels are segmented within 10ms of the hand measured onset times, and the worst performance is a difference of 50ms, which is the case in only one example out of the 324 vowels. The offset times performed a little bit worse than the onset times, but Evanini was not concerned, since measurement points are most often selected closer to the onset time of the vowel, and since vowel gestures are most often separated by short pauses or consonants. That being said, offset times were almost two-thirds within twenty milliseconds and eighty-five percent were within fifty milliseconds.

Yuan and Liberman report similar results, and all parties find the margin of error to be within acceptable tolerance. Evanini does give the caveat that performance will likely drop for interview data. His dissertation corpus is a word-list-style corpus, so his alignments are less likely to encounter laughter or disfluencies as often as in an interview corpus. This is proper warning for use of the Roswell Voices corpus, since it is an interview style corpus. There are elicitation tasks in the corpus, but I am not using them for my examples. There are two

safeguards in place for PhoSS against misalignments. The first is noted by Evanini, namely the law of large numbers. The larger the sample, the less a few erroneous measurements will affect the overall outcome. The other safeguard is the vowel summary using concordance lines. Misalignments that result in outlier measurements will be noted in the output. Through this mechanism, PhoSS alerts the researcher to erroneous measurements.

Utterly critical in regards to this project is determining where to measure vowels. It is a subject, much like English grammar, that has guidelines, fluid rules, passionate preferences, but few hard and fast directions. By far, the most popular practice in both phonetics and sociolinguistics is to measure vowels at their midpoints. The midpoint, apart from being a very convenient place to measure a vowel, is reasoned to be the point that is least affected by the surrounding sounds. In theory, it is far enough from the onglide to be away from perseverative affects and not yet influenced by anticipatory articulation.

Evanini reviews Hildebrand et al. (2005) to find out where humans mark vowels for measurement during hand-annotation. The two judges in Hildebrand mark all vowels at an average of less than one-third total duration. In the Atlas of North American English corpus(Labov, Ash, and Boberg 2006), hand selection was consistently close to one-third of the duration of the vowel (Evanini 2009). The ANAE results are probably a better indicator for measurement point selection, since Hildebrand's results are based on an informant reading a list of isolated hVd words, and lack lots of the transition effects one would expect to see in natural speech. The ANAE runs the whole gamut of vowels, so it accounts for the effects of a wider range of sound transitions on the measurement points.

Evanini, to be very thorough, also compares the performance of several different automated measurement point methods to hand-selected measurement points. He uses ANAE log files, which contain timestamps for F1 and F2 measurements for over 110,000 vowel tokens, aligns the files, and finds where within a phone the measurement was taken. Consistently, the measurement point is around one third the duration of the vowel, further supporting his decision to sample the vowels at the one-third point, and not at the customary midpoint.

Cox 2006 works with a similarly constructed corpus to Evanini's. In her study of aspects of teenagers' pronunciation of Southern Australian English, she manually aligns seventy-two elicitations of stressed vowels in the hVd context for sixty males and sixty females. The vowels are then separated into onglide, target and offglide for monophthongs and onglide, target one, transition, target two and offglide for diphthongs. With this detailed information, Cox analyzes average formant frequencies for males versus females, as well as vowel trajectories from onglide through offglide. The most relevant aspects to my work are her measurements of the relative durations of each of the component parts of the vowel. With the exception of /iː/, the vowel onset is not more than twenty percent of the vowel duration, and the average is around thirteen percent. The end of the vowel target averages sixty-five percent of the vowel duration. That places the center of the vowel target between twenty-six and forty percent of the vowel duration.

Another alternative to measuring at some proportion of the vowel is to measure at the vowel steady state. Sampling at very small increments throughout the sound wave and finding the rate of change from sample point to sample point in the sound wave will reveal a portion of the vowel that has the least amount of variation throughout the vowel duration. This portion of the vowel is the steady state. It would then be customary to measure at the center of the steady state, since this would best represent the point least affected by surrounding sounds. I have not included steady state measures in PhoSS because of the complexity of implementing steady state measurement in the Praat script, and the positive results reported by Cox and Evanini. If a study of vowel measurement points for interview corpora demonstrated that Evanini's findings do not hold, then steady state measurement would be a worthwhile alternative.

The customary midpoint measurement does not appear to be the most accurate approximation of the vowel's target. Indeed, according to Cox, the midpoint for the vowel target of [u] is right at the beginning of the vowel offglide. Cox and Evanini agree that the midpoint

of the vowel target most often ranges from twenty-five to forty percent through the vowel for monophthongs.

When it comes to measuring diphthongs, procedures are less uniform. Evanini decides that there is no significant accuracy gain by measuring diphthongs twice, and dispenses with it. To him, all vowels are best measured at one point. This assumes that a single F1, F2 measurement is the most important characteristic of a vowel, as opposed to the vowel contour, or formant trajectory. While the most important characteristic of the vowel is a matter of theoretical debate, the practice leans heavily toward selecting a single measurement point. Cox 2006 measures single points in both vowel targets for diphthongs. According to her measurements, one-third through a diphthong is, in all but one case, already in the transition between targets. The midpoint of the first target is between fifteen and twenty percent through the vowel, and the second target is very near the end, at about eighty percent of the duration (Cox 2006). Thus, a simplistic measurement of diphthongs may not be the best course of action. I can accommodate multiple sampling for diphthongs in PhoSS by adding a diphthong option. Using Cox's results as a template, and verifying with Yang (1996), I believe it will be safe to sample diphthongs first at fifteen percent and again at eighty percent overall duration.

Similar to Cox, Cho (2008) and Deterding (2000) handle sampling diphthongs differently than sampling monophthongs. While Deterding samples diphthongs at the beginning and end of the segment frames, he is working with a manually aligned corpus, which is less prone to inaccurate time stamps. I will sample diphthongs at fifteen and eighty percent of their durations. Using this method, I hope to approximate the best position for vowel formant measurement for each of the target gestures. Sampling part way into the vowel should help to mitigate any inaccuracy introduced during the alignment, while the ending measurement point I have selected should be a good approximation of the vowel's second target.

When diphthongs are sampled multiple times, they are subject to formant transition measurement. Rate-of-change (ROC) measurements are perhaps the most common kind of

diphthong analytic. The ROC method samples a diphthong twice, once at the beginning of the segment, and again at the end. The ROC is computed by subtracting the end F1 frequency from the beginning F1 frequency and dividing by the duration of the diphthong, yielding a measurement in Hz/sec. Positive ROC indicates a rising formant, while negative ROC indicates a falling formant.

Vocal-tract normalization is the process of removing the effects of physiological differences between speakers from the sound wave, while preserving relevant linguistic variation. The definition of "relevant," however, is heavily biased by the researcher's goals and conceptions of phonological theory. The two main fields of interest for vocal-tract normalization are speech recognition and sociolinguistics. Each lends itself to a different set of normalization procedures. The two main groups of vowel normalization procedures are vowel-intrinsic and vowel-extrinsic procedures. Normalization may improve vowel patterning by filtering the effects of anatomy on sound, but it also means that the sound loses some of its original qualities. Kretzschmar, Dunn, and Kim (2011) demonstrate that unnormalized formant values pattern well amongst varied speakers, supporting the use of unnormalized data. PhoSS makes no assumption about the researcher's intentions regarding normalization procedures, and defaults to no normalization.

Vowel-intrinsic procedures are so-called because they normalize vowel formants based solely on the information contained within one vowel token. There is no reliance on other tokens of the same type of vowel, or on other vowels at all. The unnormalized measurement scales — most commonly Hertz and less frequently Mel, Bark, and ERB scales — are all vowel-intrinsic measurements. The Mel, Bark and ERB scales seek to quantify sound waves based on a perceptual scale rather than as a linear measurement, like Hertz does.

Other vowel-intrinsic methods that are more than measurement scales are the Wakita method (1977) and the Bark-Difference Method, often called the Syrdal and Gopal Bark-Difference Metric (1986). Vowel-intrinsic methods have been demonstrated to be best suited for speech recognition and speech synthesis applications (Adank et al. 2004).

The early normalization studies, Wakita (1977) among them, were interested in normalization for speech recognition purposes. They were often funded by the military, which was most interested in language study for automated speech recognition and translation. By comparing the estimated vocal-tract length with a control case created from a length of tube in his laboratory, Wakita created a normalization formula which mitigates the effects of vocal-tract length differences between speakers. Using available speech recognition methods, synthesized vowels using his normalized formant frequencies were recognizable between eighty-four and ninety percent of the time. Some of incorrect vowel classification, he reports, was the effect of speaker "mispronunciation," by which he means regional accent. In sociolinguistics, this is just the kind of variation that should be preserved after normalization. Of the ten vowel types Wakita sampled, only one showed any distinction between male and female speakers after normalization. Such high recognition accuracy as far back as 1977 (for speech recognition that is very far back) showed that vowel normalization did not destroy the information that makes a vowel unique, and did not completely hide language variation, but at the same time it could mask selected speaker differences.

The Syrdal and Gopal Bark Difference metric, first published in 1986, is a vowel-intrinsic procedure that normalizes using the Bark scale. Hertz measurements are converted into Bark units before normalization. The Bark scale was first proposed by Eberhard Zwicker in 1961, and is a sound scale based on subjective loudness. Just like resonance in a tube or a musical instrument, the basilar membrane has points of maximized and minimized resonance at different frequencies. These sections of the basilar membrane can be divided into critical bands, or ranges, of hearing. The band centers correspond to frequencies that are the most perceptually prominent. As frequencies move to the edge of a band, the sounds are less perceptually distinctive. Zwicker does not expound his experimental methods in his 1961 paper, except to say that "These bands have been directly measured in experiments on the threshold for complex sounds, on masking, on the perception of phase, and most often on the loudness of complex sounds" (Zwicker 1961:248). The Bark scale is measured from 1

to 24, with 24 equating to about 15 kHz. The Syrdal and Gopal Bark Difference Metric normalizes by calculating the differences between Bark values for F1, F2, and F3. While the original Syrdal and Gopal formula uses F0 values in its calculation, the NORM project implements an alternative which calculates the differences between F1, F2, and F3. According to Adank et al. (2004), the Bark Difference metric was one of four methods that reduced the physiological differences for gender to chance level. It was the only vowel-intrinsic method to do so. One of the major advantages to the Syrdal and Gopal method is that its performance is affected neither by the number of vowel types, nor the phonological inventory of the sound system in question. It is, however, "heavily dependent on F3" (NORM: Methods), and is not recommended for speakers who have unusual F3 values due to anatomy or chance (Syrdal and Gopal 1986).

The other group of normalization procedures is vowel-extrinsic. Normalization using vowel-extrinsic methods requires aligned corpora, since it uses aggregated information on all vowel tokens from one or more speakers in order to calculate normalized formant values. The side-effect of these methods is that variation within a type is taken into account while analyzing a token of that type, and more broad information about variation is preserved. Adank et al. (2004) report that vowel-extrinsic methods perform best at reducing vocal-tract influence, while preserving sociolinguistic variation. The main drawback to vowel-extrinsic methods is that they are easily skewed based on the vowel subset being investigated. PhoSS includes two of the vowel-extrinsic procedures tested by Adank et al., Lobanov 1971 and Nearey 1977, as well as two more vowel-extrinsic methods: Watt and Fabricius 2002 and Labov 2006.

Lobanov is the oldest of the vowel-extrinsic methods, but it performs consistently amongst the best in Adank et al. (2004) and Disner (1980) for reducing gender differences, preserving phonological distinction, and preserving sociolinguistic variation. Disner reports that Lobanov performed worse than Nearey (1977 method) at preserving subtle distinctions between vowels from different languages, like the English /u/ and the Japanese /ɯ/.

The Lobanov method does have some drawbacks. Apart from being poor for comparing fine-grained cross-language distinctions, the performance of Lobanov is dependent upon the range of vowels in the data set. Calculating the normalized values takes all types and tokens into account for any given formant. The formula for Lobanov as provided by NORM is:

$$F^N_{n[V]} = (F_{n[V]} - mean_n)/S_n \tag{3.1}$$

"Where $F^N_{n[V]}$ is the normalized value for $F_{n[V]}$ (i.e., for formant n of vowel V). $mean_n$ is the mean value for formant n for the speaker in question and S_n is the standard deviation for the speaker's formant n" (NORM: methods). The normalized values are not in the Hertz scale, so they are not readily comparable with raw Hertz values. The drawbacks of the Lobanov method are common to all vowel-extrinsic methods, namely that its results are heavily influenced by the vowels included in the study. Despite this common drawback, the Lobanov method consistently performed amongst the top two normalization methods tested by Adank et al.

The other high-ranking vowel-extrinsic procedure, which performed right alongside Lobanov in both Disner and Adank et al., is the Nearey 1977 method, which Adank et al. labels Nearey1. The greatest difference between the Lobanov and Nearey methods is that Nearey uses log-mean values where Lobanov does not. Nearey published a revised method, which Adank calls Nearey2, that uses a scale factor across all formants for all vowels, instead of scaling each formant for each vowel separately as in Nearey1. Likely because of this, Nearey2 performs summarily worse than the original. For this reason it is not offered in PhoSS. The implementation of Nearey1 takes the form:

$$F^N_{n[V]} = antilog\left[log(F_{n[V]}) - mean(log(F_n))\right] \tag{3.2}$$

"Where $F^N_{n[V]}$ is the normalized value for $F_{n[V]}$, formant n of vowel V, and $mean(log(F_n))$ is the log-mean of all $F_n s$ for the speaker in question" (NORM: methods). In some cases Nearey performs better than Lobanov, but they are so close as to be nearly equal.

Watt and Fabricius (2002) developed their method because of the insufficiency of the Bark-transformation scale when comparing formant values in Bark units. The Bark conversion compresses high Hertz ranges, which are typical of women's and children's speech, into lower ranges so that they should be perceptually more similar to men's formant values. Say Watt and Fabricius, "We make no criticism of the use of Bark-transformed data, nor the validity of the scale itself, except to say that it does not in fact fully permit direct comparison of one speaker's vowel sample with another speaker's vowel sample in the way we would wish" (Watt and Fabricius 2002: 161). Their intuition was correct, according to Adank et al. The Bark-transform performed at 94% recognizability on a gender identification test, which indicates nearly absolute failure of the Bark scale to mask gender in perceptual tests. Their answer works by measuring Bark-converted vowels in relation to an S-transform, or centroid mean, for all vowels. The S-transform is calculated using the high front, high back, and lowest vowel in the set as the corners of a feature triangle. Watt and Fabricius originally developed this method for analyzing British English, which has one clear lowest vowel. American English, on the other hand, quickly reveals a shortcoming of Watt and Fabricius' method that can skew normalized results. The American English vowel system has two low vowels whose F2 values are roughly parallel to one another. The lowest value in a case like that can skew the results towards fronting, in the case of a low F1 value at $F2_{min}$, or backing if the F1 value is high at $F2_{min}$. The implementation of Watt and Fabricius in vowels.R accounts for these kinds of sound sets. The top front corner is set to $F1_{min}$, whether or not it is F1 of [i]. The bottom corner of the triangle becomes $F1_{max}$. The top back corner is $F1_{max}$,$F2_{max}$, or [u] in English. Using English vowels as examples, the S-transforms, or centroid values, for F1 and F2 are calculated using the following two formulae:

$$S(F1) = ([\text{i}]_{F1} + [\text{æ}]_{F1} + [\text{u}]_{F1})/3 \tag{3.3}$$

$$S(F2) = ([\text{i}]_{F2} + [\text{æ}]_{F2} + [\text{u}]_{F2})/3 \tag{3.4}$$

Normalized values are then calculated by dividing a vowel's mean F1 and F2 by the appropriate S-transform.

The Labov method is a variation on Nearey's method, which was used on the ANAE. It normalizes using log-mean values, but it has a means of incorporating scaling into the normalization procedure. A log-mean value for all speakers in the study serves as a center point to make a scaling constant. The formant values are then multiplied by the scaling factor, giving a Hertz-like value (note that Nearey values are not in Hertz-like values). The drawback to this method is that the log-mean center shifts with the addition of new speakers. Labov et al. (2006) report that the center does not change significantly after 345 informants. The scaling constant calculated from the ANAE is the default scaling constant for NORM and, by extension, PhoSS. It can be overridden by supplying another scaling constant as a named variable in the Labov section of the normalization module.

Each of the vowel normalization methods, whether vowel-intrinsic or extrinsic, have their advantages and disadvantages. The options included in PhoSS are what previous research shows are the most reliable options. Where normalization is appropriate, the normalization method should be chosen to fit the purpose of the study.

PhoSS outputs a .csv of normalized values (or raw values in the case of no normalization) which is compatible with spreadsheet programs and offers individual or comparative summaries where appropriate. Both summaries are centered on descriptive statistics, in particular on vowel space comparison. The individual summary looks for unusual tokens within a speaker. This may be useful for examining situational language differences, such as accent shift between home and work, or home and educational environments. The individual summary describes the overall speaker vowel space, and then gives a vowel-by-vowel report with similar information. The vowel summaries include concordance examples of exemplars that exhibit certain phonetic characteristics. For example, all vowels that fall within X standard deviations of the vowel mean are grouped together. X increases by one-half standard deviation increments until there are no more exemplars. This should lay bare any trends in the context for exemplars that are grouped together by phonetic quality.

The comparative result measures an individual against a population. That population may be as small as one other individual, or it may be a large group. The comparative summary uses the same statistics as the individual summary, but removes the informant from the population and compares him to group averages, instead of to his own averages. I focus on formant means and standard deviations since they are capable of giving insight into the vowel spread, the presence of outliers, etc. Simple statistics that are informative and pertinent to most research questions are the most important statistics for this initial effort. In all of the design decisions for PhoSS, I have tried to pattern my choices after current, respected, published research.

Chapter 4

Program Methodology Exhibited in a Comparison of the Vowel Space of Two Roswellian Peers

There are seven modules that make up the PhoSS source code. There is a main module, a TextGrid parser, and one module for each of the five main steps required to complete sound summary: alignment, formant extraction, normalization, statistical summary and output. I will explain the program methodology through an example case that compares the vowel space and corner vowels of two Roswellian peers.

I prepared the transcripts and .wav files for this example by making sure that all numbers were properly transcribed, laughter tags were inserted, speaker names were removed, and that verbal static was transcribed. I have partially processed the files for the second informant in order to show how PhoSS behaves when it encounters repeated material. The audio segments for this example are short — approximately two minutes each, and the interviewers have minimal input. A short example is sufficient, however, to demonstrate the program. I am looking at vowel space by comparing the four primary stressed corner vowels for American English, which are represented in the IPA as [i], [æ], [u], and [a], and by the CMU pronunciation dictionary as IY1, AE1, UW1, and AA1 respectively. I will process the data both with no normalization and using the Lobanov normalization procedure.

The PhoSS main module controls the program execution. It calls upon other modules to populate the main data structure, an instance of the `Speakers` class called `InformantList`. PhoSS uses the Python `OptionParser` module to parse command line options and arguments. The result of parsing the command line is a pair of variables, `opts` and `args` which contain the program options and arguments respectively. The options set methods for determining

the vowel measurement point and the normalization methods to use on the formant data. PhoSS supports selecting multiple normalization methods in one command. In the case of multiple normalizations, PhoSS writes an output file with the normalization method in the file extension. PhoSS also supports specifying a separate set of diphthongs, which are processed differently than the vowel set. For example, the option `-p "AY1 AW1"` will add AY1 and AW1 to the diphthongs list. Formants are sampled twice for diphthongs, and all of the summaries that are performed for monophthongs are performed for each of the diphthong targets.

PhoSS arguments may be any number of directories, ended by a quoted string of comma-separated vowels. For this two-informant case where I am performing a direct comparison of two speakers, examining the corner vowels with both Lobanov normalization and no normalization, the command to run PhoSS is:

```
python phoss.py -rlg ./test_one/informant_one/ ./test_one/informant_two/
"IY1, AE1, UW1, AA1"
```

PhoSS interprets every directory argument as a new speaker. Each speaker becomes an instance of the `Informant` class, which is added to the `InformantList.speakers` list. The organization is `InformantList.speakers[Informant,Informant]`, where each `Informant` contains all of the information relevant to one actual human speaker. This organization makes it easy to perform one operation at a time for all informants by iterating over the `InformantList.speakers` list.

The last argument, the vowel list, is checked for accurate syntax and interpreted as a Python list. If the user forgets to include a vowel list, PhoSS prints a reminder message, `Use a comma-delimited vowel set as last argument! Example: "AE1 AH0"`, and exits the program.f Since the example command is well formed, the program proceeds.

The minimum contents of an informant directory are a .wav file and a .txt transcript. A .TextGrid is optional; however, its absence prompts the user to perform alignment using p2fa.

```
No TextGrid found in  ../testfiles/test_one/informant_one/
Run alignment on selected audio and transcript? (y/n) y
```

In order to run alignment PhoSS must first find the p2fa alignment script. The p2fa alignment script does not have a predetermined installation directory. PhoSS checks customary Linux installation locations for p2fa/align.py. When PhoSS locates p2fa, it keeps the location in memory in case it has to run multiple alignments. PhoSS uses the .wav file name as the basis for all other file names created throughout the program (including the .TextGrid when there is not already one). Once PhoSS has found the alignment script it displays the p2fa calling command so that the user may visually verify that all of the directories are correct. PhoSS then runs the p2fa as a subprocess and waits to continue until the subprocess has completed.

```
Searching for p2fa/align.py...
Found p2fa/align.py...
Calling p2fa with path:  python /usr/local/p2fa/align.py
/home/user/testfiles/test_one/informant_one/informant_one.wav
/home/user/testfiles/test_one/informant_one/informant_one.txt
/home/user/testfiles/test_one/informant_one/informant_one.TextGrid}
```

The initial run of the alignment produces a string of p2fa warning messages:

```
SKIPPING WORD CRABAPPLE
SKIPPING WORD DADDIE'S
SKIPPING WORD EBENEZER
SKIPPING WORD BETRACTED
```

These particular warning messages indicate words that are not present in the aligner's lexicon. The user has to enter the words manually into the CMU dictionary, found at `p2fa/model/dict`. The `SKIPPING WORD` messages are often interesting objects for study in

their own right. The words that are likely to be in the skipped word list are either not real words (typos, etc.), or are uncommon enough to be left out of large dictionaries. For interview corpora, these are often place names or colloquialisms, as shown in this example. The communities of Crabapple and Ebenezer are towns in Georgia, like Roswell. `DADDIE'S` is a misspelling of `DADDY'S`. `BETRACTED` is a colloquialism, akin to a revival meeting. An internet search for "betracted meeting" produced only one correct match in a book published by the University of Georgia Press, whose author was born in Tennessee in 1893. This small investigation together with knowing the informant's age make it reasonably clear that "betracted" is a rare term that enjoyed some vogue in the South around the turn of the twentieth century and meant "revival meeting." The `SKIPPING WORD` list will not be useful for finding unique meanings of common words, but it is a good source for studying unique lexis.

After adding the missing words to the dictionary and restarting PhoSS, it shows the p2fa update messages, but no p2fa warning messages. PhoSS alerts the user when the alignments completes.

```
Resampling wav file from 48000 to 11025...
sox WARN sox: effect `polyphase' is deprecated; see sox(1) for an alternative
./tmp/sound.wav -> ./tmp/tmp.plp
Alignment complete for informant_one
Alignment previously completed for informant_two
```

Following the completed alignments, PhoSS writes a Praat script for each informant. There is not a way to interface with Praat directly from Python, as the RPy2 module does for R. Instead, I used a combination of Python loops, substitution strings and hard coded Praat script lines to write a Praat script to a file. This allowed me to circumvent some of the idiosyncrasies of the Praat scripting language. One of the benefits of combining Python and Praat was the ability to use command line options to specify formant extraction points. The default measurement point is one-third the duration of the vowel. There is a midpoint selection option, since that is another common practice, as well as a diphthong option that samples twice — once at fifteen percent and once at eighty percent of the vowel

```
for n in xrange(len(vowelset)):
    if options.midpoints == True:
        sampletype = "midpoint"
        formula = "(beg + end) / 2"
        if n == 0:
            praat_script.append("if lab$ = `%s'\n" % vowelset[n])
        else:
            praat_script.append("elif lab$ = `%s'\n" % vowelset[n])

        praat_script.append("\tbeg = Get starting point... 1 `j'\n")
        praat_script.append("\tend = Get end point... 1 `j'\n")
        praat_script.append("\t%s = %s\n" % (sampletype,formula))
        praat_script.append("\tselect Formant `name$'\n'''')
        praat_script.append("\tf1 = Get value at time... 1 `%s' \
Hertz Linear\n" % sampletype)
        praat_script.append("\tf2 = Get value at time... 2 `%s' \
Hertz Linear\n" % sampletype)
        praat_script.append("\tf3 = Get value at time... 3 `%s' \
Hertz Linear\n" % sampletype)
```

Figure 4.1: Mixing Praat and Python enables writing Praat routines by selecting command line options.

duration. The measurement point options call different Python loops that write the Praat vowel measurement point routine. With this combination I could incorporate options without trying to force Praat to interpret them. Figure 4.1 is an example of one of those Python loops. Python checks for the option that specifies midpoint selection and writes an if-loop for each of the vowels in the vowel set. I use Python to perform string substitution where the Praat script requires it. The tab characters are not necessary, but they are included to logically indent the Praat script. After PhoSS writes the Praat script, it calls Praat as a subprocess and runs the script, extracting formants at the locations the user specifies for the vowels in the vowel set. Writing the Praat script is a very quick process, so PhoSS overwrites existing .praat files and prints the message, `Wrote Praat script for informant_one`. Formant

extraction takes several seconds, so it may not be desirable to overwrite the .result file every time. When no result file exists, PhoSS automatically runs formant extraction.

```
Executing Praat script for informant_one because result file does not exist.
A result file exists for informant_two
Execute Praat script to overwrite existing? (y/n) y
Executing Praat script for informant_two because you said yes.
```

Running the Praat script yields a .result file. The .result is properly formatted input for vowels.R and the NORM web form; however, the context column is empty. Empty columns are permitted in the context and glide columns of the .result file as long as there are tabs for all empty fields. The only side effect is that all empty fields are replaced with 'NA' when the file is read into an R dataframe. There was no comfortable way to reconstruct context within the Praat script, so I decided to reconstruct it as a separate step before normalization. When there are multiple informants, PhoSS sorts and merges the .result files and writes a .commonresult file. The vowel types must be sorted by speaker and by vowel to be usable with vowels.R. The .commonresult is necessary for speaker-extrinsic normalization procedures whose normalized values factor in all speakers in the data set. Setting the `--no-compare` option prevents the .commonresult file from being written. When a .result file already exists, PhoSS alerts the user and asks whether to overwrite it. The user may desire to overwrite existing .*result files after hand-correcting the TextGrid in the case of erroneous formant measurement or inaccurate time stamps. Figure 4.2 is an excerpt from the .result file for informant two which shows the data organization before context is added.

```
speaker vowel/frame context F1 F2 F3 gl F1 gl F2 gl F3
druck_test AA1 374.8208611634132 2212.8631765128216 3391.9695808521774
druck_test AA1 511.77337024711585 1004.3139704774767 2859.977220223504
druck_test AA1 548.5045116991059 834.0718681201741 2472.3925796903054
```

Figure 4.2: The .result file after formant extraction and before context is inserted

While there are empty fields for the context at this point, they will not remain empty. Context in all of the NORM website example files is represented as a carrier word like CAR

for all instances of AA1, or YOU for all instances of UW1. PhoSS takes for granted the fact that phonetic context influences the realization of the phone, and is designed to investigate whether textual context also influences phonetic realization. Having one arbitrarily selected carrier word is insufficient to represent context as I wish to show it. There may be trends to the phonetic and textual context that can be revealed by concordancing based on phonetic criteria.

The TextGrid parser reconstructs the textual and phonetic context for each vowel token for each informant. The TextGrid parser works by populating two lists, the `phone_list` and the `word_list` with instances of the `pList` and `wList` classes. The TextGrid parser reads through the TextGrid in chunks and extracts the phones and words in order. Each phone becomes an instance of a `pList` in the `phone_list` (`phone_list[pList,pList,...]`). Each `pList` instance contains the beginning and ending times for the phone, its index in the `phone_list` and a context window with two phones to either side of it. The TextGrid parser gathers similar information about each of the words in the transcript — beginning and ending times, index in the `word_list` and the range of indices of phones from the `phone_list` that make up the word.

In order to associate the correct context with the correct vowel token, PhoSS needs to index the .result file sorting that took place earlier. PhoSS uses the `phone_list` for constructing a similar but simplified list to the one used for sorting the .result file. After sorting that list, PhoSS matches the ending time stamps of its elements to elements in the `phone_list`. Matches return the `phone_list` element index. PhoSS uses the post-sort element indices together with the `phone_list.index` to correctly dereference context from the `phone_list`. Figure 4.3 shows part of a list with the sorted vowel token, the ending time stamp, the `phone_list` index and the post-sort index. For the first element in that list, whose index is 0, the vowel context for that AA1 is the `phone_list.context` of `phone_list` element 14.

```
[('AA1', '1.18979591837', 14, 0),
 ('AA1', '107.009297052', 981, 1),
 ('AA1', '110.212018141', 1026, 2)...]
```

Figure 4.3: Vowel tokens are matched with context by `phone_list` and post-sort indices

PhoSS matches words to phones by using the `word_list.phonerange`. If a vowel has a `phone_list.index` that is in the `word_list.phonerange` for a word, then it is part of the node word. PhoSS collects four words to either side of the node word in order to recreate the textual context. PhoSS combines the phonetic and textual context into one list, which is translated into an R factor vector that replaces the previously empty context column of the dataframe with both the correct phonetic and textual context for each vowel instance. With the context in place, all of the informant information is complete and ready for normalization and summary.

PhoSS is designed to handle multiple normalization flags during one program call. Since this example includes the options for both no normalization and Lobanov normalization, PhoSS first calls the normalization module with the 'No Normalization' option. The normalization module copies the unnormalized dataframe to `informant.normed_data_frame`, which is what all of the summary functions reference. In the case of multiple normalizations, the `informant.data_frame` must remain unchanged so that R can normalize from it to other methods and store the results, one after the other, in `informant.normed_data_frame`. The `informant.sub_data_frame` for each informant is the section of the normalized dataframe that pertains to only that informant. Extracting subtables from the dataframe after normalization preserves the integrity of normalized values for speaker-extrinsic and formant-extrinsic normalization procedures.

PhoSS passes the entire `InformantList` object to the output module. If there are multiple speakers, then PhoSS carries out a comparative summary by default. If the `--no-compare`

flag has been set, PhoSS will not compare speakers. It instead writes an individual summary for each informant. As may be apparent, calling PhoSS with only one informant triggers the individual summary. Since this example uses the `-g` option, the output module calls the direct comparison summary. This differs from the default comparative summary in that the informant is removed from the group in order to make direct comparison of two individuals, or comparison of an individual to another population possible. The default comparative summary calculates group means based on the entire population. The final output files follow the pattern *informant_name.normalizationmethod.phoss*. If no normalization is specified, then the output file will have the `nonormalization.phoss` extension. As each summary is completed, PhoSS prints an update message to stdout. Once all normalization methods have been processed, PhoSS exits.

```
Summary for informant_one using Lobanov written to
  ../testfiles/test_one/informant_one/
Summary for informant_two using Lobanov written to
   ../testfiles/test_one/informant_two/
Summary for informant_one using No Normalization written to
   ../testfiles/test_one/informant_one/
Summary for informant_two using No Normalization written to
   ../testfiles/test_one/informant_two/
```

Both the comparative and individual summaries write some preliminary information about the speaker and vowel set. The file header includes the summary type, normalization method, speaker name, vowel set, and the main informant files.

```
PhoSS , A Phonetic Summarizer for Sociolinguists
Direct Comparison
No Normalization

Speaker: informant_one
Vowel Set: [  IY1  , UW1  , AA1  , AE1 ]
Audio File: /home/user/testfiles/test_one/informant_one/informant_one.wav
Transcript: /home/user/testfiles/test_one/informant_one/informant_one.txt
TextGrid: /home/user/testfiles/test_one/informant_one/informant_one.TextGrid
```

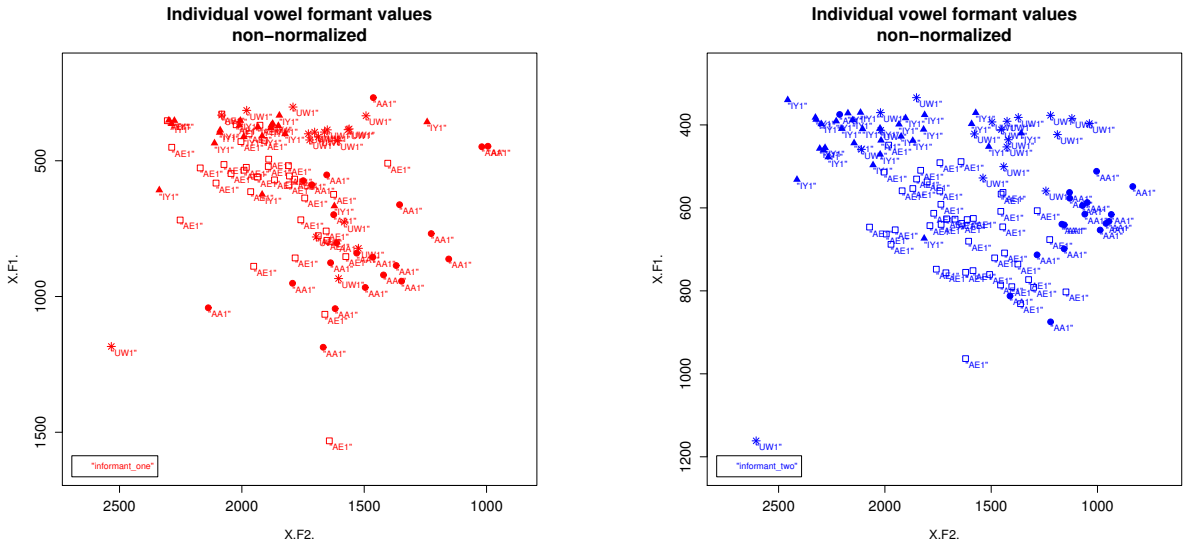
The first step of real summary is describing the overall vowel space for the individual. Harrington's studies of the Queen's Christmas addresses relied heavily on vowel space comparisons from one decade to the next. Just by comparing the absolute vowel space for informants one and two it is clear that informant two has a narrower F1 range and a wider F2 range than informant one. Ladefoged notes that F1 is inversely related to vowel height, while F2 is directly related to vowel backness (2006:188). The minimum and maximum F1 and F2 values reveal that informant one has both the lowest and the highest vowels in the group, and that informant two has the most front and the most back vowels. Figure 4.4 shows the unnormalized vowel plots for speakers one and two.

```
F1 vowel space for informant_one has a range of 1264.135 units,
  from 267.492 to 1531.628.
F2 vowel space for informant_one has a range of 1539.232 units,
  from 994.377 to 2533.609.
```

```
F1 vowel space for informant_two has a range of 827.290 units,
  from 334.256 to 1161.546.
F2 vowel space for informant_two has a range of 1771.553 units,
  from 834.072 to 2605.625.
```

Next, PhoSS compares one informant to another vowel-by-vowel. The UW1 for informant one follows the trend of the speakers' aggregate vowel space. Informant one occupies more vertical space for UW1. Informant one shows generally higher and more front UW1s than informant two. Informant two occupies a vertically smaller, but wider F2 vowel space, corresponding to more front-back variation.

```
Summary for UW1
  F1 vowel space for UW1 has a range of 881.948 units, from 302.065
    to 1184.013.
  F2 vowel space for UW1 has a range of 1040.933 units, from 1492.676
    to 2533.609.
  UW1 for informant_one centers at (F1 518.605, F2 1732.065), which is 0.313
    standard deviations from the average F1 and 0.442 standard deviations from
    the average F2.
```



(a) Vowel space for informant one is taller.

(b) Vowel space for informant two is wider.

Figure 4.4: Individual vowel plots for informants one and two

```
  The rest of the group's average for UW1 occupies vowel space centered at
    (F1 462.863, F2 1556.242) with an F1 standard deviation of 178.257 and
    an F2 standard deviation of 397.572.
```

```
Summary for UW1
  F1 vowel space for UW1 has a range of 827.290 units, from 334.256
    to 1161.546.
  F2 vowel space for UW1 has a range of 1567.448 units, from 1038.176
    to 2605.625.
  UW1 for informant_two centers at (F1 462.863, F2 1556.242), which is 0.219
    standard deviations from the average F1 and 0.705 standard deviations from
    the average F2.
  The rest of the group's average for UW1 occupies vowel space centered at
    (F1 518.605, F2 1732.065) with an F1 standard deviation of 254.855 and
    an F2 standard deviation of 249.396.
```

It is evident from the comparing this information with figure 4.4 that there are some incorrect formant measurements which may affect the mean values. For example, the UW1 token for informant two at 1161, 2605 is at the complete opposite corner from the rest of

the UW1 vowels. That is likely the product of a misalignment. In order to find vowel tokens that differ from the group, the summary module calculates the euclidean distance from the group's mean vowel center, and determines the number of standard deviations each of the informant's vowels is from that center. Vowels are grouped by distance from the center in one-half standard deviation increments, and displayed in context.

```
informant_one
17 of 18 instances are less than one half standard deviation from the mean
  vowel center.

There are 1 instances between 1 and 1.5 standard deviations from the average
  vowel center:
S K UW1 L IH1  : SEE WHEN I STARTED SCHOOL IT WAS UH A


informant_two
18 of 19 instances are less than one half standard deviation from the mean
  vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average
  vowel center:
T Y UW1 IH2 M  : LEAVES THAT'S NOT WHAT YOU HIM I FORGET WHAT
```

The vowel that shows the greatest variation between the two speakers is AA1. Informant one displays seven vowel tokens between one-half and one standard deviation from the group mean and one token between one and one and one-half standard deviations. They have a tendency to occur when the following phone is either [ɹ], or a stop. Ladefoged notes that lip rounding has a tendency to lower second and third formants (2006). If there is any degree of rounding on the AA1s preceding the [ɹ], that may explain those results. However, I find lip rounding highly unlikely before stop consonants. Informant one typically has a lower and more back AA1 than informant two. Looking at the size of a standard deviation reveals that informant two's AA1s have a tighter F1 grouping than informant two, while informant two's AA1s exhibit a slightly closer F2 grouping. This interpretation is confirmed by the vowel plot.

```
AA1
Summary for AA1
  F1 vowel space for AA1 has a range of 919.445 units, from 267.492
    to 1186.937.
  F2 vowel space for AA1 has a range of 1142.936 units, from 994.377
    to 2137.313.
  AA1 for informant_one centers at (F1 780.990, F2 1502.321), which
    is 1.411 standard deviations from the average F1 and 1.176 standard
    deviations from the average F2.
  The rest of the group's average for AA1 occupies vowel space centered at
    (F1 627.192, F2 1150.991) with an F1 standard deviation of 109.032 and
    an F2 standard deviation of 298.868.

  14 of 22 instances are less than one half standard deviation from the mean
    vowel center.

  There are 7 instances between 0.5 and 1 standard deviations from the average
    vowel center:
  K L AA1 K AH0  : OH AND AT TEN O'CLOCK IN THE MORNING EVERYBODY
  D M AA1 R CH  : UP SINGLE FILE AND MARCHED OVER TO CHURCH THEY
  IY1 G AA1 T SH  : WORDS LIKE HERE WE GOT SHEET ROCK ON BOTH
  T R AA1 K AA1  : HERE WE GOT SHEET ROCK ON BOTH SIDES UP
  AA1 K AA1 N B  : WE GOT SHEET ROCK ON BOTH SIDES UP THERE
  S T AA1 R T  : SCHOOL UNTIL THE COTTON STARTED GETTING READY TO PICK
  AH0 V AA1 G AH0  : THE THIRD WEEK OF AUGUST WAS A LITTLE CHURCH

  There are 1 instances between 1 and 1.5 standard deviations from the average
    vowel center:
  S T AA1 R T  : THEN SEE WHEN I STARTED SCHOOL IT WAS UH
```

There are only two vowel tokens that occur between one-half and one standard deviation from the mean for informant two, and they are both followed by either an R or a stop consonant, which holds with the findings for informant one.

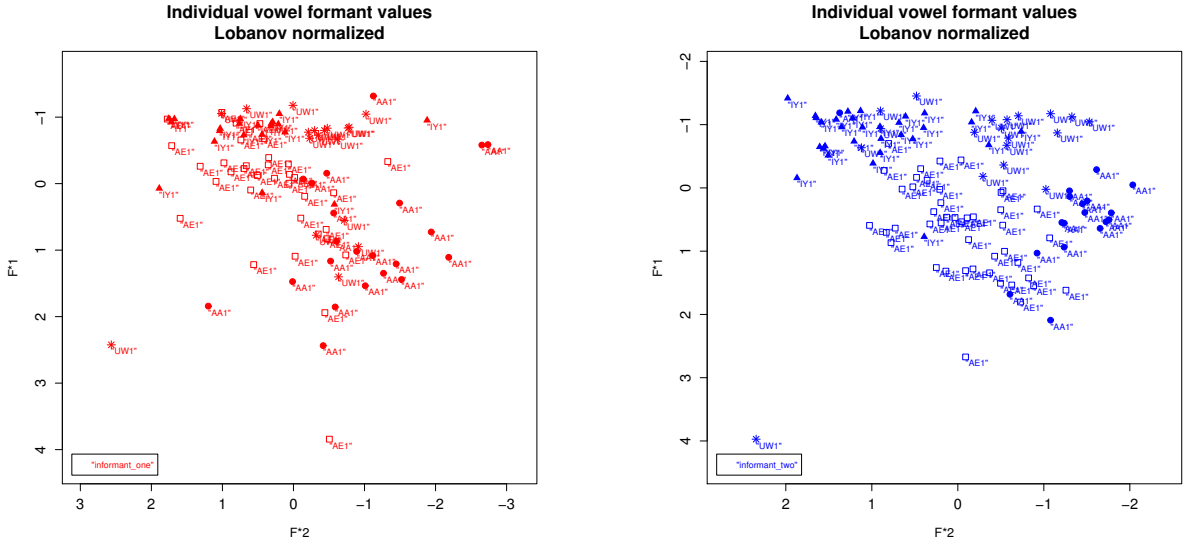
```
Summary for AA1
  F1 vowel space for AA1 has a range of 499.908 units, from 374.821
    to 874.729.
  F2 vowel space for AA1 has a range of 1378.791 units, from 834.072
    to 2212.863.
  AA1 for informant_two centers at (F1 627.192, F2 1150.991), which
    is 0.676 standard deviations from the average F1 and 1.330 standard
    deviations from the average F2.
  The rest of the group's average for AA1 occupies vowel space centered at
```

```
  (F1 780.990, F2 1502.321) with an F1 standard deviation of 227.582 and
  an F2 standard deviation of 264.112.

16 of 18 instances are less than one half standard deviation from the
  mean vowel center.

There are 2 instances between 0.5 and 1 standard deviations from the
  average vowel center:
S N AA1 T HH  : THAT THE LEAVES THAT'S NOT WHAT YOU HIM I
S T AA1 R T  : ON THE DAY WE STARTED THE SCHOOL OH MY
```

After PhoSS writes similar summaries for the rest of the vowels in the vowelset, it repeats the process for the next normalization method. All of the summary comparisons are the same for each normalization method, but the concordance results for are different due to the fact that vowel patterning differs for each normalization procedure. Figure 4.5 shows the same data as in figure 4.4 after having been normalized with the Lobanov method. There are drastic changes in the summary, especially for AE1.



(a) Poor patterning for AE1 is revealed in the concordance.

(b) AE1 patterning is slightly better for informant two.

Figure 4.5: Vowel plots of normalized results using the Lobanov method

The summaries for Lobanov normalization are significantly longer than the summaries with no normalization. The AE1 for informant one only has one exemplar within half of a

standard deviation of the group's mean vowel center. Informant one's wide-spread pattern for AE1 is much more pronounced once the data have been normalized.

The concordance list shows a shift in the R1 phone context as the distance from the mean grows. The vowel tokens closer to the mean are followed primarily by nasals and voiceless fricatives. Between one and one and one-half standard deviations from the mean the context changes to nasals and stops, fricatives falling out almost entirely. The tokens that are farthest from the mean all have sounds from the aforementioned manners of articulation in both the R1 and R2 positions.

```
Summary for AE1
  F*1 values for informant_one's AE1 is 4.914, from -1.071 to 3.843.
  F*2 values for informant_one's AE1 is 3.103, from -1.327 to 1.776.
  The rest of the group's average for AE1 occupies vowel space centered at
    (F*1 0.653, F*2 -0.051) with an F*1 standard deviation of 0.705 and a F*2
    standard deviation of 0.570.
  AE1 for informant_one centers at (F*1 0.093, F*2 0.346), which is 0.794
    standard deviations from the average F*1 and 0.696 standard deviations from
    the average F*2.

  1 of 36 instances are less than one half standard deviation from the mean
    vowel center.

  There are 5 instances between 0.5 and 1 standard deviations from the average
    vowel center:
  OW1 V AE1 N D  : YEAH BIG WOOD STOVE AND THEN AND THEY UH
  P L AE1 N AH0  : AFTER WE GET EVERYTHING PLANTED AND THEN BY THE
  D HH AE1 V S  : FINISHED THEN WE WOULD HAVE SCHOOL UNTIL THE COTTON
  OW1 B AE1 K T  : PICKED THEN YOU'D GO BACK TO SCHOOL AND THEN
  AH0 HH AE1 F AH0  : UP THERE ABOUT A HALF A MILE FROM SCHOOL

  There are 8 instances between 1 and 1.5 standard deviations from the average
    vowel center:
  S F AE1 M L  : THE WINTER TIME THIS FAMILY WOULD FURNISH WOOD FOR
  IY1 HH AE1 D AH0  : OTHER FAMILY AND WE HAD A WATER BUCKET SIT
  D R AE1 NG K  : IN IT AND EVERYBODY DRANK OUT OF THE SAME
  D Z AE1 N D  : DOWN IN THE WOODS AND DID Y'ALL HAVE A
  L HH AE1 V AH0  : WOODS AND DID Y'ALL HAVE A WOOD STOVE YEAH
  V Y AE1 B IH1  : HAVE A WOOD STOVE YEAH BIG WOOD STOVE AND
  UW1 L AE1 N D  : GO BACK TO SCHOOL AND THEN THE THIRD WEEK
  EY1 HH AE1 D W  : WEEK IN AUGUST THEY HAD WHAT THEY CALL REVIVAL
```

```
There are 6 instances between 1.5 and 2 standard deviations from the average
  vowel center:
D DH AE1 T W  : THEY JUST JUST AND THAT WAS THE OLD FOLKS
DH EY1 AE1 D AH0  : ROOMS AND THEN THEY ADDED A THIRD ROOM BUT
EY1 HH AE1 D B  : SIDES UP THERE THEY HAD BOARDS ON ONE SIDE
AY1 D AE1 N D  : BOARDS ON ONE SIDE AND NOTHING ELSE ON THE
D IY0 AE1 N D  : UH TWENTY NINE THIRTY AND THIRTY ONE WE HAD
ER0 F AE1 M L  : NEXT WEEK THE OTHER FAMILY AND WE HAD A

There are 6 instances between 2 and 2.5 standard deviations from the average
  vowel center:
K R AE1 B AH0  : I MEAN WHERE HEARTS CRAB AND KING CROSSES THERE
D HH AE1 V IH1  : LIKE THIS CHURCH WOULD HAVE IT THIRD WEEK IN
D DH AE1 T CH  : WEEK IN JULY AND THAT CHURCH WOULD HAVE THEIRS
DH EY1 AE1 D AH0  : ROOM AND THEN THEY ADDED TWO ROOMS AND THEN
IY1 HH AE1 D AH1  : AND THIRTY ONE WE HAD UH THREE ROOMS AND
lg HH AE1 D T  : THE SAME DIPPER {LG} HAD TO GO TO SOMEBODY'S

There are 10 instances greater than 2.5 standard deviations from the average
  vowel center:
T R AE1 K T  : NOW THEY CALLED IT TRACTED MEETING OH AND AT
NG OW1 AE1 N D  : IT TRACTED MEETING OH AND AT TEN O'CLOCK IN
N D AE1 T T  : TRACTED MEETING OH AND AT TEN O'CLOCK IN THE
T B AE1 K T  : AND THEN THEY MARCHED BACK TO SCHOOL WHAT WAS
Z DH AE1 T K  : TO SCHOOL WHAT WAS THAT CALLED WHAT KIND OF
T R AE1 K T  : WHAT KIND OF MEETING BETRACTED MEETING OH IT WAS
D HH AE1 V DH  : AND THAT CHURCH WOULD HAVE THEIRS THERE WERE REVIVALS
AH0 Z AE1 K SH  : SAYING BUT IT WAS ACTUALLY A REVIVAL
DH ER0 AE1 N D  : ELSE ON THE OTHER AND UH ORIGINALLY IN UH
R T AE1 F T  : TIME IT WOULD START AFTER WE GET EVERYTHING PLANTED
```

The AE1 for informant two exhibits many of the same contexts as informant one. The closest vowel tokens to the group mean are followed directly by stops, nasals and fricatives. Stops continue to dominate the R1 position through the rest of the concordance, even though the vowels furthest from the mean do not exhibit the same R1, R2 contexts that informant one shows. One interesting point from informant two is that the one-half to one standard deviation range has a disproportionate number of fricatives in the L1 position to the rest of the concordance.

```
Summary for AE1
  F*1 values for informant_two's AE1 is 3.375, from -0.703 to 2.672.
  F*2 values for informant_two's AE1 is 2.284, from -1.257 to 1.027.
  The rest of the group's average for AE1 occupies vowel space centered at
    (F*1 0.093, F*2 0.346) with an F*1 standard deviation of 0.937 and a F*2
    standard deviation of 0.714.
  AE1 for informant_two centers at (F*1 0.653, F*2 -0.051), which is 0.598
    standard deviations from the average F*1 and 0.556 standard deviations from
    the average F*2.

  9 of 45 instances are less than one half standard deviation from the mean
    vowel center.

  There are 18 instances between 0.5 and 1 standard deviations from the average
    vowel center:
  Z DH AE1 T S  : CORN THAT THE LEAVES THAT'S NOT WHAT YOU HIM
  L DH AE1 T EH1  : FORGET WHAT YOU CALL THAT ANYWAY THAT WAS HAY
  EY2 DH AE1 T W  : YOU CALL THAT ANYWAY THAT WAS HAY AND THAT
  D DH AE1 T AO1  : THAT WAS HAY AND THAT ALL IN THIS ROOM
  Z DH AE1 T W  : BUT UH THAT WAS THAT WAS I'LL I'LL SAY
  EY1 DH AE1 T W  : WAS I'LL I'LL SAY THAT WAS ONE OF MY
  AE1 D AE1 N D  : GOSSIPING ABOUT HER MAD AND EVERYTHING SO UH LUVIT
  IY1 HH AE1 D DH  : THE BOARD AND WE HAD THIS SCHOOL AT CRABAPPLE
  K R AE1 B AE1  : HAD THIS SCHOOL AT CRABAPPLE AND EBENEZER AND ONE
  AE1 B AE1 P AH0  : HAD THIS SCHOOL AT CRABAPPLE AND EBENEZER AND ONE
  AH0 L AE1 N D  : THIS SCHOOL AT CRABAPPLE AND EBENEZER AND ONE OTHER
  T HH AE1 V EH1  : BUT UH THEY DIDN'T HAVE ANYBODY TO GO TO
  T HH AE1 V EH1  : WERE THEY THEY WOULDN'T HAVE ANYBODY FROM CRABAPPLE CAUSE
  AE1 B AE1 P AH0  : WOULDN'T HAVE ANYBODY FROM CRABAPPLE CAUSE THEY WERE MAD
  ER0 M AE1 D DH  : CRABAPPLE CAUSE THEY WERE MAD THAT OH MY GOODNESS
  N S AE1 T ER0  : TO A MEETING ON SATURDAY MORNING HE CALLED ME
  T HH AE1 V AH0  : THE MONEY AND DIDN'T HAVE ANOTHER JOB SO AND
  K R AE1 B AE1  : GET THE SCHOOL IN CRABAPPLE AND YOU KNOW HOW

  There are 8 instances between 1 and 1.5 standard deviations from the average
    vowel center:
  IH0 NG AE1 NG SH  : MEAN UH CHILDREN BEING ANXIOUS TO LEARN
  K R AE1 B AE1  : WOULDN'T HAVE ANYBODY FROM CRABAPPLE CAUSE THEY WERE MAD
  AE1 B AE1 P AH0  : GET THE SCHOOL IN CRABAPPLE AND YOU KNOW HOW
  S OW1 AE1 N D  : HAVE ANOTHER JOB SO AND WE DIDN'T HAVE A
  OW1 DH AE1 T W  : NINETEEN HUNDRED THIRTY SO THAT WAS UH DURING THE
  AH0 N AE1 N D  : UH DURING THE DEPRESSION AND UH WE WALKED SIX
  L Z AE1 F T  : WE WALKED SIX MILES AFTER CHURCH THAT DAY WE
  CH DH AE1 T D  : SIX MILES AFTER CHURCH THAT DAY WE LIVED AT
```

```
There are 6 instances between 1.5 and 2 standard deviations from the average
  vowel center:
N S AE1 T IH2  : HE CALLED ME ON SATURDAY AFTERNOON AND WANTED TO
T HH AE1 V AH0  : SO AND WE DIDN'T HAVE A CAR WE WE
V D AE1 T K  : THAT DAY WE LIVED AT CRABAPPLE WE WENT DOWN
K R AE1 B AE1  : DAY WE LIVED AT CRABAPPLE WE WENT DOWN TO
AE1 B AE1 P AH0  : DAY WE LIVED AT CRABAPPLE WE WENT DOWN TO
AY1 D AE1 D IY0  : MILES TO GET MY DADDY'S CAR SO I COULD

There are 3 instances between 2 and 2.5 standard deviations from the average
  vowel center:
AW1 DH AE1 T W  : AND YOU KNOW HOW THAT WORKS THESE OTHER PEOPLE
L S AE1 T ER0  : GOING OVER THERE UNTIL SATURDAY SO WE DID NOTHING
IH0 NG AE1 N D  : TO THE SCHOOL BUILDING AND FODDER YOU KNOW WHAT

There are 1 instances greater than 2.5 standard deviations from the average
  vowel center:
Z DH AE1 T S  : KNOW WHAT FODDER IS THAT'S THE CORN THAT THE
```

While future GUIed versions of PhoSS may be able to perform positional sorting, like WordSmith Tools does, sorting is not strictly required to be able to find trends in the data. The trends are there, and just need to be described and presented in a way that human eyes can digest the information. The entirety of the output files for both informants are included as Appendix A.

Chapter 5

Future Work and Conclusions

I have been writing PhoSS as a beginning to a tool that is somewhere in between WordSmith Tools and Praat. Sociolinguistics has its moorings in text analysis since the days of Labov's New York and Martha's Vineyard studies, and it will always have an inherantly text-based dimension. For that reason, text display and text visualization are necessary facilities for a tool that is useful for sociolinguistic studies. PhoSS and text-centric output, along with the graphing facilities of NORM's vowels library makes for a tool that can handle both of the main venues for data representation in sociolinguistics and in corpus linguistics. PhoSS would benefit first and foremost from a GUI. This would necessitate some redesigning of the code, but I believe that many of the modules would function much the same as they do now.

Redesigning the PhoSS internal data structures may be necessary in order to make PhoSS a much larger program. A dedicated computer scientist could improve on the current program design with plans to make it extensible.

A GUI with menus would also make user-defined options in PhoSS much easier to control than in its command line state, where command line flags and program interruptions for user input are the main sources of customization. My vision for future work in computational sociolinguistics is to see PhoSS grow into a full-featured piece of software that fills the niche for sociolinguistic research of unstructured speech corpora in the same way that WordSmith Tools fills the niche for text analysis and that Praat fills the need for phonetic analysis. I believe that PhoSS is a beginning. I have striven to make PhoSS adhere to the current best practices, but in order to make it grow into a larger, widely-used program, it would need to

go through a user test phase, where linguists would try to carry out their own experiments and respond with their research needs.

Apart from design and aesthetic improvements to PhoSS, I have wanted several pieces of information throughout this project. The best study that I could want is an in-depth examination of alignment accuracy on the University of Georgia Linguistic Atlas Project corpora. I have noted earlier Keelan Evanini's alignment accuracy results from his dissertation. His dissertation corpus was not an interview corpus, so it is closer to the ideal case for alignment accuracy. The Linguistic Atlas Project contains more than enough recordings to turn region-by-region testing into a dissertation. The results, along with investigating methods for improving alignment accuracy on interview corpora would be of academic interest as well as industry concern. Not all commercial voice recognition systems are able to distinguish and accurately process multiple speakers on one recording (which is always the case for interview corpora). While alignment is possible, and where I have checked it visually, it is accurate, I would have liked to have numbers to support using interview corpora, or to let me know whether it is a problematic endeavor.

I would also like to have been able to train sound models using audio sampled at either 41000 or 96000 Hz in order to test my hypothesis that higher-sampled training data would lead to higher alignment accuracy on high-sampled audio. The results of comparative testing took up only a small section of one paper in Yuan and Liberman 2008, but the amount of work that goes into creating, checking, and running the training data to make sound models is substantial. I believe that answering this question would make a worthy thesis, since improving the P2FA accuracy at higher sampling rates would mean alignments improve with increased levels of phonetic detail, instead of decreased detail.

Another study of some interest, perhaps less related to the rest of the areas for improvement, would be a study of the SCOTUS corpus that parallels Jonathan Harrington's examination of vowel shift in the Queen's English. His results are widely published, and his methodology should be easy enough to replicate. Having this parallel study would be something akin

to publishing the LOB after the Brown corpus had been created. It could ultimately help to clarify the validity of the apparent time hypothesis. It is apparent, with or without the hypothesis, that pronunciation changes over time. Disproving the apparent time hypothesis just means that language change happens faster than the hypothesis suggests.

Ultimately, I want to help sociolinguistics, and especially the analysis of speech corpora to continue to develop, just as text-based corpus study has done. It will take new tools, and new ideas for visualization and information representation. I believe that the analysis of speech corpora is in the "CD phase". The 1990s corpus vogue was to release a corpus on CD. The CD included some proprietary tool for searching the corpus, which invariably lacked functionality critical to someone's research questions. That is no longer the goal. Text corpora are easily distributable via the internet, so the focus has shifted to tools that allow processing and analyzing unstructured text. As tool development continues, speech corpus analysis can progress to better processing of less structured and unstructured speech corpora.

Bibliography

[1] Adank, Patti; Smits, Roel; and van Hout, Roeland (2004) A Comparison of Vowel Normalization Procedures for Language Variation Research. *Journal of the Acoustical Society of America.* 116.5:3099-3107.

[2] Baayen, R.H. (2008) *Analyzing Linguistic Data: A Practical Introduction to Statistics using R.* Cambridge University Press: New York.

[3] Biber, Douglas (1988) *Variation Across Speech and Writing.* Cambridge: Cambridge University Press.

[4] Biber, Douglas (1998) *Corpus Linguistics: Investigating Language Structure and Use.* United Kingdom: Cambridge University Press.

[5] Bickerstaff, Garrison E., Jr. (2010) *Construction and application of Bounded Virtual Corpora of British and American English.* University of Georgia. http://purl.galileo.usg.edu/uga_etd/bickerstaff_garrison_e_201012_phd.

[6] Boersma, Paul, and Weenink, David (2010) *Praat: Doing phonetics by computer (version 5.1).* Technical report, University of Amsterdam, http://www.praat.org/.

[7] Cho, Yun Jeong (2008) The Acoustic Characteristics of American English Diphthongs by Native Speakers and Korean Learners. *Language and Linguistics.* 42:197-223.

[8] Cox, Felicity (2006) The Acoustic Characteristics of /hVd/ Vowels in the Speech of some Australian Teenagers. *Australian Journal of Linguistics.* 22.2:147-179.

[9] Davies, Mark. (2008) The Corpus of Contemporary American English (COCA): 410+ million words, 1990-present. Available online at http://www.americancorpus.org.

[10] Deterding, D. (2000) Measurements of the /eɪ/ and the /əʊ/ vowels of young English speakers in Singapore. In A. Brown, D. Deterding and E. L. Low eds., *The English Language in Singapore: Research on Pronunciation*, 93-99. Singapore: Singapore Association for Applied Linguistics.

[11] Disner, S.F. (1980) Evaluation of vowel normalization procedures. *Journal of the Acoustical Society of America*. 67.1:253-261.

[12] Evanini, Keelan (2009) *The Permeability of Dialect Boundaries: A Case Study of the Region Surrounding Erie, Pennsylvania*. Publicly accessable Penn Dissertations. Paper 86. http://repository.upenn.edu/edissertations/86.

[13] Evanini, Keelan; Isard, Stephen; Liberman, Mark (2009) Automatic formant extraction for sociolinguistic analysis of large corpora. *Proceedings of Interspeech 2009*. http://www.evanini.com/papers/evanini_INTERSPEECH09b.pdf

[14] Fabricius, Anne; Watt, Dominic; Johnson, Ezra (2009) A comparison of three speaker-intrinsic vowel formant frequency normalization algorithms for sociophonetics. *Language Variation and Change* 21:413-435.

[15] Harrington, Jonathan; Palethorpe, Sallyanne; and Watson, Catherine (2000). Vowel change in Received Pronunciation: evidence from the Queen's English [Abstract]. *Proceedings of the 7th Australian International Conference on Speech Science and Technology.*

[16] Harrington, Jonathan; Palethorpe, Sallyanne; and Watson, Catherine (2005). Deepening or lessening the divide between diphthongs? An analysis of the Queen's annual Christmas Broadcasts. In W. J. Hardcastle and J. M. Beck, eds., *A Figure of Speech: A Festchrift for John Laver*, 227-262. New Jersey: Lawrence Erlbaum Associates.

[17] Harrington, Jonathan (2010) *Phonetic Analysis of Speech Corpora*. Malaysia: Wiley-Blackwell.

[18] Hay, Jennifer, and Drager, Katie (2007) Sociophonetics *Annual Review of Anthropology.* 36:89-103.

[19] (2009) Hidden Markov Model Toolkit (HTK). University of Cambridge. http://htk.eng.cam.ac.uk/.

[20] Kennedy, Greame (1998) *An Introduction to Corpus Linguistics*. Harlow: Addison Wesley Longman.

[21] Kučera, Henry, and Francis, W. Nelson (1964) Brown University Standard Corpus of Present-Day English.

[22] Kretzschmar, William A., Jr., and Dunn, Josh (2011) *Implicational Scaling in Southern Speech Features*. ADS/LSA, Pittsburgh.

[23] Labov, William (1963) The Social Motivation of a Sound Change. *Word*. 19:273-309.

[24] Labov, William (1966) *The Social Stratification of English in New York City*. Washington, D. C.: Center for Applied Linguistics.

[25] Labov, William (2006) A Sociolinguistic Perspective on Sociophonetic Research. *Journal of Phonetics*. 34.4:500-515.

[26] Labov, William; Ash, Sharon; and Boberg, Charles (2006) *Atlas of North American English: Phonetics, Phonology and Sound Change*. Germany: Mouton de Gruyter.

[27] Ladefoged, Peter (2006) *A Course in Phonetics*. USA: Thomson Wadsworth.

[28] Lennes, Mietta (2003) *collect_formant_data_from_files.praat*. Available http://www.helsinki.fi/~lennes/praat-scripts/public/collect_formant_data_from_files.praat.

[29] Manning, Chris, and Schütze, Hinrich (1999) *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

[30] Mosteller, Frederick, and Wallace, David L. (1963) Inference in an Authorship Problem. *Journal of the American Statistical Association*. 58.302:275-309.

[31] Nearey, Terrance M. (1977) *Phonetic Feature Systems for Vowels.* Dissertation. University of Alberta.

[32] Oakes, Michael P. (1998) *Statistics for Corpus Linguistics*. Edinburgh University Press.

[33] Plichta, Bartek (2011) *Akustyk*. Online Resource: http://bartus.org/akustyk/documentation.php.

[34] Stubbs, Michael (1996). *Text and Corpus Analysis: Computer Assisted Studies of Language and Culture*. Oxford: Wiley-Blackwell.

[35] Thomas, Erik R., and Kendall, Tyler (2007) *NORM: The vowel normalization and plotting suite.* Online Resource: http://ncslaap.lib.ncsu.edu/tools/norm/

[36] Wakita, Hisashi (1977) Normalization of Vowels by Vocal-Tract Length and Its Application to Vowel Identification. *IEEE Transactions on Acoustics, Speech, and Signal Processing.* ASSP-25.2:183-192.

[37] Watt, Dominic and Fabricius, Anne (2002) Evaluation of a Technique for Improving the Mapping of Multiple Speakers' Vowel Spaces in the F_1-F_2 Plane. *Leeds Working Papers in Linguistics and Phonetics*. 9:159-173.

[38] Yang, Byunggon (1996) A Comparative Study of American English and Korean Vowels Produced by Male and Female Speakers. *Journal of Phonetics.* 24:245-261.

[39] Yuan, J., Liberman, M., (2008) Speaker identification on the SCOTUS corpus. *Proceedings of Acoustics 2008*, 5687-5690.

[40] Zwicker, Eberhard (1961) Subdivision of the Audible Frequency Range into Critical Bands (Frequenzgruppen). *The Journal of the Acoustical Society of America*, 33:248.

Appendix A

Informant Output Files

Listing A.1: Output file for `informant_one` with no normalization

```
PhoSS, A Phonetic Summarizer for Sociolinguists
Comparative Summary
No Normalization

Speaker: informant_one
Vowel Set: ['IY1', 'UW1', 'AA1', 'AE1']
Diphthongs: ['AY1']
Audio File: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.wav
Transcript: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.txt
TextGrid: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.TextGrid

F1 vowel space for informant_one has a range of 1264.135 units, from
    267.492 to 1531.628.
F2 vowel space for informant_one has a range of 1539.232 units, from
    994.377 to 2533.609.

Summary for IY1
```

F1 vowel space for IY1 has a range of 333.777 units, from 332.876 to 666.653.

F2 vowel space for IY1 has a range of 1094.711 units, from 1242.770 to 2337.481.

IY1 for informant_one centers at (F1 413.040, F2 1966.119), which is 0.107 standard deviations from the average F1 and 0.172 standard deviations from the average F2.

The group average for IY1 occupies vowel space centered at (F1 421.256, F2 2011.573) with an F1 standard deviation of 76.924 and an F2 standard deviation of 264.997.

All 21 instances of IY1 for informant_one are less than one half standard deviation from the mean vowel center.

Summary for UW1

F1 vowel space for UW1 has a range of 881.948 units, from 302.065 to 1184.013.

F2 vowel space for UW1 has a range of 1040.933 units, from 1492.676 to 2533.609.

UW1 for informant_one centers at (F1 518.605, F2 1732.065), which is 0.132 standard deviations from the average F1 and 0.265 standard deviations from the average F2.

The group average for UW1 occupies vowel space centered at (F1 489.981, F2 1641.777) with an F1 standard deviation of 217.616 and an F2 standard deviation of 341.087.

17 of 18 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

S K UW1 L IH1 : SEE WHEN I STARTED SCHOOL IT WAS UH A

Summary for AA1

F1 vowel space for AA1 has a range of 919.445 units, from 267.492 to 1186.937.

F2 vowel space for AA1 has a range of 1142.936 units, from 994.377 to 2137.313.

AA1 for informant_one centers at (F1 780.990, F2 1502.321), which is 0.350 standard deviations from the average F1 and 0.481 standard deviations from the average F2.

The group average for AA1 occupies vowel space centered at (F1 711.781, F2 1344.223) with an F1 standard deviation of 197.674 and an F2 standard deviation of 328.373.

20 of 22 instances are less than one half standard deviation from the mean vowel center.

There are 2 instances between 0.5 and 1 standard deviations from the average vowel center:

S T AA1 R T : THEN SEE WHEN I STARTED SCHOOL IT WAS UH

D M AA1 R CH : UP SINGLE FILE AND MARCHED OVER TO CHURCH THEY

Summary for AE1

F1 vowel space for AE1 has a range of 1203.624 units, from 328.003 to 1531.628.

F2 vowel space for AE1 has a range of 900.722 units, from 1403.876 to 2304.598.

AE1 for informant_one centers at (F1 613.109, F2 1889.452), which is 0.136 standard deviations from the average F1 and 0.556 standard deviations from the average F2.

The group average for AE1 occupies vowel space centered at (F1 636.590, F2 1748.710) with an F1 standard deviation of 172.773 and an F2 standard deviation of 253.119.

35 of 36 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

T R AE1 K T : NOW THEY CALLED IT TRACTED MEETING OH AND AT

For target one of AY1:

F1 vowel space for AY1 has a range of 751.590 units, from 499.209 to 1250.799.

F2 vowel space for AY1 has a range of 924.277 units, from 1108.934 to 2033.211.

AY1 for informant_one centers at (F1 755.709, F2 1519.747), which is 0.287 standard deviations from the average F1 and 0.491 standard deviations from the average F2.

The group average for AY1 occupies vowel space centered at (F1 699.880, F2 1416.238) with an F1 standard deviation of 194.244 and an F2 standard deviation of 210.600.

All 21 instances of AY1 for informant_one are less than one half standard deviation from the mean vowel center.

For target two of AY1:

gl.F1 vowel space for AY1 has a range of 964.290 units, from 551.582 to 1515.872.

gl.F2 vowel space for AY1 has a range of 658.564 units, from 1280.457 to 1939.021.

AY1 for informant_one centers at (gl.F1 894.595, gl.F2 1624.599), which is 0.466 standard deviations from the average gl.F1 and 0.507 standard deviations from the average gl.F2.

The group average for AY1 occupies vowel space centered at (gl.F1 785.293, gl.F2 1488.784) with an gl.F1 standard deviation of 234.540 and an gl.F2 standard deviation of 267.948.

20 of 21 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

EH1 N AY1 S T : BUT THEN SEE WHEN I STARTED SCHOOL IT WAS

Listing A.2: Output file for `informant_two` with no normalization

```
PhoSS, A Phonetic Summarizer for Sociolinguists
Comparative Summary
No Normalization

Speaker: informant_two
Vowel Set: ['IY1', 'UW1', 'AA1', 'AE1']
Diphthongs: ['AY1']
Audio File: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.wav
Transcript: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.txt
TextGrid: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.TextGrid

F1 vowel space for informant_two has a range of 827.290 units, from
   334.256 to 1161.546.
F2 vowel space for informant_two has a range of 1771.553 units, from
    834.072 to 2605.625.

Summary for IY1
  F1 vowel space for IY1 has a range of 333.827 units, from 339.678
     to 673.504.
  F2 vowel space for IY1 has a range of 1098.217 units, from
     1358.834 to 2457.051.
  IY1 for informant_two centers at (F1 426.822, F2 2042.365), which
     is 0.072 standard deviations from the average F1 and 0.116
     standard deviations from the average F2.
```

The group average for IY1 occupies vowel space centered at (F1 421.256, F2 2011.573) with an F1 standard deviation of 76.924 and an F2 standard deviation of 264.997.

All 31 instances of IY1 for informant_two are less than one half standard deviation from the mean vowel center.

Summary for UW1

F1 vowel space for UW1 has a range of 827.290 units, from 334.256 to 1161.546.

F2 vowel space for UW1 has a range of 1567.448 units, from 1038.176 to 2605.625.

UW1 for informant_two centers at (F1 462.863, F2 1556.242), which is 0.125 standard deviations from the average F1 and 0.251 standard deviations from the average F2.

The group average for UW1 occupies vowel space centered at (F1 489.981, F2 1641.777) with an F1 standard deviation of 217.616 and an F2 standard deviation of 341.087.

18 of 19 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

T Y UW1 IH2 M : LEAVES THAT'S NOT WHAT YOU HIM I FORGET WHAT

Summary for AA1

F1 vowel space for AA1 has a range of 499.908 units, from 374.821 to 874.729.

F2 vowel space for AA1 has a range of 1378.791 units, from 834.072 to 2212.863.

AA1 for informant_two centers at (F1 627.192, F2 1150.991), which is 0.428 standard deviations from the average F1 and 0.588 standard deviations from the average F2.

The group average for AA1 occupies vowel space centered at (F1 711.781, F2 1344.223) with an F1 standard deviation of 197.674 and an F2 standard deviation of 328.373.

17 of 18 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

S N AA1 T HH : THAT THE LEAVES THAT'S NOT WHAT YOU HIM I

Summary for AE1

F1 vowel space for AE1 has a range of 514.679 units, from 448.642 to 963.320.

F2 vowel space for AE1 has a range of 923.911 units, from 1148.343 to 2072.255.

AE1 for informant_two centers at (F1 655.375, F2 1636.116), which is 0.109 standard deviations from the average F1 and 0.445 standard deviations from the average F2.

The group average for AE1 occupies vowel space centered at (F1 636.590, F2 1748.710) with an F1 standard deviation of 172.773 and an F2 standard deviation of 253.119.

All 45 instances of AE1 for informant_two are less than one half standard deviation from the mean vowel center.

For target one of AY1:

F1 vowel space for AY1 has a range of 572.554 units, from 405.476 to 978.031.

F2 vowel space for AY1 has a range of 755.973 units, from 1014.011 to 1769.984.

AY1 for informant_two centers at (F1 646.588, F2 1317.435), which is 0.274 standard deviations from the average F1 and 0.469 standard deviations from the average F2.

The group average for AY1 occupies vowel space centered at (F1 699.880, F2 1416.238) with an F1 standard deviation of 194.244 and an F2 standard deviation of 210.600.

All 22 instances of AY1 for informant_two are less than one half standard deviation from the mean vowel center.

For target two of AY1:

gl.F1 vowel space for AY1 has a range of 538.649 units, from 406.010 to 944.660.

gl.F2 vowel space for AY1 has a range of 1233.676 units, from 976.282 to 2209.958.

AY1 for informant_two centers at (gl.F1 680.958, gl.F2 1359.143), which is 0.445 standard deviations from the average gl.F1 and 0.484 standard deviations from the average gl.F2.

The group average for AY1 occupies vowel space centered at (gl.F1 785.293, gl.F2 1488.784) with an gl.F1 standard deviation of 234.540 and an gl.F2 standard deviation of 267.948.

21 of 22 instances are less than one half standard deviation from the mean vowel center.

There are 1 instances between 0.5 and 1 standard deviations from the average vowel center:

T R AY1 NG T : OF THE BIG ONES TRYING TO GET THE SCHOOL

Listing A.3: Output file for `informant_one` with Lobanov normalization

```
PhoSS, A Phonetic Summarizer for Sociolinguists
Comparative Summary
Lobanov

Speaker: informant_one
Vowel Set: ['IY1', 'UW1', 'AA1', 'AE1']
Diphthongs: ['AY1']
Audio File: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.wav
Transcript: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.txt
TextGrid: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_one/informant_one.TextGrid

F*1 vowel space for informant_one has a range of 4.842 units, from
   -1.397 to 3.445.
F*2 vowel space for informant_one has a range of 5.374 units, from
   -2.640 to 2.734.

Summary for IY1
  F*1 vowel space for IY1 has a range of 1.278 units, from -1.146 to
      0.132.
  F*2 vowel space for IY1 has a range of 3.821 units, from -1.772 to
      2.049.
  IY1 for informant_one centers at (F*1 -0.839, F*2 0.753), which is
      0.148 standard deviations from the average F*1 and 0.262
     standard deviations from the average F*2.
```

The group average for IY1 occupies vowel space centered at (F*1 -0.896, F*2 0.956) with an F*1 standard deviation of 0.385 and an F*2 standard deviation of 0.775.

12 of 21 instances are less than one half standard deviation from the mean vowel center.

There are 6 instances between 0.5 and 1 standard deviations from the average vowel center:

N S IY1 W EH1 : BUT THEN SEE WHEN I STARTED SCHOOL

D M IY1 T IH0 : THEY CALLED IT TRACTED MEETING OH AND AT TEN

D M IY1 T IH0 : KIND OF MEETING BETRACTED MEETING OH IT WAS UH

T SH IY1 T R : LIKE HERE WE GOT SHEET ROCK ON BOTH SIDES

T W IY1 K AO1 : EVERYBODY GOT EVERYTHING WHAT WE CALLED LAID BY ITS

D W IY1 K AH0 : AND THEN THE THIRD WEEK OF AUGUST WAS A

There are 2 instances between 1 and 1.5 standard deviations from the average vowel center:

IY1 W IY1 K S : ABOUT TWO OR THREE WEEKS UNTIL EVERYBODY GOT THEIR

D W IY1 K AH0 : WHAT IN THE THIRD WEEK IN AUGUST THEY HAD

There are 1 instances between 2 and 2.5 standard deviations from the average vowel center:

D W IY1 K IH1 : WOULD HAVE IT THIRD WEEK IN JULY AND THAT

Summary for UW1

F*1 vowel space for UW1 has a range of 3.378 units, from -1.264 to 2.114.

F*2 vowel space for UW1 has a range of 3.634 units, from -0.900 to 2.734.

UW1 for informant_one centers at (F*1 -0.435, F*2 -0.064), which is 0.131 standard deviations from the average F*1 and 0.032 standard deviations from the average F*2.

The group average for UW1 occupies vowel space centered at (F*1 -0.573, F*2 -0.094) with an F*1 standard deviation of 1.056 and an F*2 standard deviation of 0.924.

10 of 18 instances are less than one half standard deviation from the mean vowel center.

There are 7 instances between 0.5 and 1 standard deviations from the average vowel center:

S K UW1 L W : THEY MARCHED BACK TO SCHOOL WHAT WAS THAT CALLED

N R UW1 M AH0 : ORIGINALLY WAS A ONE ROOM AND THEN THEY ADDED

D T UW1 R UW1 : AND THEN THEY ADDED TWO ROOMS AND THEN THEY

T T UW1 ER0 TH : OF SCHOOL FOR ABOUT TWO OR THREE WEEKS UNTIL

N Y UW1 D G : THEIR COTTON PICKED THEN YOU'D GO BACK TO SCHOOL

S K UW1 L AE1 : YOU'D GO BACK TO SCHOOL AND THEN THE THIRD

S K UW1 L W : HALF A MILE FROM SCHOOL WHAT IN THE THIRD

There are 1 instances between 2 and 2.5 standard deviations from the average vowel center:

S K UW1 L IH1 : SEE WHEN I STARTED SCHOOL IT WAS UH A

Summary for AA1

F*1 vowel space for AA1 has a range of 3.522 units, from -1.397 to 2.125.

F*2 vowel space for AA1 has a range of 3.991 units, from -2.640 to 1.351.

AA1 for informant_one centers at (F*1 0.570, F*2 -0.866), which is 0.128 standard deviations from the average F*1 and 0.143 standard deviations from the average F*2.

The group average for AA1 occupies vowel space centered at (F*1 0.468, F*2 -0.987) with an F*1 standard deviation of 0.796 and an F*2 standard deviation of 0.848.

11 of 22 instances are less than one half standard deviation from the mean vowel center.

There are 7 instances between 0.5 and 1 standard deviations from the average vowel center:

K L AA1 K AH0 : OH AND AT TEN O'CLOCK IN THE MORNING EVERYBODY

D M AA1 R CH : UP SINGLE FILE AND MARCHED OVER TO CHURCH THEY

IY1 G AA1 T SH : WORDS LIKE HERE WE GOT SHEET ROCK ON BOTH

T R AA1 K AA1 : HERE WE GOT SHEET ROCK ON BOTH SIDES UP

AA1 K AA1 N B : WE GOT SHEET ROCK ON BOTH SIDES UP THERE

AH0 K AA1 T AH0 : HAVE SCHOOL UNTIL THE COTTON STARTED GETTING READY TO

AH0 V AA1 G AH0 : THE THIRD WEEK OF AUGUST WAS A LITTLE CHURCH

There are 4 instances between 1 and 1.5 standard deviations from the average vowel center:

S T AA1 R T : THEN SEE WHEN I STARTED SCHOOL IT WAS UH

EY1 M AA1 R CH : OVER AND THEN THEY MARCHED BACK TO SCHOOL WHAT

T W AA1 Z AH1 : BETRACTED MEETING OH IT WAS UH IT WAS JUST

AH0 N AA1 N AO1 : THERE WERE REVIVALS GOING ON ALL SUMMER BUT THEY

Summary for AE1

F*1 vowel space for AE1 has a range of 4.610 units, from -1.165 to 3.445.

F*2 vowel space for AE1 has a range of 3.145 units, from -1.210 to 1.935.

AE1 for informant_one centers at (F*1 -0.073, F*2 0.485), which is 0.400 standard deviations from the average F*1 and 0.336 standard deviations from the average F*2.

The group average for AE1 occupies vowel space centered at (F*1 0.258, F*2 0.259) with an F*1 standard deviation of 0.827 and an F*2 standard deviation of 0.673.

1 of 36 instances are less than one half standard deviation from the mean vowel center.

There are 12 instances between 0.5 and 1 standard deviations from the average vowel center:

DH EY1 AE1 D AH0 : ROOMS AND THEN THEY ADDED A THIRD ROOM BUT

S F AE1 M L : THE WINTER TIME THIS FAMILY WOULD FURNISH WOOD FOR

ER0 F AE1 M L : NEXT WEEK THE OTHER FAMILY AND WE HAD A

IY1 HH AE1 D AH0 : OTHER FAMILY AND WE HAD A WATER BUCKET SIT

D R AE1 NG K : IN IT AND EVERYBODY DRANK OUT OF THE SAME

D Z AE1 N D : DOWN IN THE WOODS AND DID Y'ALL HAVE A

L HH AE1 V AH0 : WOODS AND DID Y'ALL HAVE A WOOD STOVE YEAH

OW1 V AE1 N D : YEAH BIG WOOD STOVE AND THEN AND THEY UH

P L AE1 N AH0 : AFTER WE GET EVERYTHING PLANTED AND THEN BY THE
D HH AE1 V S : FINISHED THEN WE WOULD HAVE SCHOOL UNTIL THE
COTTON
OW1 B AE1 K T : PICKED THEN YOU'D GO BACK TO SCHOOL AND THEN
AH0 HH AE1 F AH0 : UP THERE ABOUT A HALF A MILE FROM SCHOOL

There are 8 instances between 1 and 1.5 standard deviations from the average vowel center:
D DH AE1 T W : THEY JUST JUST AND THAT WAS THE OLD FOLKS
EY1 HH AE1 D B : SIDES UP THERE THEY HAD BOARDS ON ONE SIDE
AY1 D AE1 N D : BOARDS ON ONE SIDE AND NOTHING ELSE ON THE
D IY0 AE1 N D : UH TWENTY NINE THIRTY AND THIRTY ONE WE HAD
IY1 HH AE1 D AH1 : AND THIRTY ONE WE HAD UH THREE ROOMS AND
V Y AE1 B IH1 : HAVE A WOOD STOVE YEAH BIG WOOD STOVE AND
UW1 L AE1 N D : GO BACK TO SCHOOL AND THEN THE THIRD WEEK
EY1 HH AE1 D W : WEEK IN AUGUST THEY HAD WHAT THEY CALL REVIVAL

There are 7 instances between 1.5 and 2 standard deviations from the average vowel center:
Z DH AE1 T K : TO SCHOOL WHAT WAS THAT CALLED WHAT KIND OF
T R AE1 K T : WHAT KIND OF MEETING BETRACTED MEETING OH IT WAS
D HH AE1 V IH1 : LIKE THIS CHURCH WOULD HAVE IT THIRD WEEK IN
D DH AE1 T CH : WEEK IN JULY AND THAT CHURCH WOULD HAVE THEIRS
DH EY1 AE1 D AH0 : ROOM AND THEN THEY ADDED TWO ROOMS AND THEN
DH ER0 AE1 N D : ELSE ON THE OTHER AND UH ORIGINALLY IN UH
lg HH AE1 D T : THE SAME DIPPER {LG} HAD TO GO TO SOMEBODY'S

There are 5 instances between 2 and 2.5 standard deviations from the average vowel center:

K R AE1 B AH0 : I MEAN WHERE HEARTS CRAB AND KING CROSSES THERE

NG OW1 AE1 N D : IT TRACTED MEETING OH AND AT TEN O'CLOCK IN

T B AE1 K T : AND THEN THEY MARCHED BACK TO SCHOOL WHAT WAS

AH0 Z AE1 K SH : SAYING BUT IT WAS ACTUALLY A REVIVAL

R T AE1 F T : TIME IT WOULD START AFTER WE GET EVERYTHING PLANTED

There are 3 instances greater than 2.5 standard deviations from the average vowel center:

T R AE1 K T : NOW THEY CALLED IT TRACTED MEETING OH AND AT

N D AE1 T T : TRACTED MEETING OH AND AT TEN O'CLOCK IN THE

D HH AE1 V DH : AND THAT CHURCH WOULD HAVE THEIRS THERE WERE REVIVALS

For target one of AY1:

F*1 vowel space for AY1 has a range of 2.879 units, from -0.509 to 2.370.

F*2 vowel space for AY1 has a range of 3.227 units, from -2.240 to 0.987.

AY1 for informant_one centers at (F*1 0.473, F*2 -0.805), which is 0.003 standard deviations from the average F*1 and 0.076 standard deviations from the average F*2.

The group average for AY1 occupies vowel space centered at (F*1 0.470, F*2 -0.761) with an F*1 standard deviation of 0.916 and an F*2 standard deviation of 0.577.

9 of 21 instances are less than one half standard deviation from the mean vowel center.

```
  There are 8 instances between 0.5 and 1 standard deviations from
     the average vowel center:
  IH0 V AY1 V AH0  : WELL THEY CALL IT REVIVAL NOW THEY CALLED IT
  L L AY1 N D  : MORNING EVERYBODY IN SCHOOL LINED UP SINGLE FILE
     AND
  L F AY1 L AH0  : SCHOOL LINED UP SINGLE FILE AND MARCHED OVER TO
  T K AY1 N D  : WAS THAT CALLED WHAT KIND OF MEETING BETRACTED
     MEETING
  R L AY1 K DH  : WHERE ONCE A YEAR LIKE THIS CHURCH WOULD HAVE
  IY0 N AY1 N TH  : ORIGINALLY IN UH TWENTY NINE THIRTY AND THIRTY
     ONE
  AH0 T AY1 M EH1  : AND THEN BY THE TIME EVERYBODY GOT EVERYTHING
     WHAT
  AH0 M AY1 L F  : ABOUT A HALF A MILE FROM SCHOOL WHAT IN

  There are 4 instances between 1 and 1.5 standard deviations from
     the average vowel center:
  EH1 N AY1 S T  : BUT THEN SEE WHEN I STARTED SCHOOL IT WAS
  IH0 V AY1 V AH0  : HAD WHAT THEY CALL REVIVAL WELL THEY CALL IT
  IY0 V AY1 V AH0  : HAVE THEIRS THERE WERE REVIVALS GOING ON ALL
     SUMMER
  AO1 S AY1 M IY1  : KING ROAD AND CROSS I MEAN WHERE HEARTS CRAB

For target two of AY1:
  F*1 gl vowel space for AY1 has a range of 3.693 units, from -0.308
      to 3.385.
  F*2 gl vowel space for AY1 has a range of 2.299 units, from -1.641
      to 0.658.
```

AY1 for informant_one centers at (F*1 gl 0.473, F*2 gl -0.805), which is 0.003 standard deviations from the average F*1 gl and 0.076 standard deviations from the average F*2 gl.

The group average for AY1 occupies vowel space centered at (F*1 gl 0.470, F*2 gl -0.761) with an F*1 gl standard deviation of 0.916 and an F*2 gl standard deviation of 0.577.

9 of 21 instances are less than one half standard deviation from the mean vowel center.

There are 8 instances between 0.5 and 1 standard deviations from the average vowel center:

IH0 V AY1 V AH0 : WELL THEY CALL IT REVIVAL NOW THEY CALLED IT

L L AY1 N D : MORNING EVERYBODY IN SCHOOL LINED UP SINGLE FILE AND

L F AY1 L AH0 : SCHOOL LINED UP SINGLE FILE AND MARCHED OVER TO

T K AY1 N D : WAS THAT CALLED WHAT KIND OF MEETING BETRACTED MEETING

R L AY1 K DH : WHERE ONCE A YEAR LIKE THIS CHURCH WOULD HAVE

IY0 N AY1 N TH : ORIGINALLY IN UH TWENTY NINE THIRTY AND THIRTY ONE

AH0 T AY1 M EH1 : AND THEN BY THE TIME EVERYBODY GOT EVERYTHING WHAT

AH0 M AY1 L F : ABOUT A HALF A MILE FROM SCHOOL WHAT IN

There are 4 instances between 1 and 1.5 standard deviations from the average vowel center:

EH1 N AY1 S T : BUT THEN SEE WHEN I STARTED SCHOOL IT WAS

IH0 V AY1 V AH0 : HAD WHAT THEY CALL REVIVAL WELL THEY CALL IT

IYO V AY1 V AHO : HAVE THEIRS THERE WERE REVIVALS GOING ON ALL SUMMER

AO1 S AY1 M IY1 : KING ROAD AND CROSS I MEAN WHERE HEARTS CRAB

Listing A.4: Output file for `informant_two` with Lobanov normalization

```
PhoSS, A Phonetic Summarizer for Sociolinguists
Comparative Summary
Lobanov

Speaker: informant_two
Vowel Set: ['IY1', 'UW1', 'AA1', 'AE1']
Diphthongs: ['AY1']
Audio File: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.wav
Transcript: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.txt
TextGrid: /home/justin/Dropbox/Thesis/testfiles/school_case/
   informant_two/informant_two.TextGrid

F*1 vowel space for informant_two has a range of 5.273 units, from
   -1.524 to 3.749.
F*2 vowel space for informant_two has a range of 4.431 units, from
   -1.928 to 2.503.

Summary for IY1
  F*1 vowel space for IY1 has a range of 2.128 units, from -1.490 to
      0.638.
  F*2 vowel space for IY1 has a range of 2.747 units, from -0.616 to
      2.131.
  IY1 for informant_two centers at (F*1 -0.934, F*2 1.094), which is
      0.099 standard deviations from the average F*1 and 0.178
     standard deviations from the average F*2.
```

The group average for IY1 occupies vowel space centered at (F*1 -0.896, F*2 0.956) with an F*1 standard deviation of 0.385 and an F*2 standard deviation of 0.775.

16 of 31 instances are less than one half standard deviation from the mean vowel center.

There are 12 instances between 0.5 and 1 standard deviations from the average vowel center:

R P IY1 P AH0 : STORING IT IN THERE PEOPLE JUST DIFFERENT PEOPLE HAD

AY1 M IY1 N AH1 : AS UH TEACHERS I MEAN UH CHILDREN BEING ANXIOUS

D W IY1 HH AE1 : ON THE BOARD AND WE HAD THIS SCHOOL AT

AH0 N IY1 Z ER0 : ANYBODY TO GO TO EBENEZER SCHOOL BECAUSE THEY WERE

OW1 SH IY1 K AO1 : OH MY GOODNESS SO SHE CALLED ME ON WE

D M IY1 AA1 N : GOODNESS SO SHE CALLED ME ON WE WENT TO

EY1 W IY1 L IH1 : AFTER CHURCH THAT DAY WE LIVED AT CRABAPPLE WE

L W IY1 W EH1 : WE LIVED AT CRABAPPLE WE WENT DOWN TO WALKED

TH M IY1 AH0 N : MY HUSBAND WENT WITH ME AND SEE I DIDN'T

D S IY1 AY1 D : WENT WITH ME AND SEE I DIDN'T KNOW I

S DH IY1 Z AH1 : KNOW HOW THAT WORKS THESE OTHER PEOPLE THEY WERE

NG W IY1 G AA1 : SO WE DID NOTHING WE GOT TO THE SCHOOL

There are 3 instances between 1 and 1.5 standard deviations from the average vowel center:

R W IY1 HH AE1 : IN THIS ROOM WHERE WE HAD TO CLEAR OUT

D W IY1 D IH1 : ANOTHER JOB SO AND WE DIDN'T HAVE A CAR

AH1 L IY1 V Z : THE CORN THAT THE LEAVES THAT'S NOT WHAT YOU

Summary for UW1

F*1 vowel space for UW1 has a range of 5.273 units, from -1.524 to 3.749.

F*2 vowel space for UW1 has a range of 3.921 units, from -1.418 to 2.503.

UW1 for informant_two centers at (F*1 -0.705, F*2 -0.122), which is 0.125 standard deviations from the average F*1 and 0.030 standard deviations from the average F*2.

The group average for UW1 occupies vowel space centered at (F*1 -0.573, F*2 -0.094) with an F*1 standard deviation of 1.056 and an F*2 standard deviation of 0.924.

10 of 19 instances are less than one half standard deviation from the mean vowel center.

There are 8 instances between 0.5 and 1 standard deviations from the average vowel center:

T Y UW1 K AO1 : HIM I FORGET WHAT YOU CALL THAT ANYWAY THAT

S R UW1 M HH : THAT ALL IN THIS ROOM WHERE WE HAD TO

S K UW1 L OW1 : DAY WE STARTED THE SCHOOL OH MY GOODNESS THAT'S

S K UW1 L HH : THE SUMMER THAT BUT SCHOOL HOUSE BUT UH THAT

OW1 T UW1 EH2 B : HAVE ANYBODY TO GO TO EBENEZER SCHOOL BECAUSE THEY

T T UW1 AH0 M : ME ON WE WENT TO A MEETING ON SATURDAY

ER0 N UW1 N AH0 : CALLED ME ON SATURDAY AFTERNOON AND WANTED TO KNOW

D Y UW1 N OW1 : SCHOOL IN CRABAPPLE AND YOU KNOW HOW THAT WORKS

There are 1 instances greater than 2.5 standard deviations from the average vowel center:

T Y UW1 IH2 M : LEAVES THAT'S NOT WHAT YOU HIM I FORGET WHAT

Summary for AA1

F*1 vowel space for AA1 has a range of 3.187 units, from -1.266 to 1.921.

F*2 vowel space for AA1 has a range of 3.449 units, from -1.928 to 1.521.

AA1 for informant_two centers at (F*1 0.343, F*2 -1.135), which is 0.157 standard deviations from the average F*1 and 0.175 standard deviations from the average F*2.

The group average for AA1 occupies vowel space centered at (F*1 0.468, F*2 -0.987) with an F*1 standard deviation of 0.796 and an F*2 standard deviation of 0.848.

13 of 18 instances are less than one half standard deviation from the mean vowel center.

There are 4 instances between 0.5 and 1 standard deviations from the average vowel center:

AW1 T AA1 N DH : HAD TO CLEAR OUT ON THE DAY WE STARTED

S T AA1 R T : ON THE DAY WE STARTED THE SCHOOL OH MY

D F AA1 D ER0 : THE SCHOOL BUILDING AND FODDER YOU KNOW WHAT FODDER

T F AA1 D ER0 : FODDER YOU KNOW WHAT FODDER IS THAT'S THE CORN

There are 1 instances between 1.5 and 2 standard deviations from the average vowel center:

S N AA1 T HH : THAT THE LEAVES THAT'S NOT WHAT YOU HIM I

Summary for AE1

F*1 vowel space for AE1 has a range of 3.280 units, from -0.795 to 2.485.

F*2 vowel space for AE1 has a range of 2.311 units, from -1.142 to 1.169.

AE1 for informant_two centers at (F*1 0.522, F*2 0.078), which is 0.319 standard deviations from the average F*1 and 0.269 standard deviations from the average F*2.

The group average for AE1 occupies vowel space centered at (F*1 0.258, F*2 0.259) with an F*1 standard deviation of 0.827 and an F*2 standard deviation of 0.673.

9 of 45 instances are less than one half standard deviation from the mean vowel center.

There are 9 instances between 0.5 and 1 standard deviations from the average vowel center:

IY1 HH AE1 D T : THIS ROOM WHERE WE HAD TO CLEAR OUT ON

S DH AE1 T S : SCHOOL OH MY GOODNESS THAT'S SO FUNNY THEY'D BEEN

L HH AE1 D Y : PEOPLE JUST DIFFERENT PEOPLE HAD USED THAT HOUSE DURING

D DH AE1 T HH : DIFFERENT PEOPLE HAD USED THAT HOUSE DURING THE SUMMER

AH1 DH AE1 T W : SCHOOL HOUSE BUT UH THAT WAS THAT WAS I'LL

AE1 D AE1 N D : GOSSIPING ABOUT HER MAD AND EVERYTHING SO UH LUVIT

K R AE1 B AE1 : WOULDN'T HAVE ANYBODY FROM CRABAPPLE CAUSE THEY WERE MAD

ER0 M AE1 D DH : CRABAPPLE CAUSE THEY WERE MAD THAT OH MY GOODNESS

T HH AE1 V AH0 : THE MONEY AND DIDN'T HAVE ANOTHER JOB SO AND

There are 15 instances between 1 and 1.5 standard deviations from the average vowel center:

L DH AE1 T EH1 : FORGET WHAT YOU CALL THAT ANYWAY THAT WAS HAY

EY2 DH AE1 T W : YOU CALL THAT ANYWAY THAT WAS HAY AND THAT

HH EY1 AE1 N D : ANYWAY THAT WAS HAY AND THAT ALL IN THIS

D DH AE1 T AO1 : THAT WAS HAY AND THAT ALL IN THIS ROOM

Z DH AE1 T W : BUT UH THAT WAS THAT WAS I'LL I'LL SAY

EY1 DH AE1 T W : WAS I'LL I'LL SAY THAT WAS ONE OF MY

AE1 B AE1 P AH0 : WOULDN'T HAVE ANYBODY FROM CRABAPPLE CAUSE THEY WERE MAD

N S AE1 T ER0 : TO A MEETING ON SATURDAY MORNING HE CALLED ME

K R AE1 B AE1 : GET THE SCHOOL IN CRABAPPLE AND YOU KNOW HOW

AE1 B AE1 P AH0 : GET THE SCHOOL IN CRABAPPLE AND YOU KNOW HOW

S OW1 AE1 N D : HAVE ANOTHER JOB SO AND WE DIDN'T HAVE A

OW1 DH AE1 T W : NINETEEN HUNDRED THIRTY SO THAT WAS UH DURING THE

AH0 N AE1 N D : UH DURING THE DEPRESSION AND UH WE WALKED SIX

L Z AE1 F T : WE WALKED SIX MILES AFTER CHURCH THAT DAY WE

CH DH AE1 T D : SIX MILES AFTER CHURCH THAT DAY WE LIVED AT

There are 8 instances between 1.5 and 2 standard deviations from the average vowel center:

Z DH AE1 T S : CORN THAT THE LEAVES THAT'S NOT WHAT YOU HIM

IH0 NG AE1 NG SH : MEAN UH CHILDREN BEING ANXIOUS TO LEARN
N S AE1 T IH2 : HE CALLED ME ON SATURDAY AFTERNOON AND WANTED TO
T HH AE1 V AH0 : SO AND WE DIDN'T HAVE A CAR WE WE
V D AE1 T K : THAT DAY WE LIVED AT CRABAPPLE WE WENT DOWN
K R AE1 B AE1 : DAY WE LIVED AT CRABAPPLE WE WENT DOWN TO
AE1 B AE1 P AH0 : DAY WE LIVED AT CRABAPPLE WE WENT DOWN TO
AY1 D AE1 D IY0 : MILES TO GET MY DADDY'S CAR SO I COULD

There are 2 instances between 2 and 2.5 standard deviations from the average vowel center:
AW1 DH AE1 T W : AND YOU KNOW HOW THAT WORKS THESE OTHER PEOPLE
IH0 NG AE1 N D : TO THE SCHOOL BUILDING AND FODDER YOU KNOW WHAT

There are 2 instances greater than 2.5 standard deviations from the average vowel center:
L S AE1 T ER0 : GOING OVER THERE UNTIL SATURDAY SO WE DID NOTHING
Z DH AE1 T S : KNOW WHAT FODDER IS THAT'S THE CORN THAT THE

For target one of AY1:
F*1 vowel space for AY1 has a range of 3.649 units, from -1.070 to 2.579.
F*2 vowel space for AY1 has a range of 1.891 units, from -1.478 to 0.413.
AY1 for informant_two centers at (F*1 0.466, F*2 -0.719), which is 0.004 standard deviations from the average F*1 and 0.073 standard deviations from the average F*2.
The group average for AY1 occupies vowel space centered at (F*1 0.470, F*2 -0.761) with an F*1 standard deviation of 0.916 and an F*2 standard deviation of 0.577.

8 of 22 instances are less than one half standard deviation from the mean vowel center.

There are 9 instances between 0.5 and 1 standard deviations from the average vowel center:

OW1 M AY1 G UH1 : STARTED THE SCHOOL OH MY GOODNESS THAT'S SO FUNNY

AY1 L AY1 L S : WAS THAT WAS I'LL I'LL SAY THAT WAS ONE

V M AY1 B EH1 : THAT WAS ONE OF MY BEST YEARS AS FAR

ER0 Z AY1 M IY1 : FAR AS UH TEACHERS I MEAN UH CHILDREN BEING

T R AY1 NG T : OF THE BIG ONES TRYING TO GET THE SCHOOL

T M AY1 D AE1 : SIX MILES TO GET MY DADDY'S CAR SO I

S OW1 AY1 K UH1 : MY DADDY'S CAR SO I COULD DRIVE THE NEXT

D R AY1 V DH : CAR SO I COULD DRIVE THE NEXT MORNING TO

L M AY1 HH AH1 : MORNING TO SCHOOL WELL MY HUSBAND WENT WITH ME

There are 4 instances between 1 and 1.5 standard deviations from the average vowel center:

X M AY1 P R : MY PRINCIPLE WAS ONE OF

IH2 M AY1 F ER0 : NOT WHAT YOU HIM I FORGET WHAT YOU CALL

AA1 Z AY1 L AY1 : THAT WAS THAT WAS I'LL I'LL SAY THAT WAS

S IY1 AY1 D IH1 : WITH ME AND SEE I DIDN'T KNOW I WAS

There are 1 instances between 1.5 and 2 standard deviations from the average vowel center:

N OW1 AY1 W AA1 : SEE I DIDN'T KNOW I WAS GOING OVER THERE

For target two of AY1:

F*1 gl vowel space for AY1 has a range of 3.434 units, from -1.067 to 2.367.

F*2 gl vowel space for AY1 has a range of 3.085 units, from -1.572 to 1.513.

AY1 for informant_two centers at (F*1 gl 0.466, F*2 gl -0.719), which is 0.004 standard deviations from the average F*1 gl and 0.073 standard deviations from the average F*2 gl.

The group average for AY1 occupies vowel space centered at (F*1 gl 0.470, F*2 gl -0.761) with an F*1 gl standard deviation of 0.916 and an F*2 gl standard deviation of 0.577.

8 of 22 instances are less than one half standard deviation from the mean vowel center.

There are 9 instances between 0.5 and 1 standard deviations from the average vowel center:

OW1 M AY1 G UH1 : STARTED THE SCHOOL OH MY GOODNESS THAT'S SO FUNNY

AY1 L AY1 L S : WAS THAT WAS I'LL I'LL SAY THAT WAS ONE

V M AY1 B EH1 : THAT WAS ONE OF MY BEST YEARS AS FAR

ER0 Z AY1 M IY1 : FAR AS UH TEACHERS I MEAN UH CHILDREN BEING

T R AY1 NG T : OF THE BIG ONES TRYING TO GET THE SCHOOL

T M AY1 D AE1 : SIX MILES TO GET MY DADDY'S CAR SO I

S OW1 AY1 K UH1 : MY DADDY'S CAR SO I COULD DRIVE THE NEXT

D R AY1 V DH : CAR SO I COULD DRIVE THE NEXT MORNING TO

L M AY1 HH AH1 : MORNING TO SCHOOL WELL MY HUSBAND WENT WITH ME

There are 4 instances between 1 and 1.5 standard deviations from the average vowel center:

```
X M AY1 P R  : MY PRINCIPLE WAS ONE OF
IH2 M AY1 F ER0  : NOT WHAT YOU HIM I FORGET WHAT YOU CALL
AA1 Z AY1 L AY1  : THAT WAS THAT WAS I'LL I'LL SAY THAT WAS
S IY1 AY1 D IH1  : WITH ME AND SEE I DIDN'T KNOW I WAS

There are 1 instances between 1.5 and 2 standard deviations from
   the average vowel center:
N OW1 AY1 W AA1  : SEE I DIDN'T KNOW I WAS GOING OVER THERE
```

Appendix B

PhoSS Source Code

Listing B.1: PhoSS Main Module

```
1 ## phoss.py
2 ## Copyright Justin Sperlein 2011
3 ## Written for partial completion of Master's degree
4
5 import os, sys, subprocess, fnmatch, pprint
6 import phossalignment, phossformantextraction, phossnorm,
      tgparser, phosssummary, phossoutput
7 from optparse import OptionParser
8
9
10 def main(argv):
11     usage = 'usage: %prog [-hdcmblnvwr] dirs "phone set"'
12     parser = OptionParser(usage = usage)
13     parser.add_option('-d', '--debug', action='store_true',
           dest="_debug", default='store_false', help='display
           debug output')
14     parser.add_option('-c', "--no-compare", action='store_true'
           , dest="_no_compare", default='store_false', help='
           perform only individual summaries for all informants,
           and no comparative summaries')
```

```
15    parser.add_option('-g', '--direct-comparison', action='
          store_true', dest='direct_compare', default='store_false
          ', help='directly compare two speakers or one speaker to
           a separate group')
16    parser.add_option('-p', '--dipthongs', type='string', dest=
          'diphthongs', default='', help='sample diphthongs one-
          third and two-thirds of the way through the vowel')
17    parser.add_option('-m', '--midpoints', action='store_true',
           dest='midpoints', default='store_false', help='sample
          all vowels at thier midpoints')
18    parser.add_option('-b', '--bark_difference', action = '
          store_true', dest='_bark_difference', default='
          store_false', help="normalize using Syrdal and Gopal's
          bark-difference method")
19    parser.add_option('-l', '--lobanov', action='store_true',
          dest='_lobanov', default='store_false', help='normalize
          using the Lobanov method')
20    parser.add_option('-n', '--nearey', action='store_true',
          dest='_nearey', default='store_false', help="normalize
          using Nearey's 1977 method")
21    parser.add_option('-v', '--labov', action='store_true',
          dest='_labov', default='store_false', help="normalize
          using Labov's revision of the Nearey method")
22    parser.add_option('-w', '--watt_and_fabricius', action='
          store_true', dest='_watt_and_fabricius', default='
          store_false', help='normalize using the Watt and
          Fabricius method')
```

```
23      parser.add_option('-r', '--raw', action='store_true', dest=
            '_raw', default='store_false', help='summarize vowels
            using no normalization.')
24      (options, args) = parser.parse_args()
25
26      if len(args) < 2:
27          print parser.print_help()
28          sys.exit(1)
29
30      # Sort speakers list. This comes in handy for putting
            context together with informant
31      # in the proper order.
32      speakerslist = args[:-1]
33      speakerslist = sorted(speakerslist)
34
35      # Since I've tripped over it so many times, I'm accepting
            comma separated vowel sets as well.
36      if ',' in args[-1]:
37          # Remove leading and trailing whitespace from vowels,
                then split on comma...
38          vowelset = [vow.strip() for vow in args[-1].split(',')]
39      else:
40          # ... or split on whitespace.
41          vowelset = args[-1].split()
42
43      # If the user command forgets the vowel set, then this will
            trigger the message.
```

```
44      if os.path.exists(vowelset[0]):
45          print 'Use a space-delimited vowel set as last argument
                ! Example: "AE1 AH0"'
46          sys.exit(2)
47
48      if ',' in options.diphthongs:
49          dips = options.diphthongs.split(',')
50      else:
51          dips = options.diphthongs.split()
52
53      # the user may select more than one normalization method.
54      # PhoSS performs summary with each method.
55      norm_options = phossnorm.norm_options_list(options)
56
57      # Initializing the Speakers object, which is the main data
            structure for PhoSS.
58      InformantList = Speakers()
59      # add each informant to speakerslist
60      for person in speakerslist:
61          InformantList.addSpeaker(person)
62
63      # add all vowels to the vowel set and diphthongs to their
            own list
64      for informant in InformantList.speakers:
65          informant.diphthongs = dips
66          for vowel in vowelset:
67              informant.addVowel(vowel)
```

```
68
69      if options._debug == True:
70          print 'options ', options
71          print 'args ', args
72          print 'Informant Objects: ', InformantList.speakers
73          print 'Vowel Set: ', InformantList.speakers[0].VowelSet
74          pprint.pprint( \
75          [('Directory: ' + each.informantdir, each.FileGroup)
                for each in InformantList.speakers] )
76
77      # Check that there are the minimum required files in the
            informant directories.
78      for informant in InformantList.speakers:
79          if informant.audio_file == '':
80              print 'Missing audio in ', informant.informantdir
81              print 'Make sure .wav audio file is in %s and try
                    again.' % informant.informantdir
82              sys.exit(1)
83          elif informant.transcript_file == '':
84              print 'Missing transcript in ', informant.
                    informantdir
85              print 'Make sure .txt transcript is in %s and try
                    again.' % informant.informantdir
86              sys.exit(1)
87
88
89      # ALIGNMENT
```

```
90      # Check for TextGrid for each speaker. Prompt for alignment
            if it is not there.
91      for informant in InformantList.speakers:
92          if informant.TextGrid == '':
93              print "No TextGrid found in ", informant.
                    informantdir
94              ans_align = raw_input("Run alignment on selected
                    audio and transcript? (y/n) ")
95              if ans_align.lower().strip() == "y" or ans_align.
                    lower().strip() == "yes":
96                  # add textgrid after alignment
97                  informant.TextGrid = phossalignment.
                        run_alignment(informant, InformantList)
98              elif ans_align.lower().strip() == "n" or ans_align.
                    lower().strip() == "no":
99                  print usage
100                 sys.exit(1)
101             else:
102                 print "Not a valid option"
103                 print usage
104                 sys.exit(2)
105         elif not informant.TextGrid == '':
106             print 'Alignment previously completed for %s ' %
                    informant.name
107
108
109     # FORMANT EXTRACTION
```

```
110     # Write a praat script for each speaker, and run it.
111     # Merge result files.
112     for informant in InformantList.speakers:
113         phossformantextraction.write_praat_script(informant,
                options)
114         print "Wrote script for", informant.name
115
116     for informant in InformantList.speakers:
117         if os.path.exists(informant.result_file):
118             print "A result file exists for %s" % informant.
                    name
119             ans_process = raw_input("Execute Praat script to
                    overwrite existing? (y/n) ")
120             if ans_process.lower().strip() == "y" or
                    ans_process.lower().strip() == "yes":
121                 print ("Executing praat script for %s because
                        you said yes." % informant.name)
122                 phossformantextraction.run_praat_script(
                        informant)
123                 # At this point, the result file has the
                        necessary informaion
124                 # to be used with NORM, but the file needs to
                        be sorted.
125                 # Tokens of like types must be together, so
                        they are sorted thus.
126             elif ans_process.lower().strip() == "n" or
                    ans_process.lower().strip() == "no":
```

```
127                 pass
128             else:
129                 print "Not a valid option."
130                 sys.exit(2)
131         elif not os.path.exists(informant.result_file):
132             print "Executing Praat script for %s because result
                    file does not exist." % informant.name
133             phossformantextraction.run_praat_script(informant)
134             # At this point, the result file has the necessary
                    informaion
135             # to be used with NORM, but the file needs to be
                    sorted.
136             # Tokens of like types must be together, so they
                    are sorted thus.
137
138     # Sort the results by vowel by speaker
139     for informant in InformantList.speakers:
140         phossformantextraction.sort_file(informant)
141
142     # Merge the result files and write a copy to each informant
            directory
143     if options._no_compare == True:
144         pass
145     else:
146         phossformantextraction.merge_result_files(InformantList
                )
147
```

```
148     # Create a data_frame for each informant. At this point, it
            is unnormalized
149     for informant in InformantList.speakers:
150         phossnorm.import_df(informant, options)
151
152         if options._debug == True:
153             #change this so it's the individual data frame
154             # gets changed in norm
155             print informant.data_frame
156
157     # TEXTGRID PARSER
158     # Add Contexts to dataframe by creating a factor vector
159     # for the speaker from a list of contexts.
160     # Add the diphthongs to the VowelSet to get correct context
            .
161     for informant in InformantList.speakers:
162         informant.VowelSet += informant.diphthongs
163         tgparser.get_informant_context(informant, options)
164
165     # contexts are stored separately for each informant, so I
            need another function
166     # to combine them, and another to insert the context into
            the dataframes.
167         if options._debug == True:
168             print len(informant.context_list)
169             print informant.context_list
170
```

```
171     # Note to self: This function should be indented this way
172     # because of how PhoSS builds the context vector. It is not
            a mistake.
173     tgparser.insert_context(InformantList)
174     # Dataframe is complete with unnormalized values and with
            context.
175
176     if options._debug == True:
177         for informant in InformantList.speakers:
178             print informant.data_frame
179
180     #phossoutput.output(informant)
181     if norm_options == []:
182         norm_options.append('No Normalization')
183
184     for norm_opt in norm_options:
185         for informant in InformantList.speakers:
186             # NORMALIZATION
187             # Apply normalization routines, if any
188             phossnorm.normalize(norm_opt, informant)
189             # ... and extract subdataframes.
190             phossnorm.extract_sub_data_frame(informant,
                    norm_opt)
191
192         # OUTPUT
193         #
194         phossoutput.output(InformantList, options, norm_opt)
```

```
195
196
197 ########################
198 #End PhoSS Main
199 ########################
200 #PhoSS Class definitions
201 ########################
202 # The Speakers object is instantiated as InformantList throught
        the main loop.
203 class Speakers(object):
204
205     def __init__(self):
206         self.speakers = []
207         self.p2falocation = ''
208
209     # Speakers class methods
210     def addSpeaker(self,informantdir):
211         self.informantdir = os.path.abspath(informantdir)
212         self.speakers.append(Informant(informantdir))
213
214     def informants(self):
215         print self.__name__.speakers
216
217
218 # Informant objects are elements of the list at Speakers.
      speakers
219 class Informant(object):
```

```
220     def __init__(self,informantdir):
221
222         self.VowelSet = []
223         self.diphthongs = []
224
225         # All files created or used throughout the program.
226         self.informantdir = informantdir
227         self.phossfilesdir = informantdir + '/phoss/'
228         self.audio_file = ''
229         self.transcript_file = ''
230         self.TextGrid = ''
231         self.praat_file = ''
232         self.result_file = ''
233         self.common_result_file = ''
234         self.output_file = ''
235
236         for file in os.listdir(self.informantdir):
237             if fnmatch.fnmatch(file, '*.txt'):
238                 self.transcript_file = os.path.join(os.path.
                        abspath(self.informantdir), file)
239             elif fnmatch.fnmatch(file, '*.wav'):
240                 self.audio_file = os.path.join(os.path.abspath(
                        self.informantdir), file)
241             elif fnmatch.fnmatch(file, '*.TextGrid'):
242                 self.TextGrid = os.path.join(os.path.abspath(
                        self.informantdir), file)
243             elif fnmatch.fnmatch(file, '*.praat'):
```

```
244                 self.praat_file = os.path.join(os.path.abspath(
                        self.informantdir), file)
245             elif fnmatch.fnmatch(file, '*.result'):
246                 self.result_file = os.path.join(os.path.abspath
                        (self.informantdir), file)
247             elif fnmatch.fnmatch(file, '*.commonresult'):
248                 self.common_result_file = os.path.join(os.path.
                        abspath(self.informantdir), file)
249             elif fnmatch.fnmatch(file, '*.phoss'):
250                 self.output_file = os.path.join(os.path.abspath
                        (self.informantdir), file)
251
252         self.FileGroup = (self.audio_file, self.transcript_file
                , \
253         self.TextGrid, self.praat_file, self.result_file, \
254         self.output_file, self.common_result_file)
255
256         ############beyond the files portion############
257         # The name is what R uses to identify the speaker.
258         self.name = os.path.splitext(os.path.basename(self.
                audio_file))[0]
259         # R data frames
260         self.data_frame = ''
261         self.normed_data_frame = ''
262         self.sub_data_frame = ''
263
264         # Lists used while reconstructing context
```

```
265             self.phone_list = []
266             self.word_list = []
267             self.context_list = []
268
269         # Informant class methods
270         def addVowel(self,vowel):
271             self.vowel = vowel
272             self.VowelSet.append(vowel)
273
274         def addDiphthong(self,diphthong):
275             self.diphthong = diphthong
276             self.diphthongs.append(diphthong)
277
278         def pListInstance(self,phone,start,end,index,context=''):
279             self.phone_list.append(pList(phone,start,end,index,
                    context=''))
280
281         def wListInstance(self,word,start,end,index,phonerange=[]):
282             self.word_list.append(wList(word,start,end,index,
                    phonerange=[]))
283
284         def addContext(self,phone_context_string,
                word_context_string):
285             self.context_list.append((phone_context_string +' : '+
                    word_context_string))
286
287 class pList:
```

```
288     def __init__(self,phone,start,end,index,context=''):
289         self.phone = phone
290         self.start = start
291         self.end = end
292         self.index = index
293         self.sorted_index = ''
294         self.context = context
295
296 class wList:
297     def __init__(self,word,start,end,index,phonerange=[]):
298         self.word = word
299         self.start = start
300         self.end = end
301         self.index = index
302         self.phonerange = phonerange
303
304 ################
305
306 if __name__ == "__main__":
307     main(sys.argv[1:])
```

Listing B.2: Alignment Module

```
1 import os, subprocess
2
3 def run_alignment(informant, InformantList):
4     # locate p2fa align.py
5     # run a subprocess to do the alignment.
6     # name the TextGrid the same thing as the audio
7     # create a dictionary log for later additions
8
9     # check that the TextGrid I'm trying to make doesn't exist.
10    informant.TextGrid = os.path.splitext(informant.audio_file)
          [0] + '.TextGrid'
11
12    # save the p2fa installation path once you find it because
13    # you only need to find it once.
14    if InformantList.p2falocation == '':
15        print "Searching for p2fa/align.py..."
16        # find the p2fa/align.py script.
17        # Add some checks just in case it's not in /usr/
18        find_args = ['find', '/usr/local/p2fa/', '-name', '
              align.py']
19        p = subprocess.Popen(find_args, stdout=subprocess.PIPE)
20        align_path = p.communicate()[0].strip() # align.py
              absolute path
21        InformantList.p2falocation = align_path
22        print "Found p2fa/align.py..."
23    else:
```

```
24          align_path = InformantList.p2falocation
25      # the call path is what
26      call_path = "python " + \
27 align_path + " " + \
28 informant.audio_file + " " + \
29 informant.transcript_file + " " + \
30 informant.TextGrid
31
32      print "Calling p2fa with path: %s " % call_path + "\n"
33
34      os.system("python " + \
35 align_path + " " + \
36 informant.audio_file + " " + \
37 informant.transcript_file + " " + \
38 informant.TextGrid)
39
40      print "Alignment complete for %s" % informant.name
41      return
```

Listing B.3: Formant Extraction Module

```
1 import fileinput, os, sys, subprocess, pprint
2
3 def main(informant, options):
4     vowelset = informant.VowelSet
5     dir,name = os.path.split(informant.audio_file)
6     dir = os.path.relpath(dir)
7
8     name_sans_ext = os.path.splitext(name)[0]
9     praat_script = []
10
11    praat_script.append('''name$ = "%s"\n''' % name_sans_ext)
12    praat_script.append('''dir$ = "../../%s"\n''' % dir)
13    #Specify the name of the output file
14    praat_script.append('''outfile$ = name$ + ".result"\n''')
15    #If the output file already exists, delete it
16    praat_script.append('''filedelete 'dir$'/'outfile$'\n''')
17    #In output file, add a line with name, duration, F1, F2
          values
18    praat_script.append('''fileappend 'dir$'/'outfile$' speaker
          'tab$'vowel/frame'tab$'context'tab$'F1'tab$'F2'tab$'F3'
          tab$'gl F1'tab$'gl F2'tab$'gl F3'newline$'\n''')
19    praat_script.append('''Read from file... 'dir$'/'name$'.wav
          \n''')
20    praat_script.append('''select Sound 'name$'\n''')
21    praat_script.append('''To Formant (burg)... 0.01 5 5500
          0.025 50\n''')
```

```
22      praat_script.append('''Read from file... 'dir$'/'name$'.
            TextGrid\n''')
23      praat_script.append('''nInterv = Get number of intervals...
            1\n''')
24      praat_script.append('''for j from 1 to 'nInterv'\n''')
25      praat_script.append('''select TextGrid 'name$'\n''')
26      praat_script.append('''lab$ = Get label of interval... 1 'j
            '\n''' )
27
28      for n in xrange(len(vowelset)):
29          if options.midpoints == True:
30              sampletype = "midpoint"
31              formula = "(beg + end) / 2"
32              if n == 0:
33                  praat_script.append('''if lab$ = "%s"\n''' %
                        vowelset[n])
34              else:
35                  praat_script.append('''elif lab$ = "%s"\n''' %
                        vowelset[n])
36
37              praat_script.append('''\tbeg = Get starting point
                    ... 1 'j'\n''')
38              praat_script.append('''\tend = Get end point... 1 '
                    j'\n''')
39              praat_script.append('''\t%s = %s\n''' % (sampletype
                    ,formula))
```

```
40              praat_script.append('''\tselect Formant 'name$'\n
                    ''')
41              praat_script.append('''\tf1 = Get value at time...
                    1 '%s' Hertz Linear\n''' % sampletype)
42              praat_script.append('''\tf2 = Get value at time...
                    2 '%s' Hertz Linear\n''' % sampletype)
43              praat_script.append('''\tf3 = Get value at time...
                    3 '%s' Hertz Linear\n''' % sampletype)
44              praat_script.append('''\tfileappend 'dir$'/''
                    outfile$' 'name$''tab$''lab$''tab$''tab$''f1''
                    tab$''f2''tab$''f3''tab$''tab$''tab$''tab$''tab$
                    ''tab$''newline$'\n''')
45
46          else:
47              sampletype = "thirdpoint"
48              formula = "beg + ((end - beg) / 3)"
49              if n == 0:
50                  praat_script.append('''if lab$ = "%s"\n''' %
                        vowelset[n])
51              else:
52                  praat_script.append('''elif lab$ = "%s"\n''' %
                        vowelset[n])
53
54              praat_script.append('''\tbeg = Get starting point
                    ... 1 'j'\n''')
55              praat_script.append('''\tend = Get end point... 1 '
                    j'\n''')
```

```
56          praat_script.append('''\t%s = %s\n''' % (sampletype
                ,formula))
57          praat_script.append('''\tselect Formant 'name$'\n
                ''')
58          praat_script.append('''\tf1 = Get value at time...
                1 '%s' Hertz Linear\n''' % sampletype)
59          praat_script.append('''\tf2 = Get value at time...
                2 '%s' Hertz Linear\n''' % sampletype)
60          praat_script.append('''\tf3 = Get value at time...
                3 '%s' Hertz Linear\n''' % sampletype)
61          praat_script.append('''\tfileappend 'dir$'/'
                outfile$' 'name$''tab$''lab$''tab$''tab$''f1''
                tab$''f2''tab$''f3''tab$''tab$''tab$''tab$''tab$
                ''tab$''newline$'\n''')
62
63      # This includes dipthongs in the set of phones to
            sample
64      for j in xrange(len(informant.diphthongs)):
65          sampletype1 = "fifteenpercent"
66          sampletype2 = "eightypercent"
67          formula1 = "beg + ((end - beg) * 0.15)"
68          formula2 = "beg + ((end - beg) * 0.80)"
69          n = len(informant.VowelSet)
70          if n == 0:
71              praat_script.append('''if lab$ = "%s"\n''' %
                    informant.diphthongs[j])
72          else:
```

```
73                praat_script.append('''elif lab$ = "%s"\n''' %
                      informant.diphthongs[j])
74
75            praat_script.append('''\tbeg = Get starting point
                  ... 1 'j'\n''')
76            praat_script.append('''\tend = Get end point... 1 '
                  j'\n''')
77            praat_script.append('''\t%s = %s\n''' % (
                  sampletype1,formula1))
78            praat_script.append('''\t%s = %s\n''' % (
                  sampletype2,formula2))
79            praat_script.append('''\tselect Formant 'name$'\n
                  ''')
80            praat_script.append('''\tf1 = Get value at time...
                  1 '%s' Hertz Linear\n''' % sampletype1)
81            praat_script.append('''\tf2 = Get value at time...
                  2 '%s' Hertz Linear\n''' % sampletype1)
82            praat_script.append('''\tf3 = Get value at time...
                  3 '%s' Hertz Linear\n''' % sampletype1)
83            praat_script.append('''\tf1_g = Get value at time
                  ... 1 '%s' Hertz Linear\n''' % sampletype2)
84            praat_script.append('''\tf2_g = Get value at time
                  ... 2 '%s' Hertz Linear\n''' % sampletype2)
85            praat_script.append('''\tf3_g = Get value at time
                  ... 3 '%s' Hertz Linear\n''' % sampletype2)
86            praat_script.append('''\tfileappend 'dir$'/'
                  outfile$' 'name$''tab$''lab$''tab$''tab$''f1''
```

```
                tab$''f2''tab$''f3''tab$''f1_g''tab$''f2_g''tab$
                ''f3_g''newline$''\n''')
87
88      praat_script.append('''endif\n''')
89      praat_script.append('''endfor\n''')
90      praat_script.append('''select TextGrid 'name$'\n''')
91      praat_script.append('''plus Sound 'name$'\n''')
92      praat_script.append('''plus Formant 'name$'\n''')
93      praat_script.append('''Remove\n''')
94
95      if options._debug == True:
96          pprint.pprint(praat_script)
97
98      return praat_script
99
100 ## sort_file Code adapted from: http://code.activestate.com/
      recipes/440612/
101 ## Adaptation by Justin Sperlein, 2011
102 def sort_file(informant):
103     lines=[] # give lines variable a type of list
104     for line in fileinput.FileInput(informant.result_file,
          inplace=1):
105         lines.append(line.rstrip())
106     pre_sorted = lines
107     first = [lines[0]] # keep the header line separate from the
          data lines
108     rest = lines[1:]
```

```
109     rest.sort()
110     for each in rest:
111         first.append(each)
112     f = open(informant.result_file,"w")
113     for line in first:
114         f.write(line + '\n')
115     return
116
117 def write_praat_script(informant, options):
118     # Fill in the praat_file if it doesn't exist yet
119     informant.praat_file = os.path.splitext(informant.
            audio_file)[0] + ".praat"
120     f = open(informant.praat_file, 'w')
121     f.writelines(main(informant, options))
122     f.close()
123     informant.result_file = os.path.splitext(informant.
            audio_file)[0] + ".result"
124     return
125
126 def run_praat_script(informant):
127     if sys.platform == 'cygwin':
128         subprocess.Popen(['praatcon', informant.praat_file]).
                wait()
129     elif sys.platform == 'linux2':
130         subprocess.Popen(['praat', informant.praat_file]).wait
                ()
131
```

```
132 def merge_result_files(InformantList):
133     list_of_result_files = []
134     for informant in InformantList.speakers:
135         list_of_result_files.append(informant.result_file)
136
137     # Read whole of first file
138
139     f = open(list_of_result_files[0], 'r')
140     common_result = f.readlines()
141     f.close()
142
143     for file in list_of_result_files[1:]:
144         f = open(file, 'r')
145         common_append = f.readlines()[1:]
146         for line in common_append:
147             common_result.append(line)
148
149     for informant in InformantList.speakers:
150         informant.common_result_file = os.path.splitext(
              informant.audio_file)[0] + '.commonresult'
151         f = open(informant.common_result_file, 'w')
152         f.writelines(common_result)
153         f.close()
154     # I am writing the merged file to both directories because
        I thought writing to pwd was weird.
155     # Also, there is no hierarchy or preference shown to a
        member of an informant group.
```

Listing B.4: TextGrid Parser

```
1 import os, sys, pprint
2 import rpy2.robjects as robjects
3
4 # TextGrid parser
5 # perhaps making a class definition would be best, but I'm not
      that good with classes. They're new to me
6 # I'm going to use a 'readlines' approach
7 def main(informant):
8     f = open(informant.TextGrid)
9     g = f.readlines()
10    f.close()
11
12    g = remove_textgrid_header(g)
13    phone_tier = separate_phone_tier(g)
14    #print phone_tier
15    word_tier = separate_word_tier(g)
16    #print word_tier
17    clean_strings(phone_tier)
18    clean_strings(word_tier)
19
20    populate_phone_list(informant, phone_tier)
21    populate_word_list(informant, word_tier)
22
23    match_words_with_phones(informant)
24    only_phones = list_phones_only(informant) # to more easily
         get the context of phones.
```

```
25      fill_phone_context(informant,only_phones)#completes filling
            out all of the class characteristics
26      #see_word_and_pron(informant)
27      return
28
29 # clean up the strings
30 def clean_strings(tier):
31      for each in xrange(len(tier)):
32          tier[each] = tier[each].strip()
33          tier[each] = tier[each].strip('"')
34      return
35
36      # loop to get the pertinent info for the phones
37      # phone,start,end,index,context
38
39 def populate_phone_list(informant,phone_tier):
40      i=0
41      while phone_tier:
42          if phone_tier[2] in ['sp']:
43              pass
44          else:
45              informant.pListInstance(phone_tier[2], phone_tier
                    [0], phone_tier[1], i)
46              i = i + 1
47          phone_tier = phone_tier[3:]
48      return
49
```

```
50 def populate_word_list(informant, word_tier):
51     i=0
52     while word_tier:
53         if word_tier[2] in ['sp']:
54             pass
55         else:
56             informant.wListInstance(word_tier[2], word_tier[0],
                   word_tier[1], i)
57             i = i + 1
58         word_tier = word_tier[3:]
59     return
60
61 def remove_textgrid_header(g):
62     return g[12:] # real dirty, but I know the format of the
          TextGrid the p2fa puts out
63     #print g[0:3] # and the phone start times begin at line 13
64
65     # separate into the phone tier
66 def separate_phone_tier(g):
67     word_tier_start = g.index('"IntervalTier"\n')
68     phone_tier = g[:word_tier_start]
69     # print len(phone_tier)
70     # print g[:len(phone_tier)]
71     return phone_tier
72
73 def remove_secondary_intervaltier_header(w_tier):
74     return w_tier[5:]
```

```
75
76      # separate out the word tier
77  def separate_word_tier(g):
78      word_tier_start = g.index('"IntervalTier"\n')
79      word_tier = g[word_tier_start:]
80      word_tier = remove_secondary_intervaltier_header(word_tier)
81      # print word_tier
82      return word_tier
83
84  def match_words_with_phones(informant):
85      # match the start and end times for phones with those of
            words, thus getting the indices of phones per word
86      a = 0
87      b = 0
88      phoneoffset = 0
89      #print [phone.index for phone in informant.phone_list]
90  #   print len(informant.phone_list)
91      for word in informant.word_list:
92          #print word.word
93          for phone in informant.phone_list:
94
95              #if word.start == phone.start:
96              #   a = phone.index
97              if word.end == phone.end:
98                  a = phoneoffset
99                  b = phone.index + 1
```

```
100                 # because python slices go up to but not
                        including the final index of a slice
101             word.phonerange = (a,b)
102             phoneoffset = b
103             if b < a:
104                 print "WHOA! Theres something wrong here!"
105                 print word.word
106                 print word.start, phone.start
107                 print word.end, phone.end
108                 print a, b
109                 sys.exit(-1)
110 #             wrd = informant.phone_list[word.phonerange[0]:word.
    phonerange[1]]
111 #         print (word.word,word.phonerange[0],[each.phone for
    each in wrd],word.phonerange[1])
112     #print [(each.word,each.phonerange) for each in word_list]
113     return
114
115 def list_phones_only(informant):
116     only_phones = [each.phone for each in informant.phone_list]
117     for i in range(2):
118         only_phones.insert(0,'X')
119         only_phones.append('X')
120     return only_phones
121
122 def fill_phone_context(informant,only_phones):
123     #print phone_list
```

```
124     for each in informant.phone_list:
125         i = 0
126         j = 5
127         pwindow = only_phones[i:j] # context window
128         each.context = pwindow
129         del only_phones[0] #
130     return
131
132 def see_word_and_pron(informant):
133     for word in informant.word_list:
134         wrd = informant.phone_list[word.phonerange[0]:word.
              phonerange[1]]
135         print (word.word,word.phonerange[0],[each.phone for
              each in wrd],word.phonerange[1])
136
137 def get_informant_context(informant,options):
138     main(informant)
139     #pprint.pprint([(each.phone,each.end,each.index) for each
          in informant.phone_list if each.phone in informant.
          VowelSet])
140     i = 0
141     unsorted = []
142     for each in informant.phone_list:
143         if each.phone in informant.VowelSet:
144             # unsorted.append((each.phone, each.end, each.index
                  , each.context, i))
```

```
145             unsorted.append((each.phone, each.end, each.index,
                    i))
146             i += 1
147     # unsorted = [(each.phone,each.end,each.index) for each in
            informant.phone_list if each.phone in informant.VowelSet
            ]
148     sortd = unsorted[:]
149     sortd.sort()
150     #pprint.pprint(unsorted)
151     #pprint.pprint(sortd)
152     # make an unsorting key that contains 2-tuples of the
            indices for (unsorted,sorted)
153     decoder = []
154     informant.context_list = []
155     sortedrow = 0
156     for each in sortd:
157         # the first is the phone_list.index, and the second is
                the sorted element index
158         decoder.append((informant.phone_list[each[-2]].index,
                sortedrow))
159         sortedrow += 1
160         # print (informant.phone_list[each[-3]].phone, each[0])
161     #pprint.pprint(decoder)
162     #print len(decoder)
163     # the decoder is matching up the correct elements.
164
165     for (unpl, so) in decoder:
```

```
166         phone_context = informant.phone_list[unpl].context
167         phoneindex = informant.phone_list[unpl].index
168         word_context_string = ''
169         phone_context_string = ''
170         for word in informant.word_list:
171             if phoneindex in range(word.phonerange[0],word.
                phonerange[1]):
172                 if word.index - 4 < 0:
173                     word_context = [word.word for word in
                        informant.word_list[:word.index+5]]
174                 elif (word.index + 4) > len(informant.word_list
                    ):
175                     word_context = [word.word for word in
                        informant.word_list[word.index-4:]]
176                 else:
177                     word_context = [word.word for word in
                        informant.word_list[word.index-4:word.
                        index+5]]
178                 for elt in phone_context:
179                     phone_context_string += elt + ' '
180                 for elt in word_context:
181                     word_context_string += elt + ' '
182                 informant.addContext(phone_context_string,
                    word_context_string)
183     return
184
185 def insert_context(InformantList):
```

```
186     contextvector = []
187     # Concatenate the context lists for the informants.
188     for informant in InformantList.speakers:
189         contextvector += informant.context_list
190     # Convert that list into an R String Vector
191     # (StrVectors get converted to Factor vectors in data
            frames by rpy2)
192     contextfactorvector = robjects.StrVector(contextvector).
            factor()
193     # Insert context into each informant's data_frame
194     for informant in InformantList.speakers:
195         informant.data_frame[2] = contextfactorvector
196
197 if __name__ == "__main__":
198     main(sys.argv[-1]) # when called as stand alone with only
            the TextGrid as argument
```

Listing B.5: Normalization Module

```
1 import rpy2.robjects as robjects
2 from rpy2.robjects.packages import importr
3 V = importr('vowels')
4
5 def import_df(informant, options):
6     if options._no_compare == True:
7         informant.data_frame = V.load_vowels(informant.
              result_file)
8     else:
9         informant.data_frame = V.load_vowels(informant.
              common_result_file)
10    return
11
12 def norm_options_list(options):
13    norm_options = []
14    if options._bark_difference == True:
15        norm_options.append('Bark Difference')
16    if options._nearey == True:
17        norm_options.append('Nearey')
18    if options._labov == True:
19        norm_options.append('Labov')
20    if options._watt_and_fabricius == True:
21        norm_options.append('Watt and Fabricius')
22    if options._lobanov == True:
23        norm_options.append('Lobanov')
24    if options._raw == True:
```

```
25          norm_options.append('No Normalization')
26      return norm_options
27
28  def normalize(norm_opt, informant):
29      #print norm_opt
30      if norm_opt == 'Bark Difference':
31          #print 'Normalizing for %s using the %s method ...' % (
                informant.name, norm_opt)
32          normed_results = V.norm_bark(informant.data_frame)
33
34      elif norm_opt == 'Nearey':
35          #print 'Normalizing for %s using the %s method ...' % (
                informant.name, norm_opt)
36          normed_results = V.norm_nearey(informant.data_frame,
                formant_int=True, use_f3=False)
37
38      elif norm_opt == 'Labov':
39          #print 'Normalizing for %s using the %s method ...' % (
                informant.name, norm_opt)
40          normed_results = V.norm_labov(informant.data_frame,
                use_f3=False, geomean=True)
41
42      elif norm_opt == 'Watt and Fabricius':
43          #print 'Normalizing for %s using the %s method ...' % (
                informant.name, norm_opt)
44          normed_results = V.norm_wattfabricius(informant.
                data_frame, norm_means=False, mod_WF=False)
```

```
45
46      elif norm_opt == 'Lobanov':
47          #print 'Normalizing for %s using the %s method ...' % (
                informant.name, norm_opt)
48          normed_results = V.norm_lobanov(informant.data_frame)
49
50      elif norm_opt == 'No Normalization' or norm_opt == '':
51          #print '%s used for %s summary.' % (norm_opt, informant
                .name)
52          normed_results = informant.data_frame
53
54      informant.normed_data_frame = normed_results
55      return
56
57 def extract_sub_data_frame(informant, norm_opt):
58      if norm_opt == 'Bark Difference' \
59 or norm_opt == 'Lobanov' \
60 or norm_opt == 'Labov' \
61 or norm_opt == 'Watt and Fabricius' \
62 or norm_opt == 'Nearey':
63          speaker = "Speaker"
64          vowelframe = "Vowel"
65      else:
66          speaker = "speaker"
67          vowelframe = "vowel.frame"
```

```
68      informant.sub_data_frame = informant.normed_data_frame.rx(
            informant.normed_data_frame.rx2(speaker).ro == informant
            .name, True)
69      #print informant.sub_data_frame
70
71
72      informant.sub_data_frame.to_csvfile(informant.informantdir+
            'subdataframe.csv',sep='\t',row_names=False)
73      informant.normed_data_frame.to_csvfile(informant.
            informantdir+'dataframe.csv',sep='\t',row_names=False)
74      return
```

Listing B.6: Output Module

```
1 import os, phosssummary
2
3 def output(InformantList, options, norm_opt):
4     # lower() and ''.join() words from norm_opt to make a norm
          file extension
5     nopt = ''.join(norm_opt.lower().split())
6     norm_ext = '.' + nopt
7     for informant in InformantList.speakers:
8         # add normalization extention in the case of multiple
              normalizations
9         informant.output_file = os.path.splitext(informant.
              audio_file)[0] + norm_ext + '.phoss'
10
11    if len(InformantList.speakers) > 1:
12        # write individual output for all informants if you don
              't want to compare them
13        if options._no_compare == True:
14            for informant in InformantList.speakers:
15                iSummary(informant, options, norm_opt)
16        # or you make a direct comparison, so the informant is
              not in the group
17        elif options.direct_compare == True:
18            gSummary(InformantList, options, norm_opt)
19        # otherwise do a comparison of the informant to the
              group.
20        else:
```

```
21              cSummary(InformantList, options, norm_opt)
22      # Write individual output if there's just one.
23      else:
24          iSummary(InformantList.speakers[0], options, norm_opt)
25
26
27  def iSummary(informant, options, norm_opt):
28      # The normal header and informant information
29      phosssummary.write_header(informant, "Individual Summary",
            norm_opt)
30      phosssummary.informant_information(informant)
31      # Absolute vowel space
32      flabel1, flabel2 = tuple(informant.normed_data_frame.
            colnames)[3:5]
33      phosssummary.absolute_vowel_space(informant, flabel1,
            norm_opt)
34      phosssummary.absolute_vowel_space(informant, flabel2,
            norm_opt)
35      # Vowel-by-vowel summary
36      for vowel in informant.VowelSet:
37          phosssummary.vowel_header(informant, vowel)
38          # The F1 and F2 column names change by normalization
                method. This keeps it straight
39          flabel1, flabel2 = tuple(informant.normed_data_frame.
                colnames)[3:5]
40
41          # what are the maximum and minimum values for f1 and f2
```

```
42          phosssummary.max_and_min_f_values(informant, vowel,
                flabel1, norm_opt)
43          phosssummary.max_and_min_f_values(informant, vowel,
                flabel2, norm_opt)
44          # the group average for vowel X occupies the vowel
                space centered here and ranging from X to Y.
45          phosssummary.indiv_vowel_space(informant, vowel,
                flabel1, flabel2, norm_opt)
46          phosssummary.indiv_euc(informant, vowel, flabel1,
                flabel2, norm_opt)
47
48          # note to self: just remember, you have to do
                everything double.
49      for diphthong in informant.diphthongs:
50          target_one_f1, target_one_f2 = tuple(informant.
                normed_data_frame.colnames)[3:5]
51          target_two_f1_gl, target_two_f2_gl = tuple(informant.
                normed_data_frame.colnames)[6:8]
52          # Summarize first vowel target
53          phosssummary.max_and_min_f_values(informant, diphthong,
                 target_one_f1, norm_opt, dtarg="one")
54          phosssummary.max_and_min_f_values(informant, diphthong,
                 target_one_f2, norm_opt,)
55          phosssummary.indiv_vowel_space(informant, diphthong,
                target_one_f1, target_one_f2, norm_opt, dtarg="one")
56          phosssummary.indiv_euc(informant, diphthong,
                target_one_f1, target_one_f2, norm_opt, dtarg="one")
```

```
57          # Summarize second diphthong target
58          phosssummary.max_and_min_f_values(informant, diphthong,
                target_two_f1_gl, norm_opt, dtarg="two")
59          phosssummary.max_and_min_f_values(informant, diphthong,
                target_two_f2_gl, norm_opt,)
60          phosssummary.indiv_vowel_space(informant, diphthong,
               target_two_f1_gl, target_two_f2_gl, norm_opt, dtarg=
               "two")
61          phosssummary.indiv_euc(informant, diphthong,
               target_two_f1_gl, target_two_f2_gl, norm_opt, dtarg=
               "two")
62          # Then do rate of change
63          #phosssummary.rate_of_change(informant, diphthong,
               norm_opt, target_one_f1)
64
65      print "Summary for %s using %s written to %s" % (informant.
           name, norm_opt, informant.informantdir)
66
67
68
69
70 def gSummary(InformantList, options, norm_opt):
71      for informant in InformantList.speakers:
72          # write header to the file
73          phosssummary.write_header(informant, "Direct Comparison
               ", norm_opt)
74          # write informant information
```

```
75          phosssummary.informant_information(informant)
76
77          # what absolute vowel space does the speaker occupy?
78          flabel1, flabel2 = tuple(informant.normed_data_frame.
                colnames)[3:5]
79          phosssummary.absolute_vowel_space(informant, flabel1,
                norm_opt)
80          phosssummary.absolute_vowel_space(informant, flabel2,
                norm_opt)
81
82          # vowel by vowel comparison
83          for vowel in informant.VowelSet:
84              phosssummary.vowel_header(informant, vowel)
85              # The F1 and F2 column names change by
                    normalization method. This keeps it straight
86              flabel1, flabel2 = tuple(informant.
                    normed_data_frame.colnames)[3:5]
87
88              # what are the maximum and minimum values for f1
                    and f2
89              phosssummary.max_and_min_f_values(informant, vowel,
                     flabel1, norm_opt)
90              phosssummary.max_and_min_f_values(informant, vowel,
                     flabel2, norm_opt)
91
92              # the group average for vowel X occupies the vowel
                    space centered here and ranging from X to Y.
```

```
93              phosssummary.gvowel_space(informant, vowel, flabel1
                    , flabel2, norm_opt)
94              phosssummary.geuc_examples(informant, vowel,
                    flabel1, flabel2, norm_opt)
95
96          # note to self: just remember, you have to do
                everything double.
97          if informant.diphthongs != '':
98              for diphthong in informant.diphthongs:
99                  target_one_f1, target_one_f2 = tuple(informant.
                        normed_data_frame.colnames)[3:5]
100                 if norm_opt == "Nearey" or norm_opt == "Labov"
                        or norm_opt == "Watt and Fabricius" or
                        norm_opt == "Lobanov": #Nearey
101                     target_two_f1_gl, target_two_f2_gl = tuple(
                            informant.normed_data_frame.colnames)
                            [5:]
102                 else:
103                     target_two_f1_gl, target_two_f2_gl = tuple(
                            informant.normed_data_frame.colnames)
                            [6:8]
104                 # Summarize first vowel target
105                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_one_f1, norm_opt, dtarg="
                        one")
106                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_one_f2, norm_opt)
```

```
107                 phosssummary.gvowel_space(informant, diphthong,
                        target_one_f1, target_one_f2, norm_opt,
                        dtarg="one")
108                 phosssummary.geuc_examples(informant, diphthong
                        , target_one_f1, target_one_f2, norm_opt,
                        dtarg="one")
109                 # Summarize second diphthong target
110                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_two_f1_gl, norm_opt, dtarg
                        ="two")
111                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_two_f2_gl, norm_opt)
112                 phosssummary.gvowel_space(informant, diphthong,
                        target_two_f1_gl, target_two_f2_gl,
                        norm_opt, dtarg="two")
113                 phosssummary.geuc_examples(informant, diphthong
                        , target_two_f1_gl, target_two_f2_gl,
                        norm_opt, dtarg="two")
114             print "Summary for %s using %s written to %s" % (
                    informant.name, norm_opt, informant.informantdir
                    )
115
116
117
118
119 def cSummary(InformantList, options, norm_opt):
120     for informant in InformantList.speakers:
```

```
121         # write header to the file
122         phosssummary.write_header(informant, "Comparative
                Summary", norm_opt)
123         # write informant information
124         phosssummary.informant_information(informant)
125
126         # what absolute vowel space does the speaker occupy?
127         flabel1, flabel2 = tuple(informant.normed_data_frame.
                colnames)[3:5]
128         phosssummary.absolute_vowel_space(informant, flabel1,
                norm_opt)
129         phosssummary.absolute_vowel_space(informant, flabel2,
                norm_opt)
130
131         # vowel by vowel comparison
132         for vowel in informant.VowelSet:
133             phosssummary.vowel_header(informant, vowel)
134             # The F1 and F2 column names change by
                    normalization method. This keeps it straight
135             flabel1, flabel2 = tuple(informant.
                    normed_data_frame.colnames)[3:5]
136
137             # what are the maximum and minimum values for f1
                    and f2
138             phosssummary.max_and_min_f_values(informant, vowel,
                      flabel1, norm_opt)
```

```
139             phosssummary.max_and_min_f_values(informant, vowel,
                    flabel2, norm_opt)
140
141             # the group average for vowel X occupies the vowel
                    space centered here and ranging from X to Y.
142             phosssummary.vowel_space(informant, vowel, flabel1,
                    flabel2, norm_opt)
143             phosssummary.euc_examples(informant, vowel, flabel1
                    , flabel2, norm_opt)
144
145         # note to self: just remember, you have to do
                everything double.
146         if informant.diphthongs != '':
147             for diphthong in informant.diphthongs:
148                 target_one_f1, target_one_f2 = tuple(informant.
                        normed_data_frame.colnames)[3:5]
149                 if norm_opt == "Nearey" or norm_opt == "Labov"
                        or norm_opt == "Watt and Fabricius" or
                        norm_opt == "Lobanov": #Nearey
150                     target_two_f1_gl, target_two_f2_gl = tuple(
                            informant.normed_data_frame.colnames)
                            [5:]
151                 else:
152                     target_two_f1_gl, target_two_f2_gl = tuple(
                            informant.normed_data_frame.colnames)
                            [6:8]
153                 # Summarize first vowel target
```

```
154                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_one_f1, norm_opt, dtarg="
                        one")
155                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_one_f2, norm_opt)
156                 phosssummary.vowel_space(informant, diphthong,
                        target_one_f1, target_one_f2, norm_opt,
                        dtarg="one")
157                 phosssummary.euc_examples(informant, diphthong,
                         target_one_f1, target_one_f2, norm_opt,
                        dtarg="one")
158                 # Summarize second diphthong target
159                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_two_f1_gl, norm_opt, dtarg
                        ="two")
160                 phosssummary.max_and_min_f_values(informant,
                        diphthong, target_two_f2_gl, norm_opt)
161                 phosssummary.vowel_space(informant, diphthong,
                        target_two_f1_gl, target_two_f2_gl, norm_opt
                        , dtarg="two")
162                 phosssummary.euc_examples(informant, diphthong,
                         target_two_f1_gl, target_two_f2_gl,
                        norm_opt, dtarg="two")
163                 # Then do rate of change
164                 #phosssummary.rate_of_change(informant,
                        diphthong, norm_opt, target_one_f1)
```

```
165             print "Summary for %s using %s written to %s" % (
                    informant.name, norm_opt, informant.informantdir
                    )
```

Listing B.7: Summary Module

```
1 import rpy2.robjects as R
2 from rpy2.robjects.packages import importr
3 from math import sqrt
4
5 V = importr('vowels')
6
7 def write_header(informant, summary_type, norm_opt):
8     f = open(informant.output_file, 'w')
9     f.write("PhoSS, A Phonetic Summarizer for Sociolinguists\n"
          )
10    f.write("%s\n" % summary_type)
11    f.write("%s\n" % norm_opt)
12    f.write("\n")
13    f.close()
14
15 def informant_information(informant):
16    # There is not a need at this point to have diphthongs
          together with
17    # monophthongs. Remove them in order to summarize them
          separately.
18    for dip in informant.diphthongs:
19        if dip in informant.VowelSet:
20            informant.VowelSet.remove(dip)
21
22    f = open(informant.output_file, 'a')
23    f.write("Speaker: %s\n" % informant.name)
```

```
24      f.write("Vowel Set: %s\n" % informant.VowelSet)
25      if informant.diphthongs != []:
26          f.write("Diphthongs: %s\n" % informant.diphthongs)
27      f.write("Audio File: %s\n" % informant.audio_file)
28      f.write("Transcript: %s\n" % informant.transcript_file)
29      f.write("TextGrid: %s\n" % informant.TextGrid)
30      f.write("\n")
31      f.close()
32
33 def vowel_header(informant, vowel):
34      f = open(informant.output_file, 'a')
35      f.write("Summary for %s\n" % vowel)
36      f.close()
37
38 def absolute_vowel_space(informant, flabel, norm_opt):
39      if flabel == informant.normed_data_frame.colnames[3]:
40          fn = 3
41      elif flabel == informant.normed_data_frame.colnames[4]:
42          fn = 4
43      # extract a subtable of all vowels for the current speaker
44      if norm_opt == 'Bark Difference' \
45 or norm_opt == 'Lobanov' \
46 or norm_opt == 'Labov' \
47 or norm_opt == 'Watt and Fabricius' \
48 or norm_opt == 'Nearey':
49          speaker = "Speaker"
50          vowelframe = "Vowel"
```

```
51      else:
52          speaker = "speaker"
53          vowelframe = "vowel.frame"
54      vowel_subset = informant.normed_data_frame.rx(informant.
            normed_data_frame.rx2(speaker).ro == informant.name,
            True)
55
56      fmaxindex = (R.r['which.max'](vowel_subset[fn])[0]) - 1
57      fmax = vowel_subset[fn][fmaxindex]
58      fmaxctxindex = (vowel_subset[2][fmaxindex]) - 1
59      fmaxcontext = vowel_subset[2].levels[fmaxctxindex]
60
61
62      fminindex = (R.r['which.min'](vowel_subset[fn])[0]) -1
63      fmin = vowel_subset[fn][fminindex]
64      fminctxindex = (vowel_subset[2][fminindex]) - 1
65      fmincontext = vowel_subset[2].levels[fminctxindex]
66
67      ftotal = fmax - fmin
68
69      f = open(informant.output_file, 'a')
70      f.write("%s vowel space for %s has a range of %.3f units,
            from %.3f to %.3f.\n" % (flabel, informant.name, ftotal,
             fmin, fmax))
71      if fn == 4:
72          f.write("\n")
73      f.close()
```

```
74      return
75
76
77
78 # find maximum and minimum f1 and f2 values for a vowel for a
       speaker
79 def max_and_min_f_values(informant, vowel, flabel, norm_opt,
       dtarg=""):
80      if flabel == informant.sub_data_frame.colnames[3]:
81          fn = 3
82      elif flabel == informant.sub_data_frame.colnames[4]:
83          fn = 4
84      elif flabel == informant.sub_data_frame.colnames[5]:
85          fn = 5
86      elif flabel == informant.sub_data_frame.colnames[6]:
87          fn = 6
88      elif flabel == informant.sub_data_frame.colnames[7]:
89          fn = 7
90      # extract a subtable for the vowel from the VowelSet
91      if norm_opt == 'Bark Difference' \
92 or norm_opt == 'Lobanov' \
93 or norm_opt == 'Labov' \
94 or norm_opt == 'Watt and Fabricius' \
95 or norm_opt == 'Nearey':
96          speaker = "Speaker"
97          vowelframe = "Vowel"
98      else:
```

```
99          speaker = "speaker"
100         vowelframe = "vowel.frame"
101     vowel_subset = informant.sub_data_frame.rx(informant.
            sub_data_frame.rx2(vowelframe).ro == vowel, True)
102
103     fmaxindex = (R.r['which.max'](vowel_subset[fn])[0]) - 1
104     fmax = vowel_subset[fn][fmaxindex]
105     fmaxctxindex = (vowel_subset[2][fmaxindex]) - 1
106     fmaxcontext = vowel_subset[2].levels[fmaxctxindex]
107
108
109     fminindex = (R.r['which.min'](vowel_subset[fn])[0]) -1
110     fmin = vowel_subset[fn][fminindex]
111     fminctxindex = (vowel_subset[2][fminindex]) - 1
112     fmincontext = vowel_subset[2].levels[fminctxindex]
113     # 'range' misspelled because it is a Python reserved word
114     rnge = fmax - fmin
115
116     f = open(informant.output_file, 'a')
117     if dtarg == "one" or dtarg == "two":
118         f.write("For target %s of %s:\n" % (dtarg, vowel))
119     f.write("  %s vowel space for %s has a range of %.3f units,
            from %.3f to %.3f.\n" % (flabel, vowel, rnge, fmin,
            fmax))
120     return
121
122
```

```
123
124
125 def vowel_space(informant, vowel, label1, label2, norm_opt,
      dtarg=""):
126     # The group should be the dataframe without the informant
          in it.
127     # In the case of two informants, this will mean the group
          is the other informant.
128     if norm_opt == 'Bark Difference' \
129 or norm_opt == 'Lobanov' \
130 or norm_opt == 'Labov' \
131 or norm_opt == 'Watt and Fabricius' \
132 or norm_opt == 'Nearey':
133         speaker = "Speaker"
134         vowelframe = "Vowel"
135     else:
136         speaker = "speaker"
137         vowelframe = "vowel.frame"
138     # the otherdf is the dataframe with everyone in it.
139     otherdf = informant.normed_data_frame
140
141     # calculate center of vowel for the group
142     gmeandf = V.compute_means(otherdf)
143     gv = gmeandf.rx(gmeandf.rx2("Vowel").ro == vowel, True)
144     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
          == "Labov" or norm_opt == "Watt and Fabricius" or
         norm_opt == "Lobanov"):
```

```
145         gmean1 = gv[6][0] # column 6
146         gmean2 = gv[7][0] # column 7
147     else:
148         gmean1 = gv[3][0] # column 3
149         gmean2 = gv[4][0] # column 4
150     # center of the average vowel space
151     gvowelcenter = (gmean1, gmean2)
152
153     # standard deviations for the group values
154     gsdsdf = V.compute_sds(otherdf)
155     sdv = gsdsdf.rx(gsdsdf.rx2("Vowel").ro == vowel, True)
156     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
           norm_opt == "Lobanov"):
157         vsds1 = sdv[6][0] # F1_gl column standard deviation for
                the otherdf
158         vsds2 = sdv[7][0] # F2_gl column standard deviation for
                the otherdf
159     else:
160         vsds1 = sdv[3][0] # F1 column standard deviation for
               the otherdf
161         vsds2 = sdv[4][0] # F2 column standard deviation for
               the otherdf
162
163     # vowel center for individual
164     imeandf = V.compute_means(informant.sub_data_frame)
165     iv = imeandf.rx(imeandf.rx2("Vowel").ro == vowel, True)
```

```
166     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
           norm_opt == "Lobanov"):
167         imean1 = iv[6][0]
168         imean2 = iv[7][0]
169     else:
170         imean1 = iv[3][0]
171         imean2 = iv[4][0]
172     ivowelcenter = (imean1, imean2)
173     #get informant min averages and max averages?s
174
175     diff1 = abs(imean1 - gmean1)
176     diff2 = abs(imean2 - gmean2)
177
178     label1sdsfrommean = diff1 / vsds1
179     label2sdsfrommean = diff2 / vsds2
180
181
182     f = open(informant.output_file, 'a')
183     #if dtarg == "one" or dtarg == "two":
184     #    f.write("For target %s of %s:\n" % (dtarg, vowel))
185
186     f.write("  %s for %s centers at (%s %.3f, %s %.3f), which
           is %.3f standard deviations from the average %s and %.3f
            standard deviations from the average %s.\n" % (vowel,
           informant.name, label1, ivowelcenter[0], label2,
```

```
            ivowelcenter[1], label1sdsfrommean, label1,
            label2sdsfrommean, label2))
187         f.write("  The group average for %s occupies vowel space
            centered at (%s %.3f, %s %.3f) with an %s standard
            deviation of %.3f and an %s standard deviation of %.3f.\
            n" % (vowel, label1, gvowelcenter[0], label2,
            gvowelcenter[1], label1, vsds1, label2, vsds2))
188         f.write('\n')
189         f.close()
190
191
192
193
194 def gvowel_space(informant, vowel, label1, label2, norm_opt,
        dtarg=""):
195         # The group should be the dataframe without the informant
            in it.
196         # In the case of two informants, this will mean the group
            is the other informant.
197         if norm_opt == 'Bark Difference' \
198 or norm_opt == 'Lobanov' \
199 or norm_opt == 'Labov' \
200 or norm_opt == 'Watt and Fabricius' \
201 or norm_opt == 'Nearey':
202             speaker = "Speaker"
203             vowelframe = "Vowel"
204         else:
```

```
205         speaker = "speaker"
206         vowelframe = "vowel.frame"
207     # the otherdf is the dataframe with everyone except the
            current informant in it.
208     otherdf = informant.normed_data_frame.rx(informant.
            normed_data_frame.rx2(speaker).ro != informant.name,
            True)
209
210     # calculate center of vowel for the group
211     gmeandf = V.compute_means(otherdf)
212     gv = gmeandf.rx(gmeandf.rx2("Vowel").ro == vowel, True)
213     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
            norm_opt == "Lobanov"):
214         gmean1 = gv[6][0] # column 6
215         gmean2 = gv[7][0] # column 7
216     else:
217         gmean1 = gv[3][0] # column 3
218         gmean2 = gv[4][0] # column 4
219     # center of the average vowel space
220     gvowelcenter = (gmean1, gmean2)
221
222     # standard deviations for the group values
223     gsdsdf = V.compute_sds(otherdf)
224     sdv = gsdsdf.rx(gsdsdf.rx2("Vowel").ro == vowel, True)
```

```
225     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
           norm_opt == "Lobanov"):
226         vsds1 = sdv[6][0] # F1_gl column standard deviation for
                the otherdf
227         vsds2 = sdv[7][0] # F2_gl column standard deviation for
                the otherdf
228     else:
229         vsds1 = sdv[3][0] # F1 column standard deviation for
               the otherdf
230         vsds2 = sdv[4][0] # F2 column standard deviation for
               the otherdf
231
232     # vowel center for individual
233     imeandf = V.compute_means(informant.sub_data_frame)
234     iv = imeandf.rx(imeandf.rx2("Vowel").ro == vowel, True)
235     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
           norm_opt == "Lobanov"):
236         imean1 = iv[6][0]
237         imean2 = iv[7][0]
238     else:
239         imean1 = iv[3][0]
240         imean2 = iv[4][0]
241     ivowelcenter = (imean1, imean2)
242     #get informant min averages and max averages?s
243
```

```
244     diff1 = abs(imean1 - gmean1)
245     diff2 = abs(imean2 - gmean2)
246
247     label1sdsfrommean = diff1 / vsds1
248     label2sdsfrommean = diff2 / vsds2
249
250
251     f = open(informant.output_file, 'a')
252     #if dtarg == "one" or dtarg == "two":
253     #   f.write("For target %s of %s:\n" % (dtarg, vowel))
254
255     f.write(" %s for %s centers at (%s %.3f, %s %.3f), which
            is %.3f standard deviations from the average %s and %.3f
             standard deviations from the average %s.\n" % (vowel,
            informant.name, label1, ivowelcenter[0], label2,
            ivowelcenter[1], label1sdsfrommean, label1,
            label2sdsfrommean, label2))
256     f.write(" The rest of the group's average for %s occupies
            vowel space centered at (%s %.3f, %s %.3f) with an %s
            standard deviation of %.3f and an %s standard deviation
            of %.3f.\n" % (vowel, label1, gvowelcenter[0], label2,
            gvowelcenter[1], label1, vsds1, label2, vsds2))
257     f.write('\n')
258     f.close()
259
260
261
```

```
262
263 def indiv_vowel_space(informant, vowel, label1, label2,
        norm_opt, dtarg=""):
264     if norm_opt == 'Bark Difference' \
265 or norm_opt == 'Lobanov' \
266 or norm_opt == 'Labov' \
267 or norm_opt == 'Watt and Fabricius' \
268 or norm_opt == 'Nearey':
269         speaker = "Speaker"
270         vowelframe = "Vowel"
271     else:
272         speaker = "speaker"
273         vowelframe = "vowel.frame"
274     # use the speaker sub data frame and isolate the vowel we'
            re looking at.
275     voweldf = informant.sub_data_frame.rx(informant.
            sub_data_frame.rx2(vowelframe).ro == vowel, True)
276     isds = V.compute_means(voweldf)
277     if dtarg == "two":
278         f1mean = isds[6][0]
279         f2mean = isds[7][0]
280     else:
281         f1mean = isds[3][0] # F1 mean
282         f2mean = isds[4][0] # F2 mean
283
284     fsds = V.compute_sds(voweldf)
285     if dtarg == "two":
```

```
286             f1sds = fsds[6][0]
287             f2sds = fsds[7][0]
288         else:
289             f1sds = fsds[3][0]
290             f2sds = fsds[4][0]
291
292 #       V.compute_medians(voweldf)
293
294         f = open(informant.output_file, 'a')
295         f.write("  The average %s and %s values for %s are %.3f,
              %.3f, with an %s standard deviation of %.3f and an %s
              standard deviation of %.3f.\n" % (label1, label2, vowel,
               f1mean, f2mean, label1, f1sds, label2, f2sds))
296         f.write('\n')
297
298
299
300
301
302 def euc_examples(informant, vowel, label1, label2, norm_opt,
        dtarg=""):
303         zero_to_one_half = []
304         one_half_to_one = []
305         one_to_one_and_half = []
306         one_and_half_to_two = []
307         two_to_two_and_half = []
308         gt_two_and_half = []
```

```
309
310     if norm_opt == 'Bark Difference' \
311 or norm_opt == 'Lobanov' \
312 or norm_opt == 'Labov' \
313 or norm_opt == 'Watt and Fabricius' \
314 or norm_opt == 'Nearey':
315         speaker = "Speaker"
316         vowelframe = "Vowel"
317     else:
318         speaker = "speaker"
319         vowelframe = "vowel.frame"
320     # otherdf is the dataframe for the whole group, including
        the current informant.
321     # If you want to make a direct comparison and remove the
        informant from the df, use the -g flag.
322     otherdf = informant.normed_data_frame
323
324     # find center of the group's vowel space.
325     gmeandf = V.compute_means(otherdf)
326     gv = gmeandf.rx(gmeandf.rx2("Vowel").ro == vowel, True)
327     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
        == "Labov" or norm_opt == "Watt and Fabricius" or
        norm_opt == "Lobanov"):
328         gmean1 = gv[6][0] # column 3
329         gmean2 = gv[7][0] # column 4
330     else:
331         gmean1 = gv[3][0] # column 3
```

```
332         gmean2 = gv[4][0] # column 4
333     # center of the average vowel space
334     gvc = (gmean1, gmean2)
335
336     # standard deviations for the group values
337     gsdsdf = V.compute_sds(otherdf)
338     sdv = gsdsdf.rx(gsdsdf.rx2("Vowel").ro == vowel, True)
339     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
            norm_opt == "Lobanov"):
340         sdcoord = (sdv[6][0], sdv[7][0])
341     else:
342         sdcoord = (sdv[3][0], sdv[4][0])
343     # the Euclidean distance of one standard deviation
344     sded = sqrt( ((gvc[0] - sdcoord[0])**2) + ((gvc[1] -
            sdcoord[1])**2) )
345
346     #get the subframe with the right vowel instances in it
347     vf = informant.sub_data_frame.rx(informant.sub_data_frame.
            rx2(vowelframe).ro == vowel, True)
348     for row in vf.iter_row():
349         if dtarg == "two" and not (norm_opt == "Nearey" or
                norm_opt == "Labov" or norm_opt == "Watt and
                Fabricius" or norm_opt == "Lobanov"):
350             icoord = (row[6][0], row[7][0])
351         else:
352             icoord = (row[3][0], row[4][0])
```

```
353         context = row[2].levels[0]
354         # the Euclidean distance of each vowel from the group
                mean vowel center
355         ed = sqrt( ((gvc[0] - icoord[0])**2) + ((gvc[1] -
                icoord[1])**2) )
356
357         sdsfromgvc  = ed / sded
358
359         if sdsfromgvc < 0.5:
360             zero_to_one_half.append((sdsfromgvc, context))
361         elif 0.5 < sdsfromgvc < 1:
362             one_half_to_one.append((sdsfromgvc, context))
363         elif 1 < sdsfromgvc < 1.5:
364             one_to_one_and_half.append((sdsfromgvc, context))
365         elif 1.5 < sdsfromgvc < 2:
366             one_and_half_to_two.append((sdsfromgvc, context))
367         elif 2 < sdsfromgvc < 2.5:
368             two_to_two_and_half.append((sdsfromgvc, context))
369         else:
370             gt_two_and_half.append((sdsfromgvc, context))
371
372     f = open(informant.output_file, 'a')
373
374     #if len(zero_to_one_half) > 0:
375     #   f.write("  There are %d instances between 0 and 0.5
            standard deviations from the average vowel center:\n" %
            len(zero_to_one_half))
```

```
376     #   [f.write("  %s\n" % ex[1]) for ex in zero_to_one_half]
377     if len(zero_to_one_half) == len(vf[0]):
378         f.write("  All %d instances of %s for %s are less than
                one half standard deviation from the mean vowel
                center.\n" % (len(zero_to_one_half), vowel,
                informant.name))
379         f.write('\n')
380
381     else:
382         f.write("  %d of %d instances are less than one half
                standard deviation from the mean vowel center.\n" %
                (len(zero_to_one_half), len(vf[0])))
383     f.write('\n')
384
385     if len(one_half_to_one) > 0:
386         f.write("  There are %d instances between 0.5 and 1
                standard deviations from the average vowel center:\n
                " % len(one_half_to_one))
387         [f.write("  %s\n" % ex[1]) for ex in one_half_to_one]
388         f.write('\n')
389
390     if len(one_to_one_and_half) > 0:
391         f.write("  There are %d instances between 1 and 1.5
                standard deviations from the average vowel center:\n
                " % len(one_to_one_and_half))
392         [f.write("  %s\n" % ex[1]) for ex in
                one_to_one_and_half]
```

```
393         f.write('\n')
394
395     if len(one_and_half_to_two) > 0:
396         f.write("  There are %d instances between 1.5 and 2
                standard deviations from the average vowel center:\n
                " % len(one_and_half_to_two))
397         [f.write("  %s\n" % ex[1]) for ex in
                one_and_half_to_two]
398         f.write('\n')
399
400     if len(two_to_two_and_half) > 0:
401         f.write("  There are %d instances between 2 and 2.5
                standard deviations from the average vowel center:\n
                " % len(two_to_two_and_half))
402         [f.write("  %s\n" % ex[1]) for ex in
                two_to_two_and_half]
403         f.write('\n')
404
405     if len(gt_two_and_half) > 0:
406         f.write("  There are %d instances greater than 2.5
                standard deviations from the average vowel center:\n
                " % len(gt_two_and_half))
407         [f.write("  %s\n" % ex[1]) for ex in gt_two_and_half]
408         f.write('\n')
409
410     f.close()
411     return
```

```
412
413
414
415
416 def indiv_euc(informant, vowel, label1, label2, norm_opt, dtarg
      =""):
417     zero_to_one_half = []
418     one_half_to_one = []
419     one_to_one_and_half = []
420     one_and_half_to_two = []
421     two_to_two_and_half = []
422     gt_two_and_half = []
423
424     if norm_opt == 'Bark Difference' \
425 or norm_opt == 'Lobanov' \
426 or norm_opt == 'Labov' \
427 or norm_opt == 'Watt and Fabricius' \
428 or norm_opt == 'Nearey':
429         speaker = "Speaker"
430         vowelframe = "Vowel"
431     else:
432         speaker = "speaker"
433         vowelframe = "vowel.frame"
434     # separate just the vowel
435     voweldf = informant.sub_data_frame.rx(informant.
          sub_data_frame.rx2(vowelframe).ro == vowel, True)
436
```

```
437     # Find the vowel center
438     imean = V.compute_means(voweldf)
439     if dtarg == "two":
440         meancoord = (imean[6][0], imean[7][0])
441     else:
442         meancoord = (imean[3][0], imean[4][0])
443
444     # Standard deviations
445     fsds = V.compute_sds(voweldf)
446     if dtarg == "two":
447         sdscoord = (fsds[6][0], fsds[7][0])
448     else:
449         sdscoord = (fsds[3][0], fsds[4][0])
450
451     # the Euclidean distance of one standard deviation
452     # this will be the yardstick for all vowel instances
453     sded = sqrt( ((meancoord[0] - sdscoord[0])**2) + ((
            meancoord[1] - sdscoord[1])**2) )
454
455     for row in voweldf.iter_row():
456         if dtarg == "two":
457             # icoord is the coordinate of a vowel instance
458             icoord = (row[6][0], row[7][0])
459         else:
460             icoord = (row[3][0], row[4][0])
461         context = row[2].levels[0]
462
```

```
463         # the Euclidean distance of each vowel from the indiv
                vowel center
464         ed = sqrt( ((meancoord[0] - icoord[0])**2) + ((
                meancoord[1] - icoord[1])**2) )
465
466         sdsfrommean  = ed / sded
467
468         if sdsfrommean < 0.5:
469             zero_to_one_half.append((sdsfrommean, context))
470         elif 0.5 < sdsfrommean < 1:
471             one_half_to_one.append((sdsfrommean, context))
472         elif 1 < sdsfrommean < 1.5:
473             one_to_one_and_half.append((sdsfrommean, context))
474         elif 1.5 < sdsfrommean < 2:
475             one_and_half_to_two.append((sdsfrommean, context))
476         elif 2 < sdsfrommean < 2.5:
477             two_to_two_and_half.append((sdsfrommean, context))
478         else:
479             gt_two_and_half.append((sdsfrommean, context))
480
481     f = open(informant.output_file, 'a')
482     if len(zero_to_one_half) == len(voweldf[0]):
483         f.write("  All %d instances of %s for %s are less than
                one half standard deviation from the mean vowel
                center.\n" % (len(zero_to_one_half), vowel,
                informant.name))
484         f.write('\n')
```

```
485
486     else:
487         f.write("  %d of %d instances are less than one half
                standard deviation from the mean vowel center.\n" %
                (len(zero_to_one_half), len(vf[0])))
488     f.write('\n')
489
490     if len(one_half_to_one) > 0:
491         f.write("  There are %d instances between 0.5 and 1
                standard deviations from the average vowel center:\n
                " % len(one_half_to_one))
492         [f.write("  %s\n" % ex[1]) for ex in one_half_to_one]
493         f.write('\n')
494
495     if len(one_to_one_and_half) > 0:
496         f.write("  There are %d instances between 1 and 1.5
                standard deviations from the average vowel center:\n
                " % len(one_to_one_and_half))
497         [f.write("  %s\n" % ex[1]) for ex in
                one_to_one_and_half]
498         f.write('\n')
499
500     if len(one_and_half_to_two) > 0:
501         f.write("  There are %d instances between 1.5 and 2
                standard deviations from the average vowel center:\n
                " % len(one_and_half_to_two))
```

```
502             [f.write("  %s\n" % ex[1]) for ex in
                    one_and_half_to_two]
503             f.write('\n')
504
505         if len(two_to_two_and_half) > 0:
506             f.write("  There are %d instances between 2 and 2.5
                    standard deviations from the average vowel center:\n
                    " % len(two_to_two_and_half))
507             [f.write("  %s\n" % ex[1]) for ex in
                    two_to_two_and_half]
508             f.write('\n')
509
510         if len(gt_two_and_half) > 0:
511             f.write("  There are %d instances greater than 2.5
                    standard deviations from the average vowel center:\n
                    " % len(gt_two_and_half))
512             [f.write("  %s\n" % ex[1]) for ex in gt_two_and_half]
513             f.write('\n')
514
515         f.close()
516         return
517
518
519
520
521 def geuc_examples(informant, vowel, label1, label2, norm_opt,
        dtarg=""):
```

```
522     zero_to_one_half = []
523     one_half_to_one = []
524     one_to_one_and_half = []
525     one_and_half_to_two = []
526     two_to_two_and_half = []
527     gt_two_and_half = []
528
529     if norm_opt == 'Bark Difference' \
530 or norm_opt == 'Lobanov' \
531 or norm_opt == 'Labov' \
532 or norm_opt == 'Watt and Fabricius' \
533 or norm_opt == 'Nearey':
534         speaker = "Speaker"
535         vowelframe = "Vowel"
536     else:
537         speaker = "speaker"
538         vowelframe = "vowel.frame"
539     # separate individual from the group
540     otherdf = informant.normed_data_frame.rx(informant.
            normed_data_frame.rx2(speaker).ro != informant.name,
            True)
541
542     # find center of the group's vowel space.
543     gmeandf = V.compute_means(otherdf)
544     gv = gmeandf.rx(gmeandf.rx2("Vowel").ro == vowel, True)
```

```
545     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
            norm_opt == "Lobanov"):
546         gmean1 = gv[6][0] # column 3
547         gmean2 = gv[7][0] # column 4
548     else:
549         gmean1 = gv[3][0] # column 3
550         gmean2 = gv[4][0] # column 4
551     # center of the average vowel space
552     gvc = (gmean1, gmean2)
553
554     # standard deviations for the group values
555     gsdsdf = V.compute_sds(otherdf)
556     sdv = gsdsdf.rx(gsdsdf.rx2("Vowel").ro == vowel, True)
557     if dtarg == "two" and not (norm_opt == "Nearey" or norm_opt
            == "Labov" or norm_opt == "Watt and Fabricius" or
            norm_opt == "Lobanov"):
558         sdcoord = (sdv[6][0], sdv[7][0])
559     else:
560         sdcoord = (sdv[3][0], sdv[4][0])
561     # the Euclidean distance of one standard deviation
562     sded = sqrt( ((gvc[0] - sdcoord[0])**2) + ((gvc[1] -
            sdcoord[1])**2) )
563
564     #get the subframe with the right vowel instances in it
565     vf = informant.sub_data_frame.rx(informant.sub_data_frame.
            rx2(vowelframe).ro == vowel, True)
```

```
566     for row in vf.iter_row():
567         if dtarg == "two" and not (norm_opt == "Nearey" or
              norm_opt == "Labov" or norm_opt == "Watt and
              Fabricius" or norm_opt == "Lobanov"):
568             icoord = (row[6][0], row[7][0])
569         else:
570             icoord = (row[3][0], row[4][0])
571         context = row[2].levels[0]
572         # the Euclidean distance of each vowel from the group
              mean vowel center
573         ed = sqrt( ((gvc[0] - icoord[0])**2) + ((gvc[1] -
              icoord[1])**2) )
574
575         sdsfromgvc  = ed / sded
576
577         if sdsfromgvc < 0.5:
578             zero_to_one_half.append((sdsfromgvc, context))
579         elif 0.5 < sdsfromgvc < 1:
580             one_half_to_one.append((sdsfromgvc, context))
581         elif 1 < sdsfromgvc < 1.5:
582             one_to_one_and_half.append((sdsfromgvc, context))
583         elif 1.5 < sdsfromgvc < 2:
584             one_and_half_to_two.append((sdsfromgvc, context))
585         elif 2 < sdsfromgvc < 2.5:
586             two_to_two_and_half.append((sdsfromgvc, context))
587         else:
588             gt_two_and_half.append((sdsfromgvc, context))
```

```
589
590     f = open(informant.output_file, 'a')
591
592     #if len(zero_to_one_half) > 0:
593     #   f.write("  There are %d instances between 0 and 0.5
            standard deviations from the average vowel center:\n" %
            len(zero_to_one_half))
594     #   [f.write("  %s\n" % ex[1]) for ex in zero_to_one_half]
595     if len(zero_to_one_half) == len(vf[0]):
596         f.write("  All %d instances of %s for %s are less than
                one half standard deviation from the mean vowel
                center.\n" % (len(zero_to_one_half), vowel,
                informant.name))
597         f.write('\n')
598
599     else:
600         f.write("  %d of %d instances are less than one half
                standard deviation from the mean vowel center.\n" %
                (len(zero_to_one_half), len(vf[0])))
601     f.write('\n')
602
603     if len(one_half_to_one) > 0:
604         f.write("  There are %d instances between 0.5 and 1
                standard deviations from the average vowel center:\n
                " % len(one_half_to_one))
605         [f.write("  %s\n" % ex[1]) for ex in one_half_to_one]
606         f.write('\n')
```

```
607
608     if len(one_to_one_and_half) > 0:
609         f.write("  There are %d instances between 1 and 1.5
                standard deviations from the average vowel center:\n
                " % len(one_to_one_and_half))
610         [f.write("  %s\n" % ex[1]) for ex in
                one_to_one_and_half]
611         f.write('\n')
612
613     if len(one_and_half_to_two) > 0:
614         f.write("  There are %d instances between 1.5 and 2
                standard deviations from the average vowel center:\n
                " % len(one_and_half_to_two))
615         [f.write("  %s\n" % ex[1]) for ex in
                one_and_half_to_two]
616         f.write('\n')
617
618     if len(two_to_two_and_half) > 0:
619         f.write("  There are %d instances between 2 and 2.5
                standard deviations from the average vowel center:\n
                " % len(two_to_two_and_half))
620         [f.write("  %s\n" % ex[1]) for ex in
                two_to_two_and_half]
621         f.write('\n')
622
623     if len(gt_two_and_half) > 0:
```

```
624             f.write("  There are %d instances greater than 2.5
                    standard deviations from the average vowel center:\n
                    " % len(gt_two_and_half))
625             [f.write("  %s\n" % ex[1]) for ex in gt_two_and_half]
626             f.write('\n')
627
628         f.close()
629         return
630
631
632
633 # NOTE: Incomplete. This would need to include the durations as
        a column in the dataframe (That could be assembled at the
        same time as the context), and the durations need to be
        included to make sure that ROC is divided over duration for
        each instance.
634 def rate_of_change(informant, diphthong, norm_opt, flabel1):
635         roc_list = []
636         if norm_opt == 'Bark Difference' \
637 or norm_opt == 'Lobanov' \
638 or norm_opt == 'Labov' \
639 or norm_opt == 'Watt and Fabricius' \
640 or norm_opt == 'Nearey':
641             speaker = "Speaker"
642             vowelframe = "Vowel"
643         else:
644             speaker = "speaker"
```

```
645         vowelframe = "vowel.frame"
646     vowel_subset = informant.sub_data_frame.rx(informant.
            sub_data_frame.rx2(vowelframe).ro == diphthong, True)
647     # subtract the beginning f1 from the ending f1
648     for row in vowel_subset.iter_row():
649         roc = row[6][0] - row[3][0]
650         context = row[2].levels[0]
651         roc_list.append((roc, context))
652     positive_roc = [elt for elt in roc_list if elt[0] >= 0]
653     negative_roc = [elt for elt in roc_list if elt[0] < 0]
654     positive_roc.sort(reverse=True)
655     negative_roc.sort(reverse=True)
656     f = open(informant.output_file, 'a')
657     f.write("%s Rate of Change for %s\n" % (flabel1, diphthong)
            )
658     for proc in positive_roc:
659         f.write("  %.3f, %s\n" % (proc[0], proc[1]))
660     for nroc in negative_roc:
661         f.write("  %.3f, %s\n" % (nroc[0], nroc[1]))
662     f.write('\n')
```