

5-1-2010

# Syllables and Concepts in Large Vocabulary Speech Recognition

Paul A. De Palma

Follow this and additional works at: [https://digitalrepository.unm.edu/ling\\_etds](https://digitalrepository.unm.edu/ling_etds)

---

## Recommended Citation

De Palma, Paul A.. "Syllables and Concepts in Large Vocabulary Speech Recognition." (2010). [https://digitalrepository.unm.edu/ling\\_etds/9](https://digitalrepository.unm.edu/ling_etds/9)

This Dissertation is brought to you for free and open access by the Electronic Theses and Dissertations at UNM Digital Repository. It has been accepted for inclusion in Linguistics ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact [disc@unm.edu](mailto:disc@unm.edu).

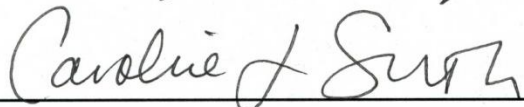
Paul De Palma  
*Candidate*

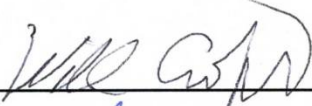
Linguistics  
*Department*

This dissertation is approved, and it is acceptable in quality  
and form for publication:

*Approved by the Dissertation Committee:*

 \_\_\_\_\_, Chairperson

 \_\_\_\_\_

 \_\_\_\_\_

 \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

**SYLLABLES AND CONCEPTS IN LARGE VOCABULARY**

**CONTINUOUS SPEECH RECOGNITION**

**BY**

**PAUL DE PALMA**

A.B., English, St. Louis University, 1969

M.A., English, University of California at Berkeley, 1975

M.S., Computer Science, Temple University, 1990

**DISSERTATION**

Submitted in Partial Fulfillment of the  
Requirements for the Degree of

**Doctor of Philosophy**  
**Linguistics**

The University of New Mexico  
Albuquerque, New Mexico

**May, 2010**

## DEDICATION

Alla mia cara madre che mi ha insegnato a scrivere.

Al mio caro padre che mi ha insegnato a parlare.

To Walter J. Ong, S.J (1912 – 2003), whose life-long dedication to scholarship has remained an inspiration these many years.

## ACKNOWLEDGEMENTS

I would like to acknowledge the enthusiasm, encouragement, creativity, and learning of Dr. George Luger, advisor, dissertation chair, host, and friend. Without Dr. Luger this entire adventure in Albuquerque would not have happened. I would also like to acknowledge the rest of my committee, Dr. William Croft, Dr. Caroline Smith, and Dr. Charles Wooters. I am deeply grateful for their seemingly boundless knowledge of linguistics and speech recognition, for their willingness to read and re-read my many partial efforts, for their flexibility and understanding, and, above all, for never having asked, not once, what someone my age wants with a Ph.D. in Linguistics.

I would like further to acknowledge Gonzaga University and its School of Engineering and Applied Science along with my Dean, Dr. Dennis Horn, for having allowed me the enviable freedom of a sabbatical year at the University of New Mexico. Thanks also to the Next It Corporation of Spokane, WA for software, for hardware, for data, and for making its talented staff available to answer my many, and sometimes naïve, questions. Finally, and above all, thanks to my wife, Heidi Gann, for her unwavering support, her love, and for her willingness to endure many months alone—even though she thought the whole enterprise a little crazy.

**SYLLABLES AND CONCEPTS IN LARGE VOCABULARY**

**CONTINUOUS SPEECH RECOGNITION**

**BY**

**PAUL DE PALMA**

**ABSTRACT OF DISSERTATION**

Submitted in Partial Fulfillment of the  
Requirements for the Degree of

**Doctor of Philosophy  
Linguistics**

The University of New Mexico  
Albuquerque, New Mexico

**May, 2010**

**SYLLABLES AND CONCEPTS IN LARGE VOCABULARY CONTINUOUS  
SPEECH RECOGNITION**

**BY**

**PAUL DE PALMA**

A.B., English, St. Louis University, 1969  
M.A., English, University of California at Berkeley, 1975  
M.S., Computer Science, Temple University, 1990  
Ph.D., Linguistics, The University of New Mexico, 2010

**ABSTRACT**

Transforming an acoustic signal to words is the gold standard in automatic speech recognition. While recognizing that orthographic transcription is a valuable technique for comparing speech recognition systems without respect to application, it must also be recognized that transcription is not something that human beings do with their language partners. In fact, transforming speech into words is not necessary to emulate human performance in many contexts. By relaxing the constraint that the output of speech recognition be words, we might at the same time effectively relax the bias toward writing in speech recognition research. This puts our work in the camp of those who have argued over the years that speech and writing differ in significant ways.

This study explores two hypotheses. The first is that a large vocabulary continuous speech recognition (LVCSR) system will perform more accurately if it were trained on syllables instead of words. Though several researchers have examined the use of syllables in the acoustic model of an LVCSR system, very little attention has been paid to their use in the

language model. The second hypothesis has to do with adding a post-processing component to a recognizer equipped with a syllable language model. The first step is to group words that seem to mean the same thing into equivalence classes called *concepts*. The second step is to insert the equivalence classes into the output of a recognizer. The hypothesis is that by using this concept post-processor, we will achieve better results than with the syllable language model alone.

The study reports that the perplexity of a trigram syllable language model drops by half when compared to a trigram word language model using the same training transcript. The drop in perplexity carries over to error rate. The error rate of a recognizer equipped with syllable language model drops by over 14% when compared with one using a word language model. Nevertheless, the study reports a slight increase in error rate when a concept post-processor is added to a recognizer equipped with a syllable language model. We conjecture that this is the result of deterministic mapping from syllable strings to concepts. Consequently, we outline a probabilistic mapping scheme from concepts to syllable strings.



## TABLE OF CONTENTS

List of Figures .....	xii
List of Tables .....	xiv
Preface (and an Apologia) .....	xv
1.0        What is Automatic Speech Recognition and Who Uses It?.....	1
1.2        Classifying the Speech Recognition Task.....	5
1.3        Thoughts on “Normal Science” .....	6
1.4        Automatic Speech Recognition As Cryptanalysis .....	12
1.5        A First Pass at Bayesian Inference and Automatic Speech Recognition .....	15
1.6        Why ASR is Hard .....	17
1.7        The Architecture of Large Vocabulary Speech Recognitions Systems .....	19
1.7.1     Feature Extraction .....	19
1.7.2     The Acoustic Model and the Lexicon .....	22
1.7.3     The Language Model .....	24
1.7.4     Representing the Language Model .....	30
1.8        Scoring .....	32
1.9        The Syllable/Concept Hypothesis .....	34
1.10      Adding an Expert System .....	37
1.11      The Shape of Things to Come .....	39
2.0        Previous Work .....	41

2.1	Syllables .....	41
2.1.1	Syllabifiers .....	46
2.1.2	Stress and the NIST Syllabifier.....	49
2.1.3	Syllables and Stress.....	50
2.2	Syllables in the Acoustic Model .....	52
2.3	Syllables in the Language Model.....	59
2.4	Equivalence Classes in Speech Recognition.....	62
3.0	Experiment 1, Perplexity: A First Look at Syllables in the Language Model	66
3.1.0	Materials .....	74
3.1.2	Specific Techniques .....	76
3.1.3	Results.....	77
3.1.4	Discussion .....	79
4.0	Experiment 2: A Syllable Language Model.....	83
4.1	Materials .....	87
4.2	Specific Techniques .....	88
4.3	Generating A Syllable Language Model.....	89
4.4	Experiment 2.1 .....	90
4.4.1	Results for Experiment 2.1 .....	91
4.4.2	Discussion of Experiment 2.1 .....	93
4.5	Experiment 2.2.....	94

4.5.1	Results from Experiment 2.2 .....	96
4.5.2	Discussion of Results for Experiment 2.2.....	97
4.6	Stress Markings and the NIST Syllabifier, Experiment 2.3.....	98
4.6.1	Materials .....	100
4.6.2	Technique.....	100
4.6.3	Results.....	100
4.6.4	Discussion .....	101
5.0	Experiment 3: A Concept Component.....	103
5.2	The Concept Model.....	107
5.3	Experiment 3.1 .....	111
5.3.1	Materials .....	112
5.3.2	Results from Experiment 3.1 .....	113
5.3.2	Discussion .....	114
5.4	Experiment 3.2.....	118
5.4.1	Materials .....	120
5.4.2	Results.....	122
5.4.3	Discussion .....	123
6.0	Conclusions and Future Research.....	125
6.1	The Contours of a Probabilistic Concept Component .....	127
6.1.1	Computing the Prior Probability .....	131

6.1.2	Computing the Likelihood .....	132
6.1.3	Computing the Most Likely Concept Sequence .....	133
6.2	Conclusion .....	134
References		136
Appendices		150
Appendix A	A Derivation of Bayes' Theorem .....	151
Appendix B	Finite State Automata, Hidden Markov Models, Dynamic Programming.....	169
Appendix C	The Training Transcript.....	184
Appendix D	Reference Files .....	209
Appendix E	Four Equivalence Classes Used in Experiment 3 .....	213
Appendix F	Results, Experiments 2.1 and 2.2 .....	217
Appendix G	Output of Sclite for Experiment 3.1 .....	229
Appendix H	Output of Sclite for Experiment 3.2 .....	233
Appendix I	Software Written for the Study.....	237
Appendix J	Word-Transcription Lexicon .....	294
Appendix K	Word-Syllabification Lexicon .....	325
Appendix L	SONIC Symbol Set (from Pellom and Hacıoglu, 2005, p. 12) .....	356

## List of Figures

Figure 1.1: Diagram of an LVCSR System .....	19
Figure 1.2: HMM for the Word Six .....	23
Figure 1.3: Mini Corpus.....	27
Figure 1.4: Bigram Probabilities for Figure 1.3.....	28
Figure 1.5: ARPA Formatted Language Model.....	31
Figure 1.6: Symbol Error Rate.....	32
Figure 1.7: Alignment of Two Strings .....	33
Figure 1.8: A Recognizer with a Syllable Language Model and a Concept Post-Processor ..	37
Figure 1.9: Adding Expertise, Speech Synthesis, and a Web Display .....	38
Figure 2.1: General Syllable Structure.....	45
Figure 2.2: “The cumulative frequency of syllables in the entire Switchboard Corpus as a function of syllable frequency rank compared with the cumulative Frequency of occurrence for words in the same corpus” .....	54
Figure 4.1: Some Components of a Recognizer .....	84
Figure 4.2: Mean Errors as a Function of N-gram Size ( $N = 1, 2, 3, 4$ ) .....	92
Figure 4.3: Mean Ratio of Syllables to Words by Error Type.....	93
Figure 4.4: Word Reference File and Two Output Files .....	95
Figure 4.5: Syllable Reference File .....	95
Figure 4.6: Mean Errors as a Function of N-gram Size, $N = 1, 2, 3, 4$ .....	96
Figure 4.7: Mean Ratio of Syllables to Words by Error Type Across N-gram Sizes 2, 3, 4...	97
Figure 5.1: The Complete System .....	103
Figure 5.2: Bijection .....	105

Figure 5.3: Surjection.....	106
Figure 5.4: Ratio of Concept Runs to Syllable Runs by Error Type .....	114
Figure 5.5: N-Best Hypotheses .....	120
Figure 5.6: Ratio of Mean Errors Experiment 3.2 to Mean Errors Experiment 3.1 .....	123

## List of Tables

Table 2.1: Frequency distribution of words by syllable length. ....	53
Table 3.1: Perplexity, Word Language Model.....	78
Table 3.2: Perplexity, Syllable Language Model.....	78
Table 3.3: Mean Perplexity of Normed Word Language Models .....	78
Table 3.4: Mean Perplexity of Normed Syllable Language Models .....	79
Table 3.5: Perplexity of Word and Syllable Models for Bigrams and Trigrams.....	80
Table 3.6: Comparison of Entropy Rates for Syllables and Words.....	81
Table 4.1: Mean Errors for Word Language Models by N-gram Size .....	91
Table 4.2: Mean Errors for Syllable Language Models by N-gram Size .....	92
Table 4.3: Mean Errors for Syllable Language Models by N-gram Size .....	96
Table 4.4: Mean Improvement By Category Across N-gram Sizes 2, 3, 4 .....	97
Table 5.1: WANT Equivalence Class.....	109
Table 5.2: Errors for Concept Language Model .....	113
Table 5.3: Mean Errors Experiment 2, Syllable Language Model .....	113
Table 5.4: The BE component of the Concept Model .....	116
Table 5.5: Mean Errors Experiment 3.2, Concept Language Model .....	123

## Preface (and an Apologia)

Many years ago, I studied with the great rhetorician, Walter J. Ong<sup>1</sup>. I was introduced to the peculiar—and to me unnoticed—distinction between oral and written work. Consider the very beginning of the catalog of the captains from a prose translation of Book II of the *Iliad* (Iliad, 2010):

Peneleos, Leitus, Arcesilaus, Prothoenor, and Clonius were captains of the Boeotians. These were they that dwelt in Hyria and rocky Aulis, and who held Schoenus, Scolus, and the highlands of Eteonus, with Thespeia, Graia, and the fair city of Mycalessus. They also held Harma, Eilesium, and Erythrae; and they had Eleon, Hyle, and Peteon; Ocalea and the strong fortress of Medeon; Copae, Eutresis, and This be the haunt of doves; Coronea, and the pastures of Haliartus; Plataea and Glisas; the fortress of Thebes the less; holy Onchestus with its famous grove of Neptune; Arne rich in vineyards; Midea, sacred Nisa, and Anthedon upon the sea. From these there came fifty ships, and in each there were a hundred and twenty young men of the Boeotians.

Or how about the opening lines of Genesis in the *King James Bible* (Genesis, 2010, 1-9):

In the beginning God created the heaven and the earth.  
And the earth was without form, and void; and darkness was upon the face of the deep. And the Spirit of God moved upon the face of the waters.  
And God said, Let there be light: and there was light.

---

<sup>1</sup> Ong is perhaps best known for his study of Peter Ramus (1515-1572) whose pedagogical theories are associated with the rise of lecture-style education and the decline of Socratic dialogue. Ong links dialogue with pre-print cultures and lecture with the invention of moveable type (and other things, of course) (Ong, 1958).



And God saw the light, that it was good: and God divided the light from the darkness.

And God called the light Day, and the darkness he called Night. And the evening and the morning were the first day.

And God said, Let there be a firmament in the midst of the waters, and let it divide the waters from the waters.

And God made the firmament, and divided the waters which were under the firmament from the waters which were above the firmament: and it was so.

The *Iliad* and the *Bible* were produced in pre-literate cultures and both, even in translation and printed, retain rhetorical features that are distinctive of all such cultures. These include a sense of redundancy, the repetition of what has already been said both in manner and matter. There are many other characteristics of oral cultural productions, all of them shared by peoples across the globe, and all of them contributing to the “poetic” feel we associate with the *King James Bible* (for example). What often gets lost, however, is that the familiar *Iliad* and *Odyssey*, the *King James Bible*, and *Beowulf*, along with the scores of other ancient cultural productions that have survived, were *spoken*, not just read aloud—because writing had not been invented—but spoken. Though this seems obvious enough, at the time it was a revelation to me, because print and silent reading and the particular analytic outlook fostered by print had been privileged over all else in my education (Ong, 1982).

Much later, when I was studying computer science as a graduate student, I happened upon what I take to be a little-known article, by a self-defined “freelance linguist,” Donald Hindle (1983). His article argues, mostly on discourse-related grounds, that speech and writing are different phenomena. Though I did not have a clear notion of speech in those

days, I probably thought it simply amounted to unedited writing. To take one of many examples, Hindle observes the relative frequency in speech over writing of pronouns without a distinct referent (“You never know!,” “It’s ridiculous that he said that,” p. 82).

I filed away Ong’s enormous body of work and Hindle’s suggestive article until I began thinking about automatic speech recognition (ASR) in the last several years. Though I assume that the many linguists and linguistically-trained computer scientists who have developed the field understand the now relatively uncontroversial observation that speech and writing differ in substantial ways, it is sometimes not apparent in the research itself. Here, for example, is the National Institute of Standards and Technology (NIST) describing one of its projects: “The Rich Transcription evaluation series promotes and gauges advances in the state-of-the-art in several automatic speech recognition technologies. The goal of the evaluation series is to create recognition technologies that will produce transcriptions which are more readable by humans and more useful for machines” (NIST-RT 2008).

Even granting that the project being described is transcription itself, a quick look at any “correct” transcription against which an automatic speech recognition system is scored, shows a distinctive print bias.<sup>2</sup> It is often a transcription of a particularly hygienic variety. Consider, for example, the transcribed reference files found in Appendix D that supposedly match the audio files used in this study and described in full later.<sup>3</sup> They have been scrubbed by their makers of all disfluencies. They have, in fact, the characteristics of the kind of

---

<sup>2</sup> Instead of the cumbersome “automatic speech recognition system” or (to be introduced later), “large vocabulary continuous speech recognition system,” I will use the term “recognizer” to refer to software that takes speech as input and produces text as output. For now, that text output is word strings. Later, we will extend the definition of text to include syllable strings and collections of words that I will call *concepts*. See Sections 4 and 5.

<sup>3</sup> These were obtained from Next It (Next It, 2008). See Appendix D for a description of how the recordings were made.

edited interviews that one finds in newspapers and magazines. Here, for instance, is a “33-year-old business woman” speaking to a reporter from *The New York Times*: “We have never seen anything like this in our history. Even the British colonial rule, they stopped chasing people around when they ran into a monastery” (Sang-Hun, 2007, p. 1). Though it would be nice to have a recording and transcription of the actual interview, we can get a sense of what the reporter left out (and put in) by looking at any hand-transcribed corpus of spontaneous speech. Here is the very first segment from the Buckeye Corpus (Pitt, 2006):

yes <VOCNOISE> i uh <SIL> um <SIL> uh <VOCNOISE> lordy <VOCNOISE>  
um <VOCNOISE> grew up on the westside i went to <EXCLUDE-name> my  
husband went to <EXCLUDE-name> um <SIL> proximity wise is probably within a  
mile of each other we were kind of high school sweethearts and <VOCNOISE> the  
whole bit <SIL> um <VOCNOISE> his dad still lives in grove city my mom lives  
still <SIL> at our old family house there on the westside <VOCNOISE> and we  
moved <SIL> um <SIL> also on the westside probably couple miles from my mom.<sup>4</sup>

It is reasonable to assume that a hand-transcription of the acoustic signal that the 33-year-old business woman generated would look something like the Buckeye transcription.

My early training and these simple observations about speech and writing are relevant here not because I intend to offer further evidence that speech and writing differ. Ong’s enormous *oeuvre* and more recent studies in corpus and usage-based linguistics have rendered this observation uncontroversial, at least in certain circles (Tomasello, 2003). My early training is relevant because it made me receptive to observations that I attribute to Drs.

---

<sup>4</sup> <VOCNOISE> is an unidentified sound. <SIL> is a period of silence.

Luger and Wooters, namely that if we relax the constraint that the output of speech recognition be human useable, effectively relaxing the bias towards the written word found throughout computational linguistics, we might be able to generate accurate accounts of the speaker's intent to be passed off to a knowledge-based systems (see Section 1) for further processing.<sup>5</sup>

In fact, transforming speech into strings of words is not necessary to emulate human performance in many contexts. Once we replace the constraint that the output of a recognizer be a string of words, then two hypotheses suggest themselves. The architecture of current recognizers usually have a component known as a language model (see Section 1 for a detailed explanation). This is a collection of probabilities of word sequences drawn from a given corpus. A language model records the fact that the words “to run” are more probable “to to” in most corpora. The first hypothesis is that a language model that contains syllable sequence probabilities will perform better than a word-based language model. This is the subject of Sections 3 and 4. The second hypothesis is that the addition of a post-processing component to handle collections of words and phrases that mean the same thing--call them

---

<sup>5</sup> That bias could be as central to computational linguistics for practical reasons as the well-known physical symbol system hypothesis once was to artificial intelligence: “A physical symbol has the necessary and sufficient means for general intelligent action” (Newell and Simon, 1976, p. 116). A physical symbol system will be familiar to any linguist in the generativist tradition. It is a collection of symbols operated on by a collection of processes to produce collections of still other symbols. This sounds an awful lot like an abstraction of a computer or even a context-free grammar (which, of course, it is) or even like mathematics itself. Begin with some symbols, apply some rules, get still other symbols. Begin with speech, digitize it, apply some transformations—this time probabilistic—and words come out the other side. This is not to argue that computational linguists have been short-sighted, only that they may have been influenced both by the tools at hand and the intellectual tradition that generated these tools. To be fair, a perfect transcription of an acoustic signal could serve as the limiting case in speech recognition. Researchers do not insist on transcriptions because they don't understand the difference between speech and writing, but because it is a standard against which all recognizers can be judged. I owe this observation to Charles Wooters, personal communication.

*concepts*--will yield still better results.<sup>6</sup> This is subject of subject of Section 5. As an example, an utterance like (1) is syllabified in (2) and conceptualized in (3):

I want to fly to Spokane (1)

ay w\_aa\_n\_td t\_uw f\_l\_ay t\_uw s\_p\_ow k\_ae\_n (2)

WANT GO s\_p\_ow k\_ae\_n (3)

Now, (3) is not especially usable in its present form and the substitution was accomplished under very rigid guidelines. Both topics are considered in Section 6, where the broad outlines of a probabilistic substitution technique based on part-of-speech tagging is provided.

Although what is being proposed in this study is an engineering artifact, there is no reason why it should not be inspired by human performance. This is the thin line that all builders of artifacts must tread. To use a well-worn example, airplanes have wings for the same aerodynamic reasons that birds do. But they need not flap, or be covered in feathers. Humans, it appears, do a reasonable—though far from perfect—job of recognizing speakers from three seconds of telephone speech (Schmidt-Nielsen & Crystal, 2000). Producing a text transcription appears to be a considerably more difficult task. In fact, the very notion that each of us does for our conversational partners what *The New York Times* reporter did for his informant is undercut by the growing body of work in functional and usage-based linguistics.

This is the inspirational part. We humans do not, probably cannot, transcribe what our conversational partner just said. As Walter Ong observed (in many forms over his long career), one might be able to recall a sound—call it back—but there is no place to look for it. Sounds exist only as they are going out of existence. Unlike the written word which is fixed,

---

<sup>6</sup> For now, let's defer a definition of "mean," and agree that when talking to a machine, "I want," and "I would like," mean the same thing.

sounds are events that occur, a fact that is codified in at least one language. The Hebrew *dabar* carries both the sense of “word” and “event” (Ong, 1984). We can, however, provide a paraphrase, something that the notion of a concept is intended to simulate.

Speech recognition in its current form rests heavily on the use of Bayes’ Theorem and its use for computing the probabilities of sequences of events with hidden Markov models. Appendix A is a primer on probability theory, leading up to a derivation of Bayes’ Theorem. Appendix B contains an introduction to finite state automata, hidden Markov models, and dynamic programming, all techniques used in speech recognition. Finally, Section 2 considers previous work on the use of the syllables and concepts, along with a short introduction to the underappreciated—by engineers, in any case—syllable.

All input data used in the study can be found in Appendices C through E. All output data can be found in Appendices F through H. The software developed for this study can be found in Appendix I. Appendices J and K contain the words used in the study along with their machine-made syllabifications and phonetic transcriptions. Appendix L contains the phonetic symbol set employed by the recognizer used in this study.

## 1.0 What is Automatic Speech Recognition and Who Uses It?

Researchers in the field of automatic speech recognition (ASR) differ on the scope of the field. Frederick Jelinek, one of the pioneers of statistical speech recognition, the technique used in this study, says that a “speech recognizer ... can be thought of as a voice-actuated ‘typewriter’ in which a computer program carries out the transcription and the transcribed text appears on the word display” (Jelinek, 1997, p.1). Jurafsky and Martin (2008) tell us more precisely that “the task of speech recognition is to take an acoustic waveform and produce as output a string of words” (p. 287). Finally, Rabiner and Juang (1993, p. 1), authors of an early and widely used text on speech recognition, take a considerably broader approach, citing HAL of *2001* and R2D2 of *Star Wars* as exemplars of what researchers would like to produce. They acknowledge, of course, “that we are far from achieving the desired goal of a machine that can understand spoken discourse on all subjects by all speakers in any environment.”

For the purposes of this study, we will assume that the goal of ASR is to map an acoustic signal to some sort of textual output. We use the qualifier “some sort” because that textual output is usually assumed to be word strings, where a string is just a sequence of alphabetic characters. But, in fact, as we will see, speech recognition systems as currently constructed are indifferent as to whether these sequences of characters are strings of words or strings of anything else that can be represented by characters. The hypothesis that drives this study is that if we construct a recognizer to produce strings of sub-word units, that we loosely call *syllables*, or strings of meta-word units, that we loosely call *concepts*, we can improve the accuracy of the recognizer. The words *syllables*, *concepts*, and *accuracy* all will

be defined more precisely as we proceed. For now, we can think of a syllable, broadly, as some principled division of a word, a concept, as a collection of word or words that mean roughly the same thing, and accuracy as adherence of textual output to the acoustic signal.

Acknowledging with Rabiner and Juang that we are not close to producing HAL, one might reasonably ask of what possible use are current systems, these way stations on the road to presumed human-like recognition and comprehension. As it happens, there are many uses for such systems. Most of them fall under the category of human-computer interaction or HCI. It was not long ago that only mechanism for a user to communicate his or her wishes to computer was through long strings arcane commands entered at a keyboard. Even this, in the early 1970s was a dramatic improvement over the same arcane commands tediously punched onto decks of cards and entered into a card reader. These card readers were in turn enormously more efficient than reserving time in the computer room and physically setting blocks of switches for each program to be run (Tanenbaum and Woodhull, 2006). There is no reason to think that contemporary point and click interfaces, augmented with textual input, from a computer usability point of view, are the final word.

The most obvious community to benefit from voice-augmented HCI is the sight-impaired for whom it might be argued that the evolution from textual input to point and click devices has made interacting with a computer somewhat more difficult. The paradigm case for a point and click user is the person sitting at a desk. But this does not exhaust the possibilities. Anytime a computer user's eyes or hands are not available, as in situations where equipment or objects have to handled, while using a computer, the point and click model has clear limitations. From automobile mechanics, to industrial device operators, to medical technicians, to airplane pilots, to surgeons, any application for which the user



requires hands and eyes while using a computer, might be improved by speech recognition (and synthesis, of course).

Telephony is another class of computer applications that could be improved by speech recognition. Though as clients we might look fondly on the days before automated call centers (“Press 3 for billing”), these call centers are improved, or at least augmented, when the phone trees accurately respond to voice activation. The introduction of web-based travel services a decade ago has resulted in a generally poor outlook for travel agency as a career (Bureau of Labor Statistics, 2010). Still more changes are in the works. An improvement on the point and click travel agent appears to be one augmented with the capacity to respond to textual input. Next It Corporation, the company that sponsored the beginnings of this study, has developed an interface for Alaska Airlines that uses keyboard input of spontaneous questions to augment a point and click interface (Next It, 2010). A voice interface would further augment the service. An interface that combines point and click devices with voice has been shown to be considerably more efficient than point and click alone in one case study (Cohen et al., 1998).

Perhaps the application we might be most familiar with is dictation. Dictation is the text transcription of an extended monologue by a single speaker. Long before there were ASR systems, there were amanuenses who quite laboriously performed this task. The best-known are probably Milton’s daughters who listened and transcribed their father’s composition of *Paradise Lost* (Jurafsky and Martin, 2008). More recently, the novelist Cynthia Ozick, in her novella, *Dictation*, has imagined a meeting between the amanuenses of two early twentieth century novelists, Joseph Conrad and Henry James, whose repetitive stress injuries rendered writing painful (Benfey, 2008). The leading ASR in this area has

been the software suite *Dragon Naturally Speaking*. The drawback, however, is that such systems must be trained to a single speaker's voice, a constraint that we will remove for the current study—with an attendant drop in accuracy.

Perhaps most surprising, given the history of funded research for ASR, is that not a single one of the books consulted for this brief overview of uses of the technology mentions surveillance (Jelinek, 1997; Jurafsky and Martin, 2008; Holmes and Holmes, 2001; Huang et al., 2001; Rabiner, 1993). As we will see later, funding from the Defense Advanced Research Products Agency (DARPA) helped give ASR its current shape<sup>7</sup>. Although some in the ASR community might not like to admit it, surveillance organizations around the world are potential beneficiaries of the product. According to a 2007 report from the Congressional Research Service, “Whereas some observers once predicted that the NSA was in danger of becoming proverbially dead due to the spreading use of encrypted communications, it appears that NSA may now be at greater risk of being ‘drowned’ in information” (as cited in Bamford, 2008, p. 2). We could go into this in detail with suitably dazzling numbers, but the point is that though the NSA has been much in the news in recent years because of warrantless wiretapping during the Bush administration (Risen and Lichthblau, 2005), what has not been in the news is exactly how the NSA would use the results of the taps. As Bamford (2008) points out, the NSA's data has outgrown its headquarters in the Baltimore-D.C. area. It recently built a 470,000 square foot data warehouse in San Antonio, Texas (not far, interestingly, from a Microsoft server farm that processes the full range of internet

---

<sup>7</sup> It must be pointed out that this is true of computing generally. To cite a well-known example, the protocols that enable the Internet to operate were developed through DARPA funding in the 1970s. The result was ARPANET, a military communications system whose decentralized nature was supposed to make it resistant to nuclear attack.

communications). This amount of data is beyond the capacity of humans to process.

Though, of course, the workings of the NSA are classified, it is easy to imagine the agency employing voice recognition software to scan phone calls looking words indicating a threat to national security.

## 1.2 Classifying the Speech Recognition Task<sup>8</sup>

Different tasks require different kinds of speech recognizers and perform with different levels of accuracy. As of 2006, the best ASR systems (or combinations of systems as we will at the end of this chapter) had error rates ranging from .5% to 20%.<sup>9</sup> One parameter in speech recognition is whether the system is intended to recognize the speech of a single speaker on whose voice it has been trained versus an arbitrary speaker of either gender, with his or her own regional dialect and personal idiolect. Another parameter is the size of the vocabulary. ASR performs quite well on the digits task where the system is asked to recognize sequences drawn from the ten digits. It performs less well as the vocabulary grows. As of this writing (2010), a large vocabulary system, the type considered in this study, have vocabularies from 20,000 to 60,000 words. A third parameter is whether the speech is isolated or continuous. The recognition of isolated words, surrounded by pauses, is easier for recognizers than continuous speech where one word blends into the next. A fourth parameter is whether the speech is read, human-to-machine, or conversational. Speech that is read, say the text of the *New York Times* or part of a human-computer dialog, tends to be easier to recognize than conversational speech between two or more humans, since read speech tends to have been cleansed of filled pauses, restarts, and other disfluencies. Humans

---

<sup>8</sup> This section is drawn from Jurafsky and Martin (2009).

<sup>9</sup> The standard measure of error is the word error rate (WER), defined in Section 1.8.

talking to machines, as we shall see later in this study, talk more slowly and distinctly than they do to one another. A fifth parameter is mode of environment and mode of input. Speech input in a quiet environment using a head-mounted microphone is easier to recognize than speech spoken into a cell phone on a busy street.

The speech recognition architecture described in Section 1.3 and used in experiments described in later sections is applicable to large vocabulary continuous speech recognition (LVCSR) systems that are developed and trained independently of any single speaker. It depends crucially on stochastic techniques like hidden Markov models (HMM), Bayesian inference and an algorithmic technique known as dynamic programming. These are discussed in Appendixes A and B.

### **1.3 Thoughts on “Normal Science”**

Research into automatic speech recognition has a long history, dating to the earliest days of computing. It began roughly at the same time that researchers were developing compilers for the first high-level programming languages. Bell Labs, RCA research, and MIT’s Lincoln Labs all used developing ideas in acoustic-phonetics to work on the recognition of single digits, syllables, and certain vowel sounds. Work continued through the 1960s in the United States, Japan, and the Soviet Union, using pattern-recognition techniques. This work received a significant boost through advances in digital signal processing techniques in the 1970s. In all cases, however, the effort was to develop systems that could recognize single words.

Two developments in the 1980s gave systems their modern shape. The first was Defense Advanced Research Products Agency (DARPA) funding for research into large

vocabulary continuous speech recognition. The second was the adaptation of statistical techniques, most notably hidden Markov models (HMM)<sup>10</sup>, to the speech recognition problem (Rabiner & Juang, 1993; Jurafsky & Martin, 2000; Jurafsky & Martin, 2009).

Nevertheless, despite a half-century of research, despite massive Department of Defense funding, despite the obvious rewards available to those who solve (and patent) the speech recognition problem, despite enormous strides made through the application of statistical modeling techniques, LVCSR is still an open research problem. Performance levels, as of the late 90's "were still at least an order of magnitude worse than that exhibited by a human being" (Moore, 1998, p.7). Ten years later, SONIC, a speech recognition system developed at the University of Colorado's Center for Speech and Language Research, and the one used in this study, posted a word error rate of over 40% when presented with continuous, unconstrained conversational speech from the SWITCHBOARD corpus (Godfrey, et al., 1992; SONIC, 2007).

It is important to note that these results are not the best possible. The National Institute of Standards and Technology (NIST) has sponsored meetings for several years in which a variety of recognizers are tested in controlled settings. At its 2004 meeting, NIST evaluated several systems against both relatively sanitized news broadcast news speech (BN) and conversational telephone speech (CTS) drawn from the Fisher corpus (Cieri et al., 2004). A hybrid system composed of pieces drawn from research recognizers developed at Cambridge University and the research labs, SRI, BBN and LIMSI (*Laboratoire d'Informatique pour la Mécanique et les Sciences de l'Ingénieur*) posted a word error rate of

---

<sup>10</sup> Hidden Markov models are central to modern speech recognition. Appendix B is a short description for those unfamiliar with the technique.

11.6% on the BN task. A hybrid system drawn from components developed at IBM and SRI posted a word error rate of 14.9% on the CTS task. These results were not achieved in real time, roughly at the same rate as human processing. The BN results required up to ten times real time, while the CTS results needed up to 20 times real time. Though the results drop considerably when the systems are constrained to perform in time less than or equal to real time (15.3 % BN and 19.0% CTS), these hybrid systems perform considerably better than SONIC (Fiscus, et al., 2004).

The word error rate (WER) percentages refer to three kinds of errors: insertions, deletions, and substitutions. So, in an utterance of twenty words, the real-time performance of the hybrid system would have about four insertions, deletions, or substitutions. SONIC would have about eight. The scoring software was developed at NIST. While recognizing that WER is not a perfect metric—it treats the deletion of an open class word, like a noun or verb, as no more serious than the deletion of a closed class word, like a preposition or determiner—WER is the standard by which a system is judged. Jurafsky and Martin (2008) consider several other techniques especially appropriate for dialog systems. We consider WER in some detail in Section 1.07.

Despite its less than perfect performance in the most general of domains—large vocabulary continuous speech uttered by arbitrary speakers—ASR has begun to take on the characteristics of what Thomas Kuhn has called “normal science.” “Normal science,” according to Kuhn “means research firmly based upon one or more past scientific achievements, achievements that some particular scientific community acknowledges for a time as supplying the foundation for further practice” (Kuhn, 1970, p. 10). The contours of a contemporary normal science are summarized in textbooks that “expound the body of

accepted theory, illustrate many or all of its successful applications, and compare these applications with exemplary observations and experiments” (p. 10). ASR has a number of standard textbooks. Early on the scene were Rabiner and Juang (1993), and Jelinek (1997). These were followed by Holmes and Holmes (2001), by the classroom-directed Jarafsky and Martin (2000 & 2008), and, most comprehensively, by Huang et al. (2001). Indeed much of the information (not the selection, exposition, organization, and expression, of course) for this introductory section comes from these texts.

One striking characteristic is just how similar all of the books are, right down the equations used to describe the process of mapping an acoustic signal to word strings. The clear implication is that there is general agreement on the approach to ASR and this agreement was in place as early as 1993. This implicit pact, now nearly two decades old, runs counter to the popular notion that advances in computing change with breakneck speed. It is also a scientific illustration of that old standby called the *law of diminishing returns*. Early breakthroughs yield the greatest rewards, but further applications of technology (or fertilizer or labor power) yield increasingly reduced returns. At this juncture, even a 2% reduction in error rate in an ASR is cause for celebration.<sup>11</sup>

There is another consequence of a discipline maturing into a normal science. The mechanism for communicating results gradually shifts from books addressed to the educated reader—Darwin’s *Origin of the Species* is Kuhn’s example—to very short articles on increasingly subtle aspects of the field, addressed only to other researchers who, indeed, are the only ones able to read them. So, most of the articles cited in this study are conference papers, sometimes not much more than a half-dozen pages in length. Full-length articles tend

---

<sup>11</sup> This observation comes from a conversation with Dr. Fan Yang of Next It Corporation.

to be archival, and books on ASR addressed to a general audience—even a general audience of computer scientists or phonologists or phoneticians or electrical engineers—simply do not exist. The closest approximation is Jurafsky and Martin (2008), but even that textbook assumes a substantial background and where it does not, leaves one with the unsatisfying feeling that complexities have been glossed over.

One of the underplayed observations in Kuhn’s work is that the outcome of normal scientific experiments is often anticipated. “As was anticipated,” is a familiar phrase in experimental work. To test this, we searched Google Scholar on 3/2/2010 for the phrase, “as was anticipated.” We were rewarded with 5,890 hits. Not all of them are from scientific articles, of course, nor do all report experimental data. Yet enough do to persuade us of Kuhn’s observation. A motivating hypothesis behind the current study is that ASR will achieve better results if we replace the statistical arrangement of words from in most recognizers with a statistical arrangement of syllables. There are several reasons to do this. Though, as we discuss later, the number of syllables of English is contested, no one contests that it is considerably less than the number of words. Further, the syllables in a language—again as we discuss later—tend to form a closed set, at least when compared to the set of words which gains (and loses) members rather rapidly. From the point of view of ASR, this makes for a smaller and steadier target. At the start of this study, we anticipated that the changes just described would produce better results. In fact, they did, as will be apparent Section 4.

Then why conduct the experiments? The immediate response is that, though we anticipated the results, anticipation is not data. We wanted to make sure. A less obvious response is found in Kuhn: “Though its [the experiment’s] outcome can be anticipated, often



in detail so great that what remains to be known is itself uninteresting, the way to achieve that outcome remains very much in doubt. Bringing a normal research problem to a conclusion is achieving the anticipated in a new way, and it requires the solutions of all sorts of complex instrumental, conceptual, and mathematical puzzles” (Kuhn, 1962, p. 36). The software included in Appendix I required almost two years to develop. As we will see later, several researchers have experimented with aspects of the parameters considered in this study. No one, to our knowledge, has considered all of them and in the arrangement that we have developed. The construction and execution of the experiments presented in this study required “the solutions of all sorts of complex instrumental, conceptual, and mathematical puzzles.”

Though the results were anticipated, they are considerably more robust than was expected. The observation that led to the experiments is a simple one. In certain very constrained, but very widely occurring, circumstances, the full range of natural language, evolving as it did to solve the full range of human communication problems, is redundant. Moreover, its redundancy expresses itself as a complexity familiar and joyous to any linguist, but seemingly impossible to specify sufficiently for computer modeling. This was not always obvious, of course. The seventy year history of computer processing of natural language is filled with fits and starts, some as naïve as imagining that translation is simple dictionary lookup, others, influenced by the generative paradigm, worked with symbols, logic, and pattern matching, still others, as early as the 1940s and 1950s hit upon the statistical paradigm that now dominates speech recognition and machine translation (Jurafsky and Martin, 2008; Luger, 2009; Koehn, 2010).

Twenty years ago, the speech recognition component of the computational linguistics community agreed upon stochastic modeling as the dominant paradigm for the field. Though this paradigm has resulted in enormous advances, the problem of how to map an acoustic signal to words is still far from solved. The approach advocated here is that the problem as stated is too constrained for many applications. Researchers in the field have taken the approach that if they can map an acoustic signal to words, surely the limiting case, they can map it to other strings as well.<sup>12</sup> This allows them to compare systems without respect to applications. Thus, the results that are presented in Sections 3, 4, and 5 require that the output of the recognizer be in a format not directly useable by humans. One finds in computing that there is always a price to be paid. The price we must pay to reduce the error rate by over 14% (see Section 4) is to have output that must undergo further processing to be useful. This might take the form of an expert system post-processor, a subject considered in Section 6, “Conclusions and Further Research.”

#### **1.4 Automatic Speech Recognition As Cryptanalysis**

It may help to clarify the process of ASR by showing its relationship to a human enterprise that dates back at least to the time of the Greeks, cryptography and cryptanalysis. At the very highest level it is possible to conceive of the speech recognition task as one of communicating over a noisy channel. The acoustic waveform is treated as a noisy, i.e., partially scrambled, version of a pre-existing string of words. The task is to build a model of the channel so we can figure out how the input was scrambled. This will let us recover, from the set of all possible output the strings, the one that best matches our source (Jurafsky and Martin, 2008). There are two problems here. The first is that we will need a metric that lets

---

<sup>12</sup> I owe this observation to Charles Wooters in a personal communication.

us judge the goodness of fit between the input and the output. The second is that we cannot examine all possible outputs but must restrict ourselves only to those that have a reasonable chance of matching our input. This bears a high level relationship to cryptography.<sup>13</sup> Arab mathematicians in the 9<sup>th</sup> century discovered how to crack simple substitution ciphers<sup>14</sup> based on the known frequencies of letters in the alphabet and the 19<sup>th</sup> century mathematician Charles Babbage developed a statistical technique to attack the so-called Vignere polyalphabetic cipher which had stood unbroken for three centuries (Trappe and Washington, 2006; Singh, 2000).

Though frequency analysis does not always work, the one truly unbreakable cryptosystem, known as the One Time Pad is unbreakable precisely because the encrypted text provides no information about the plain text. Without going into detail, the intuition is still obvious. A cryptographic key is part of the mechanism by which the message is encrypted. In the Caesar cipher, for example, the key is the alphabet obtained by rotating the conventional Roman alphabet two positions to the right. Users of the One Time begin by randomly<sup>15</sup> generating a key as long as the message itself. They then transform the key and

---

<sup>13</sup> I owe this observation to Knight (1999).

<sup>14</sup> Substitution ciphers are those where letters are substituted for one another in some principled fashion. One of the best-known and earliest substitution ciphers is the Caesar cipher where letters two positions to the right are substituted for the letter itself. Thus A becomes C, B becomes D, and Z becomes B by wrapping the alphabet. It was attributed to Julius Caesar by his second century biographer, Suetonius.

<sup>15</sup> The idea of computer-generated randomness is itself problematic. Computers, like all machines, are deterministic devices. Identical inputs under identical circumstances yield identical outputs. This is a problem when attempting to generate random sequences using a computer. The solution is to construct a pseudo-random number generator seeded by some number, say the elapsed time since 1970, that will not occur again. The sequences that are generated are not truly random. They will reoccur given the same seed value. This is a problem, of course, in high stakes cryptography since, if the opponent knows the algorithm that generated the seed value, she can generate seeds and candidate pseudo-random number sequences. A more secure method is to use algorithm-based methods from number theory that are thought to be intractable. The Blum-Blum-Shub pseudo-random bit generator is one such method. See Trappe and Washington (2006) for further discussion.

message into their component bits using the ASCII table<sup>16</sup>. The final step is to perform a bit-wise exclusive operation on each of the bits (using the rules:  $0 \text{ xor } 0 = 0$ ,  $0 \text{ xor } 1 = 1$ ,  $1 \text{ xor } 0 = 0$ ,  $1 \text{ xor } 1 = 1$ ) to produce the encrypted text. Recipients of the encrypted message, through a prior agreement, are in possession of the key used in encryption. They perform the same bit-wise exclusive or operation on the encrypted text. This produces the original message. The security comes from both randomly generating the key—that is encoding no information about the message—and discarding the key after a single use. The moral of the story is that information in the form of a probabilistic relationship between the message and the encrypted text must be present for cryptanalysts to gain a toe-hold.

The architecture of the generic automatic speech recognition system to be considered below, rests heavily on a classic technique used to attack cryptographic systems known as the *chosen plaintext attack*. In this attack, cryptanalysts have brief access to the encryption device. Though the key cannot be produced, they encrypt many carefully chosen plaintexts, using them to deduce the key itself. A technique very similar to this one, known as the *known plaintext attack*, was used by the Allies in the Second World War to crack the German Enigma cipher. In the case of speech recognition, as we will see below, a component of the recognizer known as the acoustic model has at its disposal a large sample of encrypted, that is to say recorded, speech along with a transcription of that speech. Using these two items, the probabilities of various acoustic patterns given a word string are computed. The recognition stage hypothesizes a string of words based on the input acoustic signal.

---

<sup>16</sup> The ASCII table is a 256 member set of encodings, eight bits to the character, for all characters on the standard keyboard plus many others.

## 1.5 A First Pass at Bayesian Inference and Automatic Speech Recognition<sup>17</sup>

Continuing with the cryptanalytic metaphor and stated somewhat more precisely, ASR allows us to answer the following question: “What is the most likely string of words of all word strings in the target language given some acoustic input.” As noted in the last section, this can be decomposed into two related problems. The first is that we will need a metric that lets us judge the goodness of fit between the input and the output. The second is that we cannot examine all possible outputs but must restrict ourselves only to those that have a reasonable chance of matching our input. Suppose we treat the acoustic input as a sequence of individual acoustic observations, the mechanism for which will be provided in Section 1.7:

$$O = o_1, o_2, \dots, o_t \quad (1.1)$$

In the same fashion, we can treat output—a sentence, let us suppose—as a string of words,

$$W = w_1, w_2, \dots, w_n \quad (1.2)$$

As we will see, the idea that an ASR produces a string of words is more constraining than is necessary. Instead of speaking of the output as words, the practice in most expositions of ASR, we will instead speak of the output as a string of symbols drawn from some set of legal symbols, with the sole constraint that the symbols be text-based, that is encoded using the ASCII table. Thus (1.2) becomes:

$$S = s_1, s_2, \dots, s_n \quad (1.3)$$

---

<sup>17</sup> The speech recognition architecture described in this and the following sections has been in place since the late 1980s. Every book consulted on the subject is remarkably similar right down to the equations used. These include Holmes and Holmes, 2001; Huang, et al., 2001; Jelinek, 1997; Jurafsky and Martin, 2006; Jurafsky and Martin, 2009; Koehn, 2010; and Rabiner and Juang, 1993. The presentation most closely resembles Jurafsky and Martin’s two editions, since these are the texts that the author first used to investigate speech recognition. Nevertheless, the generalization from words to symbols is my own and is at the heart of this study.

In probabilistic terms, ASR tries to find the most probable string of symbols given an acoustic observation:

$$\text{hyp}(S) = \underset{S \in L}{\operatorname{argmax}} P(S|O) \quad (1.4)$$

where the subscript  $S \in L$  simply means that  $S$  is one of the  $L$  hypothetical strings generated by the recognizer. Equation 1.4 is read, “The hypothesized symbol string is the one with the greatest probability given the sequence of acoustic observations.”

The next task is to make 1.4 operational. Specifically, we need to find a way for a given sequence of symbols  $S$  and a sequence of acoustic observations  $O$  to compute the probability  $P$  of  $S$  given  $O$ <sup>18</sup>. To do so, we invoke Bayes’ Theorem<sup>19</sup>:

$$p(x|y) = \frac{p(y|x)*p(x)}{p(y)} \quad (1.5)$$

$p(x)$  is commonly referred to as *the prior probability of  $x$* .  $p(y|x)$  is referred to as *the likelihood of  $y$  given  $x$* .

Rewriting 1.4 using 1.5 gives:

$$\text{hyp}(S) = \underset{S \in L}{\operatorname{argmax}} \frac{P(O|S)*P(S)}{P(O)} \quad (1.6)$$

Equation 1.6 maximizes over all possible symbol strings. But notice that  $P(O)$  does not change. Our acoustic observation is our acoustic observation, no matter the potential string of symbols. Since we don’t care about the probabilities themselves, but only the symbol string with the highest probability,  $P(O)$  can be eliminated, leaving us with Equation 1.7 which is far easier to compute:

---

<sup>18</sup> Readers unfamiliar with probability theory will find a tutorial in Appendix A. The discussion of conditional probability in section A.2 is especially relevant for ASR.

<sup>19</sup> Bayes’ Theorem is derived in Appendix A.

$$\text{hyp}(S) = \operatorname{argmax}_{S \in L} P(O|S) * P(S) \quad (1.7)$$

In world of Bayesian inference  $P(O/S)$  is referred to as the *likelihood probability* and  $P(S)$  is called the *prior probability*. In the world of ASR, the terms also have names.  $P(O/S)$  is referred to as the *acoustic model*. It expresses the probability that a string of symbols would be associated with an acoustic signal.  $P(S)$  is called the *language model*. In the more constrained case of words, it expresses the probability that a sequence of words would be uttered by a speaker of English (for example). In the case of symbols, it simply expresses the probability of symbol string appearing in our language, L, however we have chosen to define it. Finally,  $\operatorname{argmax}$ , the function that allows us to choose the most probable sequence of symbols from the candidate products in Equation 1.6 is called the decoder. In practice this is the Viterbi algorithm<sup>20</sup>. The present study is concerned only with adjustments to the language model, the first of which has already been hinted at. Instead of word sequence probabilities, we will populate our language model with syllable sequence probabilities. Both the acoustic model and the Viterbi algorithm are treated as black boxes.

## 1.6 Why ASR is Hard

The problem for which ASR systems are designed to solve is hard. Its solution—providing an accurate mapping from acoustic input to textual output for any speaker of a given language under any circumstances—has eluded researchers despite a half century of well-funded effort. The difficulty goes beyond the problem domain to range of tools that are

---

<sup>20</sup> A tutorial on dynamic programming, of which the Viterbi algorithm is an instance, is provided in Appendix B.

customarily needed for its solution. ASR is interdisciplinary in nature. Its current expression requires at least passing familiarity with results in the following fields<sup>21</sup>:

- Signal Processing: the mechanism for extracting information from an acoustic signal
- Acoustics: the underlying relationships between the human vocal tract and the speech signal as expressed in wave mechanics
- Information Theory: techniques for measuring the goodness of various statistical models
- Linguistics, especially phonetics/phonology: the underlying relationships between sounds in a language
- Mathematics: Probability and statistics, differential equations (in the case of wave mechanics), and number theory (in the case of cryptography to which ASR is closely related)
- Computer Science: the study of algorithms and particularly techniques for efficiently implementing the theoretical results from the above disciplines in hardware and software.

This range of knowledge is far beyond the capacity of a single person. The usual approach, certainly the one adopted in this study, is to acquire an acquaintance with all of the disciplines and then to concentrate on the handful of techniques used to investigate a single area. So, as noted above, this study is an investigation of the language model through three experiments with variations. They require knowledge of Bayesian inference techniques,

---

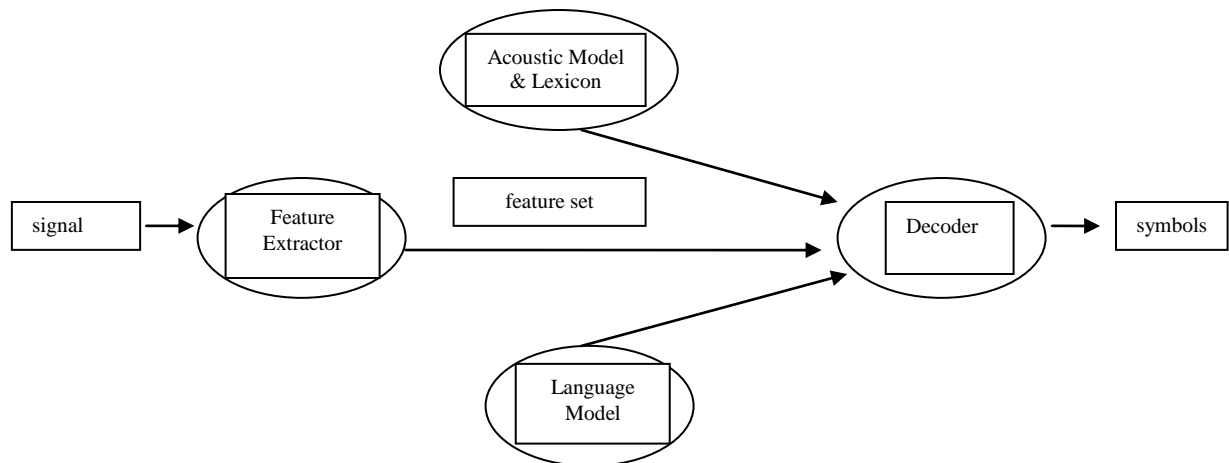
<sup>21</sup> I owe these observations to Rabiner and Juang (1993).



familiarity with the language of phonology, familiarity with the basics of information theory, and a deep acquaintance with the techniques of software development and implementation. The latter is implicit in the development of the software necessary to run the experiments and will not be explored further. The interested reader can find the the code developed for this project in Appendix I.

### 1.7 The Architecture of Large Vocabulary Speech Recognitions Systems

Speech recognition systems have a four part architecture as shown in Figure 1.1 (adapted from Jurafsky and Martin 2000).



**Figure 1.1:**

**Diagram of an LVCSR System**

The following sections provide an overview of the major components.

#### 1.7.1 Feature Extraction

Feature extraction belongs to digital signal processing, a subfield of electrical engineering. Indeed, the IEEE, the primary professional organization of electrical engineer includes a special interest group called the Signal Processing Society. The IEEE Conference

on Acoustics, Speech, and Signal Processing (ICASSP) is one of the primary venues for research in automatic speech recognition. From the signal processing point of view, speech is an acoustic signal. Engineers represent it in mel frequency cepstral coefficients (MFCC), a set of 39 feature vectors, “each vector representing the information in a small time window of the signal” (Jurafsky and Martin, 2008, p. 295). The first step in finding the feature vectors is to convert the analog representation of speech, as captured by the impact of sound waves on a microphone, to a digital signal. In order to do this, the amplitude of the signal is sampled periodically. The technique is to sample the speech quickly enough to capture the contours of the input signal but to be mindful of the physical capacity of disk space. Taking 20,000 samples per second, or 20 kHz is more than adequate for speech processing. If we were concerned solely with telephone speech, we could get by with much less, since the telephone switching network filters out high frequencies. In fact, the audio files used in this study were constructed by sampling telephone input at 8 kHz.

The next step has to do with both with the presence of aperiodic sounds that are part of the input, and sampling that does not coincide precisely with the period of the waveform. The idea is to extract the data from a short segment of speech that characterizes—it is hoped—a piece of a phone. The term *phone* rather than *phoneme* is used in the ASR literature. Whereas a phoneme is an abstraction enclosing all of the allophones that comprise it, a phone is just a unit of sound, presumably one that can be made with the human vocal apparatus during normal speech. The process of extraction is called *windowing*. The output of the windowing is subjected to a fast fourier transformation (FFT) which computes the amplitudes, in effect, the amount of energy, at each frequency range composing the sampled

signal. This is known as the spectrum. The logarithm of the output of the FFT is the mel scale value.

Now we are ready to capture what are known as the *cepstral features* of the signal, referred to collectively as the *cepstrum*. The cepstrum is an attempt to model the vocal tract of the speaker. The motivation behind such a model can be found in the large literature on vocal tract normalization (Johnson, 2005). If the spectrum is the output of an FFT, the cepstrum is the FFT of the log of the spectrum. To compute this, we plot the log of each amplitude in the spectrum against the frequency. Given this plot, we use the same FFT technique to produce its spectrum. The result is the cepstrum, from which the feature extractor retains the first twelve values. To these we add the energy in the window, defined as the sum of the amplitude of the samples within the window. We now have thirteen features.

Since the speech signal is not constant within the window, we compute the delta of the twelve cepstral features, defined as the amount of change for each feature between windows. We also compute the double delta, the change in each delta between windows. In addition, we compute the delta and double delta for the energy. This gives a total of 39 features, exactly the number used in the University of Colorado's LVCSR system, SONIC (Pellom and Hacıoglu, 2002) (Jurafsky and Martin, 2000), (Jurafsky and Martin 2008), (Jelinek 1998), (Rabiner and Juang 1993).

The feature set noted in Figure 1.1 are just representations of the acoustic observations symbolized in Equation 1.1. For the purposes of this study, feature extraction is treated as a black box. It is a component of the software developed at the University of Colorado, adapted at the Next It Corporation and provided by Next It for this study.

### 1.7.2 The Acoustic Model and the Lexicon

We turn now to a quick look at the acoustic model. Though the presence of the 39 position feature vector makes  $P(O / W)$  tricky to compute, we can at least give the flavor of how it is constructed. The acoustic model guesses the likelihood of observed feature vectors given a sequence of words<sup>22</sup>. A simplified version of what actually occurs is called vector quantization, effectively the transformation of the feature vectors to countable symbols. We begin by constructing a codebook. If we have 8 bits to work with, we can produce 256 distinct codes, each representing a feature vector. Next, each of the vectors in a training set of sounds is associated with one of the codes. Now we are ready to use the system. As the feature vectors from the observed sound sequence come in, they are compared to the 256 codes. A best fit determination is made according to a distance metric, for example, the Euclidean distance over an 8-dimensional space.

Not surprisingly, information is lost in the reduction from a 39 position feature vector to an 8 position code. Current LVCSR systems use more sophisticated statistical modeling techniques. Sound probabilities are represented as subphonemic units called triphones, a unit of sound and its left and right context, arranged into sequences known as hidden Markov models (HMM)<sup>23</sup>. By assuming that each cepstral feature has a normal Gaussian distribution, the system can use the Gaussian probability density function<sup>24</sup> to guess the probability that a particular HMM could generate a single cepstral feature:

---

<sup>22</sup> It's more complicated than this, since  $W$  is represented not just as words but as subphonetic sequences. But the idea is the same.

<sup>23</sup> See Appendix B.

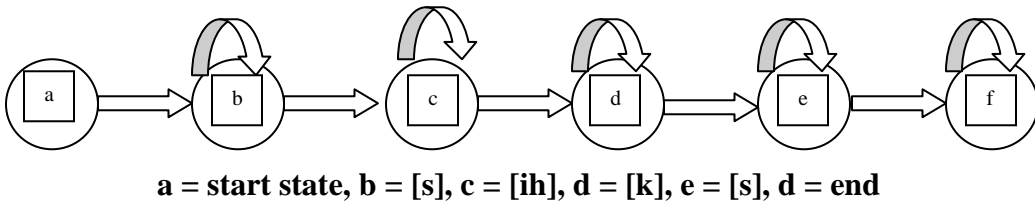
<sup>24</sup> The graph of this function is the well-known bell curve

$$f = \frac{1}{\sqrt{\sigma^2 2\pi}} \exp\left(-\frac{(x-u)^2}{2\sigma^2}\right) \quad 1.8$$

where

- $f$  is the probability that a specific HMM state could generate one value of the feature vector
- $x$  is value of the HMM state
- $u$  is the state's expected value, or mean
- $\sigma^2$  is the state's variance about the mean

This discussion has glossed over the problem of how the mean and variance for each state was computed, in effect, how we managed to train the model. To begin, we borrow a simplified HMM for the word *six* from Jurafsky and Martin for Figure 1.2. The circles represent the sounds that comprise the word in ARPAbet notation, a transcription system that uses text for the conventional IPA symbols<sup>25</sup>. The arrows represent transition probabilities. That is, given that the last sound (or phone) was [ih] there is a non-zero probability that the next sound will be [k]. The arrows pointing back to the state, represent the differing lengths of each sound depending speaker (i.e., [s ih k s], as opposed to [s ih ih ih k s]).



**Figure 1.2: HMM for the Word Six**

<sup>25</sup> Unless otherwise stated, all transcriptions are in the ARPAbet notation found in Jurafsky and Martin (2008, pp. 217-218).

If we imagine that we have many such HMMs as words, in fact, one for each sequence of sounds encountered in the training set, the collection of HMMs represent a lexicon. In the words of Jurafsky and Martin (2009, p. 294), the lexicon is “a set of pronunciations for words, each pronunciation consisting of a set of subphones, with the order of the subphones specified by the transition probabilities.”

The simplest place to start is to imagine that we have an already labeled training set. That is each acoustic observation is labeled with a distinct HMM state. Then we could compute the mean and variance over all acoustic observations for each state. The problem is more difficult than this since we really don’t know which observations corresponds to which state. It is even more complex when we consider that each of the observations consists of 39 separate features. In practice, acoustic models undergo a process of “forced alignment:” HMM states are aligned with acoustic observations using the Viterbi algorithm (see Appendix B). The result of this entire process is a collection of probabilities of acoustic observations given word sequences. See Section 5 for a discussion of the specific acoustic model used in this study.

### **1.7.3 The Language Model**

Though the language model is less difficult to describe than the acoustic, we should confess to the reason for its existence in the first place. The problem of acoustic recognition is so far from solved that the language model serves to prune unlikely word sequences from the candidates (Greenberg 2002).  $P(S)$  is just the probability that a given symbol sequence occurs in the sample. If  $S$  were a single symbol it would be computed by the number of times it occurs in the sample that comprises the language model divided by the number of

symbols in the sample. But, in fact,  $S$  is a string of symbols. This is more difficult to compute than for a single symbol, but only slightly so.

When Shakespeare says in *The Tempest*, “What’s past is prologue,” he might have been speaking an object called an *N-gram*. With an *N-gram* model, we can try to predict the  $N$ th word in a sequence based on the previous  $N - 1$  words. The intuition here is obvious. Given the sequence of words, “The University of,” “California” seems more likely to follow *of* than *in*. This observation uses no explicit linguistic knowledge. Though we might know that in English nouns follow prepositions, not other prepositions, this knowledge is not explicitly encoded in the LVCSR. It is there implicitly, however, since in a text that contains the string *of California* and does not contain *of in*, the probability of the former will exceed zero, while the probability of the latter will not. For the moment, we will drop the distinction between symbols and words made earlier in this section. The construction of an *N-gram* language model is identical whether we are thinking of words or of sequences of characters. But just as it makes no difference to a computer whether we refer to a memory location by its hexadecimal address or indirectly by an English-like variable name, while it makes all the difference in the world to software developer, the use of familiar word sequences instead of symbol sequences will make the following discussion clear.

So, the probability of a word sequence is just the probability of the words that compose the sequence:

$$P(W) = P(w_1 \dots w_n) \tag{1.9}$$

We can decompose Equation 1.9 with the probability chain rule:

$$P(w_1^n) = P(w_2|w_1)P(w_3|w_1^2) \dots P(w_n|w_1^{n-1})^{26} \quad 1.10$$

Of course, this does not look much easier to compute. To make things simpler, LVCSR systems approximate the full decomposition by looking only a specified number of words into the past. This is known as the *Markov Assumption*. A model that looks one word in the past is called a *bigram*, two words a *trigram* and N words an *N-gram*. The general form for the Markov Assumption for an arbitrary N-gram is given in Equation 1.11:

$$P(w_n|w_1^{n-1}) \approx P(w_n|w_{n-N+1}^{n-1}) \quad 1.11$$

For example, suppose we are working with trigrams and we are trying to determine the probability of the fifth word in the sequence, that is,  $N = 3$  and  $n = 5$ . So, we approximate the probability of the fifth word given the previous four ( $P(w_n|w_1^{n-1})$ ) by computing the probability of the fifth word given the previous two:

$$P(w_n|w_{n-N+1}^{n-1}) = P(w_5|w_3^4), N = 3, n = 5 \quad 1.12$$

The next assumption made is that all of the N-grams are independent. This is not true, of course. Syntactic, semantic, and discourse structure can reach across multiple words. Still, the larger the N in the N-gram, the more of this structure we capture. Given the assumption of independence, to estimate the joint probability of a sequence of N-grams, we simply multiply the probabilities of each of the N-grams. Thus, if we were computing the unigram probability of an utterance we would multiply the probabilities of each word in the utterance. If we were computing the bigram probabilities of an utterance, we would multiply the bigram probabilities of each bigram that composed the utterance and so on<sup>27</sup>.

---

<sup>26</sup> See Luger (2009); Jurafsky and Martin (2008).

<sup>27</sup> It's not quite this easy. The algorithm for computing the trigram probability requires initial and ending bigram probability and so on for larger N-grams. The idea, nevertheless, is to multiply the estimated probabilities of the N-grams that compose the utterance.



To make this more concrete, assume a bigram sequence. To compute the bigram probability<sup>28</sup> of a word A given a word B, we count the number of bigrams BA sequences and divide by all bigrams that share the same first word B. Since all appearances of word B must equal the bigram count for bigrams beginning with B, bigram probability is expressed in Equation 1.13.

$$P(w_n | w_{n-1}) = \frac{\text{Count}(w_{n-1}w_n)}{\text{Count}(w_{n-1})} \quad 1.13$$

To illustrate, we adapt a corpus found in Huang et al. (2001, p. 560). It is necessary to use the special start symbol <s> to account for the bigram probability of utterance initial words. Likewise, we end each utterance with the special end symbol (</s>) for technical reasons having to do with probability distribution.

<s> <b>Paul wrote his thesis</b> </s>
<s> <b>James wrote a different thesis</b> </s>
<s> <b>Paul wrote a thesis suggested by George</b> </s>

*Figure 1.3: Mini Corpus*

So, using Equation 1.13 and Figure 1.3, we can compute the bigram probability of “<s> Paul.”

$$P(\text{Paul} | \text{<s>}) = \text{Count}(\text{<s>}, \text{Paul}) / \text{Count}(\text{<s>}) = 2/3$$

---

<sup>28</sup> Strictly speaking, the bigram probability is not the probability of a particular bigram appearing. That would be computed by dividing the number of bigrams of a particular type by the number of bigrams. Our computation is an instance of normalization. It serves as an estimate of the true probability (Huang et al., 2001).

This technique results in the following (non-exhaustive) table of bigram probabilities.

$P(\text{wrote} \text{Paul}) = 2/2$	$P(a \text{wrote}) = 2/3$
$P(\text{thesis} a) = 1/2$	$P(</s> \text{thesis}) = 2/3$

**Figure 1.4: Bigram Probabilities for Figure 1.3**

With these we can estimate the probability of “Paul wrote a thesis” as:

$$\begin{aligned}
 P(\text{Paul wrote a thesis}) &= P(\text{Paul}|<s>)P(\text{wrote}|\text{Paul})P(a|\text{wrote})P(\text{thesis}|a)P(</s>|\text{thesis}) \\
 &= .148
 \end{aligned}$$

One implication of the N-gram model is that increasing the size of N usually results in a better model. We can illustrate this with an extended example from Jurafsky and Martin (2008) who in turn adapted the technique from Shannon (1951). Suppose we could construct a unigram model of some very large corpus of English writing, say the complete works of Shakespeare. We have a long list of words (884,647 to be exact) followed by their probabilities. Now suppose we randomly generate a number between 0 and 1 and print out the word (or words) whose probability is closest to the random number generated. We could construct a bigram model of Shakespeare using the same technique, a trigram and so forth. Jurafsky and Martin did this very thing, whose results are reported below (Jurafsky and Martin, 2008, p. 93):

#### Unigram

To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have  
 Every enter now severally so, let  
 Hill he late speaks; or! A more to leg less first you enter  
 Are where exeunt and sighs have rise excellency took of.. Sleep knave we. Near; vile like

## Bigram

What means, sir. I confess she? Then all sorts he is trim, captain.

Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.

What we, hath got so she that I rest and sent to scold and nature bankrupt, nor the first gentleman?

## Trigram

Sweet prince, Falstaff shall die. Harry of Monmouth's grave.

This shall forbid it should be branded, if renown made it empty.

Indeed the duke; and had a very good friend.

Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.

## Quadrigram

King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;

Will you not tell me who I am?

It cannot be but so.

Indeed the short and the long. Marry, 'tis a noble Lepidus

The larger the N-gram, the more coherent the sentences, where coherence is shorthand for linguistic structure. This is dramatic but not really very surprising. There are over  $10^{17}$  possible quadrigrams. After the program has chosen "It cannot be but," there are only five possible continuations in Shakespeare, one of which is "so." Given these facts, the machine generated a sentence not just Shakespeare-like, but directly out of King John (act III, scene iv).

#### 1.7.4 Representing the Language Model

Most LVCSR systems store their language models in ARPA<sup>29</sup> format which is a text file followed by all N-grams of probability greater than 0 in the language model. Figure 1.5 is a piece of the ARPA-formatted language model used in the experiments in this study. The file begins with some simple header information that tells the recognizer how many N-grams of each type. For illustration purposes, the language model generation software was run to generate all N-grams of sizes 1 through 4.

The negative numbers on the left side of each number are N-gram probabilities but expressed as logarithms. The reason for this has to do with the limitations of computers. Some N-grams are highly improbable. If we multiplied the probability of many improbable N-grams together, we would get a very small number, frequently less than the precision of the machine. The use of log probabilities instead of raw probabilities produces larger numbers. For example, suppose we had a sequence of four N-grams, each of whose probability estimates was .05. Then the probability estimate for the entire sequence is  $(.05)^4 = 6.25 \times 10^{-6}$ . The log of this value is -5.204. Logarithms have another advantage. Adding the logs of real number is equivalent to multiplying them. Addition can be done more quickly than multiplication on a computer.

The number to right of some N-grams is referred to as backoff weight. It is produced by an algorithm that tries to compensate for the certain necessary limitations in any corpus. See Huang et al. (2001, pp. 562-570) and Jurafsky and Martin (2008, pp. 97-107) for a complete discussion.

---

<sup>29</sup> Advanced Research Projects Agency.

```

\data\

ngram 1=485
ngram 2=1930
ngram 3=650
ngram 4=507
\1-grams:
-0.923700  </s>
-100.000000 <s>
-0.905346  @UNK
-1.871636  A      -0.415531
-3.477764  ABOUT -0.150353
...

\2-grams:
-0.484316  BOOK A      -0.267162
-1.375284  BOOK AIRLINE
-1.375284  BOOK RESERVATIONS
-1.375284  BOOK SOME
-1.375284  BOOK TRAVEL
...

\3-grams:
-0.258522  BOOK A FLIGHT
-0.301441
-1.111498  BOOK A TICKET
...

\4-grams:
-0.565298  BOOK A FLIGHT </s>
-1.540344  BOOK A FLIGHT FOR
-0.392099  BOOK A FLIGHT FROM

```

**Figure 1.5: ARPA Formatted Language Model**

As already noted, the current study is an investigation of alternative language models and a post-processing feature that we will call the *concept component* or *concept model*.

An operational LVCSR with a fully-trained acoustic model is the starting point for the investigation summarized in this thesis. Though the acoustic model is the fruit of complex advances in digital signal processing, and very clever training algorithms, some researchers

believe that the bulk of future advances in speech processing will be made by refining the language model.<sup>30</sup>

## 1.8 Scoring

The general task of a speech recognition is to take as input an acoustic waveform and to produce as output a string of symbols probabilistically related to the same string encoded in the acoustic waveform. The success of a system is measured in the word error rate (WER). Though we will use this metric throughout the study, since our outputs will be, depending on the experiment, words, syllables, and syllables plus another symbol sequence that we will call *concepts* and explain later, WER seems a bit misleading. Since words, syllables, and concepts, fall (by our definition) under the general category, symbols, we will use the term *symbol error rate* (SER) and allow the context to distinguish which kind of symbol we are discussing. Figure 1.6 shows the formula. Symbol error rate is the 100 times the number of insertions, substitutions, and deletions divided by the number of tokens in correct transcript of the utterance.

$$SER = (100 * (I + S + D)) / T$$

where:

- $I$  is the number of insertions
- $S$  is the number of substitutions
- $D$  is the number of deletions
- $T$  is the total number of symbols in the correct transcript of the utterance

**Figure 1.6: Symbol Error Rate**

---

<sup>30</sup> Charles Wooters, personal communication.

Scoring throughout this study, indeed throughout all comparative studies of speech recognition, is done with software available from the National Institute of Standards and Technology known as *sclite* (Sclite, 2009). Sclite makes use of a dynamic programming algorithm that measures how closely two strings are to one another. The specific algorithm used is called *minimum edit distance* and is explained in detail in Appendix B.

In the typical case, the audio input files are hand-transcribed to symbols. These are referred to as *reference files*. The recognizer generates hypothetical strings, based on audio input, that correspond the reference files. These are referred to collectively as *hypothesis files*. Sclite attempts to align these files (see Appendix B) and compute the cost of transforming one into the other, effectively constructing a measure for how significantly the files differ.

Consider the example in Figure 1.7.

Reference	i		r	a	n		*	*		t	O		t	h	e		D	o	o	R	
Hypothesis	i		g	o	t		i	t		*	*		t	h	e		P	h	o	N	e
Operation			S				I			D							S				

**Figure 1.7: Alignment of Two Strings**

The hypothetical utterance required four operations, two substitutions, an insertion, and a deletion. Since there are five symbols (words in this case) in the reference file, the symbol error rate is given by:

$$SER = 100 \left( \frac{2 + 1 + 1}{5} \right) = 80\%$$

Speech recognizers do quite well with single digits, with a SER in the neighborhood of .5%. They also do well with read speech in 5 – 20,000 word corpora. Error rates here are

about 3%. It is with large vocabulary continuous speech that things get hard. LVCSR word error rates range from 20% to over 40% on telephone speech, depending on whom you read.

## 1.9 The Syllable/Concept Hypothesis

To summarize, five major components of speech recognizer are the feature extractor, the acoustic model, the lexicon, the language model, and the decoder. The extractor samples the acoustic waveform at specific intervals and transforms it into a collection of spectral features. The acoustic model is a collection of probabilities of acoustic observations given a word sequence. The lexicon is an HMM sequence of sounds in each word. The language model is a collection of word sequence probabilities. The decoder uses the acoustic and language models to transform the spectral features into text. The acoustic model is trained on sequences of sounds, or phones.<sup>31</sup> The language model is trained on sequences of words (Rabiner & Juang, 1993; Jelinek, 1998; Jurafsky & Martin, 2009). As noted this study is entirely concerned with the language model and with a post-processing component called the *concept model* or *component*. A fully functioning, fully-trained acoustic model is assumed.

The important point to notice is that the two organizing elements in a speech processing system are phones and words. An alternative hypothesis suggests itself. Would a speech recognition system perform more accurately if it were trained on syllables instead of on phones and words?

Notice that there are two questions here, the question of syllables in the acoustic model and the question of syllables in the language model. The use of phones in the acoustic model is inadequate for reasons that will be familiar to any phonetician. A study of the

---

<sup>31</sup> A triphone actually, which is a phone with its left and right context. This will be explained more fully in Section 2.

Until then, and speaking loosely, the word *phone* will serve.



SWITCHBOARD corpus found that over 20% of the manually annotated phones are never realized acoustically. Speakers tend to delete phones<sup>32</sup> that would be expected in the canonical pronunciation, especially when speaking rapidly. Even taking this into account, words are pronounced in many different ways depending on dialect, idiolect, speaking rate, acoustic environment, and pragmatic/semantic context. On the other hand, studies using SWITCHBOARD have shown that 99% of canonical syllables are realized in speech (Greenberg et al., 1999). Further, syllables encode prosodic information that might contribute to the speech recognition (Godfrey et al., 1992; Greenberg et al., 2002).

Syllables in the language model have been less widely studied. The reason, as suggested above, is clear. The language model contains statistical arrangements of word sequences. It guides the decoder in making a best guess as to what *word* sequence was uttered. Word identification might be less critical, however, where the LVCSR is not intended to transform an acoustic signal to human-useable output. One example is tracking children's reading error rates. Here the task is not to output a word, but to assess whether the utterance was the expected utterance (Bolanos, et al., 2007). Experiments associated with syllables in the language model are the subject of Sections 3 and 4.

Now let's go downstream a bit. In certain very limited domains words or phrases can be treated as equivalent. For example, consider a call center equipped to handle airline reservations. *I want, I would like, I'd like, I need* could all be treated as meaning the same thing. Meaning? For our purposes, the suggestion that meaning expresses intentionality on

---

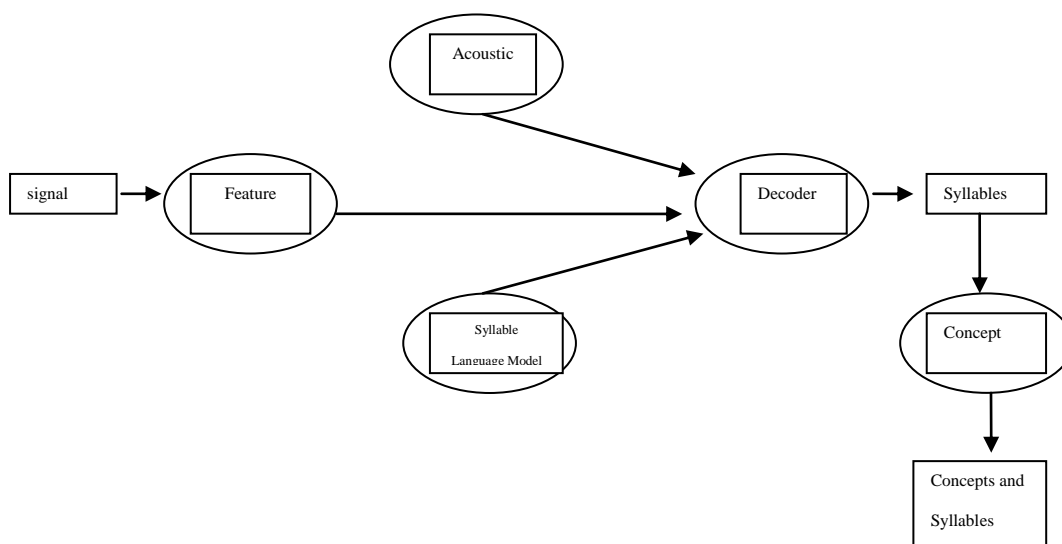
<sup>32</sup> As with all things, matters are slightly more complicated. Browman and Goldstein (1990) found that many seemingly non-existent segments are, in fact, articulated but not expressed acoustically: "We propose that most of the phonetic units (gestures) that characterize a word in careful pronunciation will turn out to be observable in connected speech, although they may be altered in magnitude and in their temporal relation to other gestures" (p. 360).

the part of the speaker seems close enough. Whether a speaker says, *I want to fly from Spokane to Seattle on October 10*, or *I would like to fly from Spokane to Seattle on October 10*, it seems clear enough that the speaker is announcing a very specific intent that requires a very specific response—a list of available flights. The collection of words and phrases “with the same meaning” are referred to as *concepts* by at least one other research group (Hacioglu and Ward, 2002, p. I-226; Pellom et al., 2001).<sup>33</sup> If an LVCSR reduces *I want*, etc. to a single concept, WANT, it may produce better results. Proof-of-concept experiments associated with this kind of reduction are described in Section 5. Finally, Section 6 conjectures that, given the positive results reported in Sections 3, 4, 5, we might produce even better results if we built a stochastic model that would probabilistically map concepts to words and word sequences. This probabilistic concept component would serve as a post-processor to a conventional speech recognition system. Figure 1.8 shows the complete system.

As of this writing, the experimental data gathered for the use of syllables in the language model has been quite robust. We have no reason to believe that these results will not scale up to larger corpora. We consider the current version of the post-processor as described in Section 5 to be a proof-of concept. In Section 5, we argue that the current configuration produces an upper bound on the error rate. We anticipate that the research outlined in Section 6 and to be performed in the future will reduce the error rate substantially.

---

<sup>33</sup> It is easy to see that these are formal equivalence classes, a term I prefer, given the long history of the term “concept” in cognitive science (Rosch, 1978; De Palma and Weiner, 1992; Weiner and De Palma, 1993). Nevertheless, to be consistent with a major research group, in fact, the developers of the recognizer used in this study, we follow Pellom et al. in this study. See Section 5 for further discussion.



**Figure 1.8: A Recognizer with a Syllable Language Model and a Concept Post-Processor**

### 1.10 Adding an Expert System

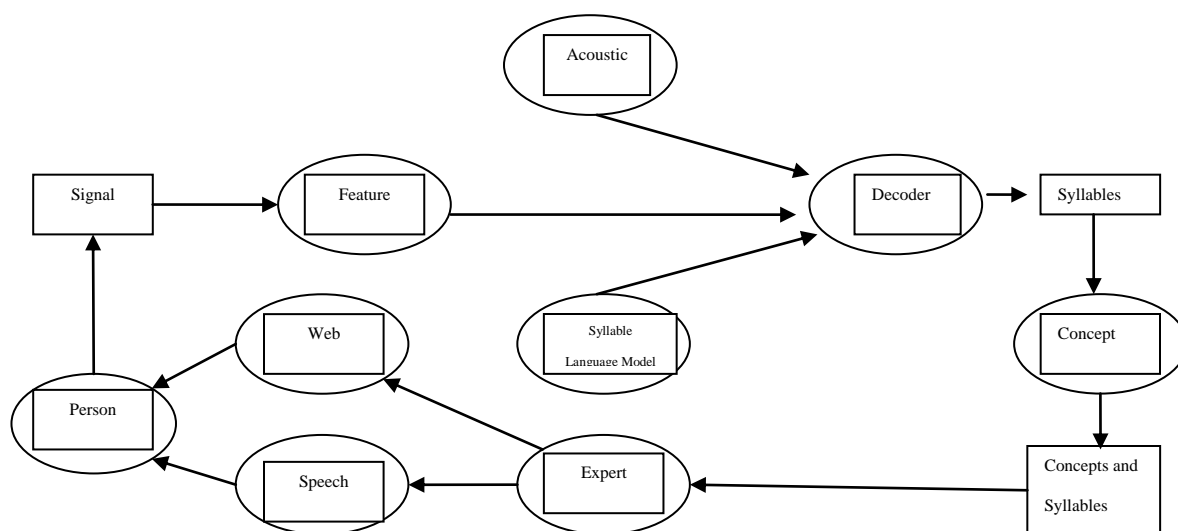
The obvious question at this point is, “What happens to the concepts and syllables that the system outputs.” These could be passed on to an expert system for further processing. The addition of an expert system, a speech synthesizer, and a web/phone display application closes the loop, as indicated in Figure 1.9. The term, “expert system,” gained currency among researchers in artificial intelligence in the 1980s. Frustrated with the slow pace of progress in general purpose problem solvers such as those pioneered by Newell and Simon in the 1960s and 1970s, researchers began trying to model explicitly human-like expertise in specific problem domains. The first task in such a system is often referred to as “knowledge engineering,” whereby the system’s designers encode knowledge, often in the form of rules, by interviewing known experts. Though the systems do not scale up to more complex problems or alternative problem domains—a system that performs medical diagnosis cannot easily be made to perform chemical analysis, just as medical doctors do not

become chemists without further training—they have performed remarkably well (Luger, 2009; Newell, 1990; Waterman, 1986).

For example, as of this writing (3/3/2009), Alaska Airlines provides an expert system travel agent. The web interface to the travel agent introduces the virtual agent (“Hi there. My name is Jenn”) and invites the user to enter typed questions. The same question was entered in two different forms:

- I’d like to go to San Francisco March 23, returning April 1. (1)
- I’d like to fly to San Francisco and come back April 1. (2)

The system responded with identical lists of to/from flights on the correct dates along with the output, “Here are the available flights based on the information you provided” (Alaska, 2010). We discuss reasons for treating utterances (1) and (2) as identical in Section 6. The addition of an expert system is proposed as further research in Section 6.



**Figure 1.9: Adding Expertise, Speech Synthesis, and a Web Display**

### 1.11 The Shape of Things to Come

In this section we have introduced the problem of automatic speech recognition.

While doing so, we have:

- Described where ASR is used
- Argued that large vocabulary continuous speech recognition is different from other types of speech recognition
- Provided a short history of ASR
- Argued that ASR is a mature discipline that meets Thomas Kuhn's specifications for "normal" scientific work
- Shown that ASR is related to cryptanalysis
- Introduced the role of Bayesian inference in ASR
- Described the components of a speech recognizer
- Explained the metric used to judge the goodness of speech recognition
- Argued for the syllable/concept hypothesis that underpins this study
- Shown how the system represented in this study could be augmented with an expert system to close the loop between human input and machine output.

In the following sections we will outline previous work on the use of syllables and concepts in ASR (Section 2), followed by descriptions of three experiments with variations (in Sections 3, 4, and 5). Section 3 considers an *a priori* estimate of the goodness of the language model. Section 4 describes several experiments designed to show that the use of syllables in the language model will result in robust speech recognition. Section 5 considers the worst-case scenario for the use of a post-processing concept model. Finally, Section 6

outlines future research, considering the development of a probabilistic concept model in some detail.

## 2.0 Previous Work

The syllable has been greatly under appreciated among speech recognition researchers. Children, at least of my generation, learned the one sure-bet syllabification rule by the seventh grade: when a word decomposes into *VCCV*, syllabify after the first consonant. Could it be that its very familiarity has led to its relative obscurity? Whatever the case, we cannot really discuss syllabification, the process that underlies the experiments in Sections 3, 4, and 5, until we have an idea about syllable structure. Syllable structure leads naturally into some arguments for its use in ASR followed by a survey of previous work on the use of the syllable in both the acoustic and language models. We end this section with a short account of work that bears familial relationship to the idea of a concept as described in Section 1.

### 2.1 Syllables

Much of the work on the use of the syllables rather than phones and words in LVCSR systems simply assumes that the syllable is a unit of language structure. So, Ganapathiraju et al. say that “A syllable ... seems to be an intuitive unit for representation of speech sounds” (1997, p.1). Sethy and Parthasarthy say more than most when they write that “the syllable is a basic unit of speech consisting of two or more phones, including a nucleus that is usually a vowel, and is generally perceived as having no interrupting pause within it” (1998, p.30). Steven Greenberg comes closest to a full appreciation of the complexity of the syllable when he calls it a “linguistic ‘wolf’ in phonetic clothing” (Greenberg, 1999, p. 162). He singles out for particular disapproval Rabiner and Juang (1993), one of the classic speech recognition texts. The book has a total of two index entries for *syllable*, and seems to treat it as just one

of several possible, and easily defined, sub-word units. What Greenberg does not mention is that Rabiner and Juang implicitly tell us why the syllable might be useful in the language model of a speech recognizer, namely that there are only about 1000 of them in English (Rabiner & Juang, 1993, p. 436).

We should point out that there is significant disagreement on this score. Stewart et al. (1998) put the count at 10,000. Ganapathiraju et al. (1997b) concur that the number is 10,000. Wang et al. (2001, p. 430) get at two problems when they say that “the large number of syllables (over 30,000) in English presents a challenge in terms of trainability.” The first is that a difference of a factor of 30 between the low and high estimates, points to the difficulty that LVCSR practitioners have in pinning down the syllable in the first place. The other is that if the actual number of syllables in English is anything close to Wang’s estimate, the problem of constructing a trisyllabic<sup>34</sup> acoustic model is probably close to intractable, since it would require  $(3 \times 10^4)^3 \sim 30$  trillion distinct syllable models. This is the combinatorial explosion so frequently warned about in texts on artificial intelligence (Luger, 2005).

An intriguing possibility, given the fuzziness of the notion of a syllable is to use the demisyllable. A syllable consists of an initial demisyllable (onset plus nucleus) followed by the final demisyllable (nucleus plus coda). Here is one compelling reason for investigating their use in LVCSR: “The English language requires about 1400 such units” (Bhaskararao, et al., 1991, p. 517). Nevertheless, investigation is constrained by the tools at hand. We are

---

<sup>34</sup> By analogy to a triphone, the structure used in LVCSR systems. A triphone is a phone and its left and right context.



not aware of any readily available software that demisyllabifies a large text-transcribed corpus. As will be clear in Section 4, a syllabifier is necessary for the proposed experiments.

Even putting aside the fact that the number of syllables is probably significantly greater than 1000, we can agree with Greenberg and other linguists that the complexity of syllabification is underappreciated by engineers. Elizabeth Selkirk (1999) devotes the first few pages of an essay on syllables arguing for the familiar hierarchical structure that divides syllables into onset and rhyme, each of which is further subdivided and each of which obeys phonotactic constraints. In fact, according to Selkirk, the argument for a hierarchical conception of the syllable goes back to at least the late forties. The next few paragraphs argue for both the usefulness of the syllable as a linguistic structure and its complexity in English. This seems a reasonable outlay of time, since engineers building LVCSR systems consistently underrate the complexity of the syllable and some linguists even deny its existence (Kohler, 1966).

Many of the terms encountered in linguistic theory are based on terms used from the Middle Ages onward—and themselves based on classical sources—to teach Latin. Nouns, verbs, subjects, words, sentences, and syllables are among them (Tomasello, 2003; Kenstowicz, 1999). This gives these terms an air of artificiality when applied to non-Western languages. The syllable has been particularly hard to pin down as an element of phonological structure, in part, because it is not a sound, but rather an abstraction over particular organizations of sound. Nevertheless, as Kenstowicz (1999, p. 250) has argued, the syllable has come to be recognized as “an essential unit of phonological organization,” at least in the generative tradition. The question that naturally arises is why should an empirical study with the ultimate goal of building an artifact care about the generative tradition. The

answer is an easy one. The syllabifier used in this study is based on Kahn's 1976 dissertation from MIT, the very soul of generative grammar (NIST, 2007; Kahn, 1976).

Generative researchers have offered several arguments for the existence of syllables, all of them in one way or another driven by a desire to derive parsimonious phonological rules. Consider the sound sequences [t r] and [t l], the first is found word initially in English while the last is not.<sup>35</sup> From the generative point of view, we have a rule. But we can be more precise. What happens word medially, as in *Atlantic* and *atrocious*? Notice that the /t/ in *atrocious* is aspirated, while the /t/ in *Atlantic* is glottalized. If we admit the existence of syllables and specify that [t r] may be a syllable onset while [t l] may not, we produce syllable breaks before /t/ in *atrocious* and after /t/ in *Atlantic*. This allows us to account for the aspirated and glottalized /t/ allophones: syllable-initial /t/ is aspirated and syllable-final /t/ is glottalized. Said another way, since the presence of [t r] word initially and the absence of [t l] in that position only constrains the shape of words, we are entitled to ask why the pattern repeats itself word-medially, and with allophonic consequences. The implication is that the syllable lets us express phonotactic rules that might go otherwise unnoticed.

Once we agree on the usefulness of the syllable, the next step is to appreciate the complexity, but also the regularity, of its structure. The traditional approach, as noted above, has been to characterize a syllable as structure composed of an optional consonant onset followed by a rhyme that is further divided into a nucleus followed by an optional consonant coda. Figure 2.1 gives the basic form. In general, sonority rises from onset to nucleus and falls toward the coda. Phonologists appear to agree that speech sounds form the following

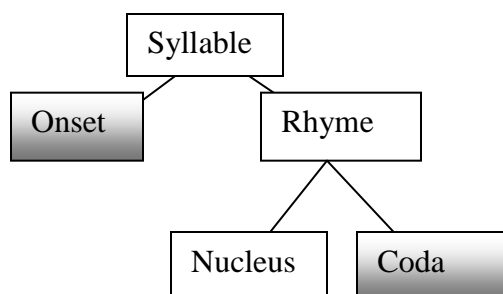
---

<sup>35</sup> The following discussion of syllable structure, along with examples, comes from Kenstowicz (1999).

sonority hierarchy, from highest to lowest: vowels, glides, liquids, nasals, obstruents

(Kenstowicz 1999).

The hierarchy is reflected in the basic form, with a high sonority vowel in the nucleus position preceded by a lower sonority consonantal onset. All languages appear to have at least {V, CV} in their syllabic inventory. Most supplement it with codas, producing {V, CV, VC, CVC}.



**Figure 2.1: General Syllable Structure**

Languages, such as English, with complex syllable inventories supplement them in ways that are usually consistent with the sonority hierarchy. Thus English permits double consonant onsets, for example, *twist*, *priest*, *fly*, and *gripe*. If we take these monosyllabic words as having a nucleus at the vowel, in every case, the double consonant onset, has a consonant lower in the sonority hierarchy preceding one above it in the hierarchy. Thus the obstruent [t] precedes the glide [w] in *twist*. So sonority rises to vowel and then falls to the fricative [s], an obstruent, and falls further to another obstruent [t], but lower in the sonority hierarchy than [s].

In the English the sonority hierarchy is useful but not absolute. For instance, the obstruent [p] does not cluster with the glide [w], though the position in the sonority hierarchy

would permit it. One possibility is that English avoids grouping sounds that use the same articulators, the lips in this instance. Even this prohibition is not absolute.

Consider [s] which violates the presumed restriction on contiguous consonants using the same point of articulation (e.g., *slip*), violates the sonority hierarchy (e.g., *spit*), and even participates in three consonant onsets (e.g., *split*). We have not yet mentioned words like *depth*, where the plosive [p] precedes an affricate in the coda, a sound of higher sonority. These and other features would have to be considered to give a full account of the English syllable structure. One has to wonder if the initial teasing regularity of the English syllable followed by the many exceptions is not an argument for treating syllable structure, like speech recognition itself, probabilistically. In fact, as we will see in the next section, probabilistic syllabifiers do that very thing. In any case, the picture should be in focus: syllable structure in English is far more complex than what one may have learned in grammar school.

### 2.1.1 Syllabifiers

The next question is, given an English word, how do we produce a syllabification? This question is of more than theoretical interest, since the hypothesis being entertained here requires training the language model on a syllabified corpus. Computational linguists for the past decade have been producing syllabifications using software available from The National Institute of Standards and Technology, a federal agency that is the repository of scientific and engineering measurement standards in the United States (NIST, 2007). The NIST syllabifier implements Daniel Kahn's widely cited dissertation (Kahn, 1976).

The conundrum is that any syllabification software implements a theory of syllables. Since Kahn worked at MIT in the seventies, there is no surprise to learn that he provides a

collection of English-centric rules justified by elicited data. His rules are stated clearly enough to have been implemented in at least one other syllabifier (Stewart, 1998).

Nevertheless, the question to be faced is the rather difficult one of deciding how accurate his rules are, difficult because we have to specify what we mean by *accurate*. In the natural sciences, we gauge the accuracy of a model by asking how closely it allows us to make predictions about the phenomenon under investigation. So, early in the last century Robert Millikan and Harvey Fletcher measured the charge on an electron by balancing a falling droplet of oil between two charged metal plates. The claim made was not that the results were true for all electrons for all time, but that, within a certain margin of error, the results would be both reproducible and consistent with observations obtained elsewhere. In short, the results were not deductive but, rather, probabilistic<sup>36</sup>.

Taking a cue from physics, we might measure the accuracy of Kahn's rules against a randomly chosen, sufficiently large, group of human subjects across a random selection of language groups. The problem is that while people appear to agree on the nucleus of syllables, they are not at all in agreement about their boundaries (Goslin & Frauenfelder, 2000). This lack of agreement has prompted some investigators to question the very existence of the syllable as a universal. As Kohler (1966) argues, the syllable is either unnecessary (because the division of the word is already known by other means), impossible (because the division is arbitrary), or harmful (because the division "clashes with grammatical formatives" (p. 207)). But to suggest that syllabification is problematic

---

<sup>36</sup> In fact, this is science idealized. As a physicist who examined his notebooks says, "Millikan had a good idea of the results he expected, although his expectations seem to have changed as he went along" (Franklin, 1981, p. 192). John Waller's readable book (Waller, 2004), where I learned about Franklin, argues that science is a human endeavor, with all its foibles, including the willingness to bend observations to conform to one's wishes.

because human syllabifiers disagree is like saying that Millikan's experimental results are meaningless, because we can find a few hundred (or thousand, or million) electrons whose charges vary from his results.

The disagreement among informants with respect to a particular syllabification is analogous to the variance of the charge of individual electrons around a mean discovered through experimental observation. Put another way, just as physicists gather data about physical systems, linguists gather data, not about linguistic systems in the abstract, but about language used by those subjects who participate in linguistic experiments. It seems uncontroversial (at least in certain circles), to assert that there is no language apart from its use. The best thing we can ask of a particular model is how closely it conforms to experimental data.

In fact, there are other syllabifiers, possibly more congenial to usage-based linguists (Hall, 2006). These try to guess new syllabifications on the basis of a corpus of words that have already been syllabified. Where the original syllabifications come from, of course, is a question that must be addressed. One study of three syllabifiers that use this technique indicates that they outperform the Kahn-derived NIST syllabifier (Marchand, Adsett, Damper 2007).

Still, we propose using the NIST syllabifier in this study simply because it is available, it remains the standard in computational linguistics, and produces results useful in an engineering context. Whether we call the output of that piece of software a *syllable* or a *segment* is immaterial. Of course, this raises the question of the theoretical status of this study. To which the answer is that we make no theoretical claims about the status of the syllable cross-linguistically (or for that matter, even in English). If the experiments

described in Sections 3, 4, and 5 propose produce positive results, then we can conclude that Kahn's work captures something about English. Whether that something is a complete description of English syllabification is another issue. If the performance of a speech recognizer enhanced with a syllable language model performs better than one with a traditional word-based model, a possible next question is whether or how it would perform if it were equipped with a better syllabifier. This is an empirical and, at bottom, an engineering question.

### 2.1.2 Stress and the NIST Syllabifier

The NIST syllabifier operates both with and without inserted stress markings. So, for example, given the string:

(1) beater after

NIST will produce output whether it is given

(2) b '1 iy t '0 ax r # '1 ae f t '0 ax r

or

(3) b iy t ax r # ae f t ax r

where "1" and "0" indicate stress and no stress respectively. This is interesting in itself, since two of Kahn's five rules mention stress, both in cases of ambisyllabic consonants. The probable NIST approach is to put a syllable break between two intervocalic consonants if stress is not marked. As we will see in Experiment 2.3 (Section 4.6), the inclusion of stress markings, given the way syllabifications are chosen for further processing in this model, has no effect. Nevertheless, as we point out in item 6 of Section 6.0, the inclusion of stress markings may yield useful results under a different protocol than the one used in this study.

For that reason, and because stress is central to Kahn’s model, I provide a discussion of experimental results linking syllables to stress in the next subsection.

### 2.1.3 Syllables and Stress

There is reason to believe that stress is involved in human syllabification, though no one appears to have compared stress-marked and non-stress-marked input to the NIST syllabifier with human subjects. Cutler and Norris (1988) argue, based on experimental data, that human lexical access—in English—is augmented by segmenting at strong syllables in continuous speech. English is a stress language, stress being “its most noticeable structural characteristic” (Cutler & Norris, 1988, p. 114).<sup>37</sup> By this they mean, among other things, that strong syllables are characterized by full vowels, whereas weak syllables contain schwa or some reduced form of a vowel. *Eye* and *pill* are strong syllables. The second syllables in *ion* and *scrounges* are weak. Initial strong syllables, they observe, are three times as common in lexical (non-function) words as initial weak syllables. Further, those words beginning with strong syllables occur twice as frequently as weak syllables, implying that English speakers use six times as many words that begin with strong syllables than those that begin with weak syllables. These observations lead to a testable hypothesis: “segmentation for lexical access occurs at strong syllables” (Cutler and Norris 1988, p. 115). The article presents experimental evidence that supports the hypothesis.

This points to a problem in most current speech recognition systems. Up till now, we have been assuming that the acoustic models of most systems are sequences of phones. More

---

<sup>37</sup> Later, Cutler, Dahan, & van Donselaar (1997) review research on how prosody affects comprehension at the word, syntax, and discourse levels. They continue to argue that prosody contributes to word access “to the degree that its use is efficient.” At the same time they observe that while most models of lexical access are “computationally explicit, we know of no attempt to implement modeling of the prosodic structure of the input” (p. 185). Section 4 examines the use of stress-marked vs. non-stress-marked input to the NIST syllabifier.



precisely, they are frequently a phone with its left and right context, known as a *triphone*. Consider [ae] in *his cat in*. Three triphones capture the context: [z k ae] [k ae t] [ae t ih]. The use of triphones goes a long way toward capturing coarticulation issues and is the state-of-the-art in LVCSR. Yet while they capture context, they seem to undervalue stress. Huang et al. (2002) mention that LVCSR systems use word-level stress in creating phones. This appears to mean that to the extent that different stress patterns result in different acoustic realizations, stress is captured. One question is whether the stress realized acoustically at the triphonic level might not be more accurately retrieved at the syllabic level. If, in fact, speech segmentation, as Cutter and Norris argue, is triggered by strong syllables, then LVCSR systems are ignoring available information as to the identity of a lexical item. Speech recognition is, in essence, a classification process, first of phones, then of words. During phonetic classification, the presence of a full vowel could signal the system to attempt a lexical access with that vowel plus the longest onset permitted by the current sequence of phones. In theory, this would perform better than attempting an access at every phone.

Were LVCSR systems to make greater use of stress, the first hurdle would be how to identify it. It is well-understood that duration and amplitude, along with  $F_0$ , play a role in stress determination. For instance, stressed nuclei are usually considerably longer than unstressed nuclei. One study found that that an algorithm that used the product of duration and amplitude to mark stress yielded results that approximated those of hand transcribers (Greenberg et al., 1999; Greenberg, 2001; Greenberg, 2002). Since stress is involved in syllabification, the use of syllables in the acoustic model—the subject of the next section—implicitly employs stress in speech recognition. We return to stress in Section 4.6.

## 2.2 Syllables in the Acoustic Model

The experiments developed for this study and presented in Sections 3, 4, and 5 are concerned with the language model and with a post-processor that we call the *concept model* or *component*. Nevertheless, much of the prior investigation of syllables in ASR has been of their use in the acoustic model. The present study would be incomplete without surveying the arguments for their use in the acoustic model and reporting results from the literature.

McAllaster et al. (cited in Greenberg, 1999) constructed a simple study that showed what is intuitively clear, namely the closer the match between what the speaker says and what is stored in the acoustic model, the more accurate the recognition. Acoustic models contain two mismatches with respect to speaker utterances. First, they contain several pronunciations for the same word, many of which may not be encountered in a given session. Second, the system may encounter pronunciations that are not part of the acoustic model. To cope with both problems, McAllaster matched the Switchboard Corpus precisely with the entries in a speech recognition system. He then input the Switchboard Corpus to the now-aligned system. The word error rate was reduced from 40% to less than 5%.

While attempting to ascertain the source of the mismatch, Greenberg, et al. (1999) did a study that is widely cited in the literature. His team phonetically transcribed four hours of the Switchboard Corpus. They discovered, for example, that the word *and* has eighty pronunciation variants in the sample, that *I* has 53, and that the word *that* has a full 117 variants out of 328 tokens.

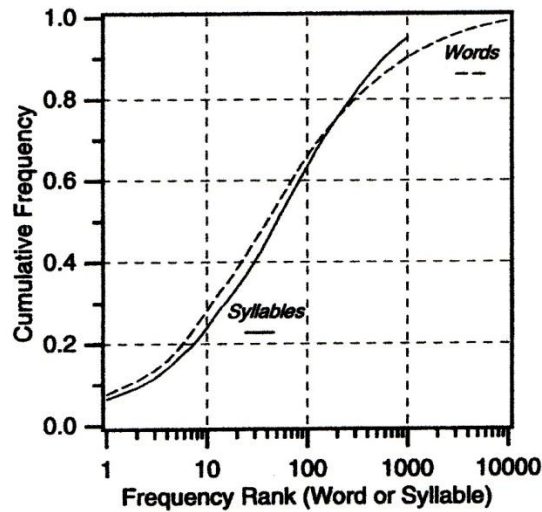
Frequency is a significant factor in determining the extent of pronunciation variation, with the ten most common words accounting for a quarter of all lexical occurrences, and the 100 most common accounting for almost two-thirds of tokens in the Switchboard Corpus.

Most of these, of course, are function words. According to Greenberg, many that are not function words “stem from just a few basic nominal, adjectival or verb forms. Clearly, mastery of these 100 most common words goes a long way towards facilitating comprehension of spoken discourse” (Greenberg, 1999, p. 166). Monosyllabic words account for only 22% of the corpus by type. But when the number of tokens are counted—the number of times a monosyllabic word is used—the fraction jumps to four fifths. See Table 2.1. Greenberg points out that a telephone dialog corpus from the 1920s shows a nearly identical relationship to that displayed in Table 2.1. between frequency and syllable length.

**Table 2.1: *Frequency distribution of words by syllable length.***  
***All*: word tokens, *STP*: tokens transcribed by the Switchboard**  
**Transcription Project, *Lexicon* to word types** (from Greenberg, 1999, p. 167).

# Syllables	Usage (%) (All)	Usage (%) (STP)	Lexicon (%)
1	81.04	78.42	22.39
2	14.30	16.31	39.76
3	3.50	3.72	24.26
4	0.96	0.95	9.91
5	0.18	0.23	3.21
6	0.02	0.03	0.40

Here is where syllables reenter the picture. The statistical distribution of the 300 words that occur most frequently is almost identical to the distribution of the most common syllables. That is, if you plot curves of the cumulative frequency as a function frequency rank for both words and syllables, the curves are almost identical, as shown in Figure 2.2.



**Figure 2.2:** “The cumulative frequency of syllables in the entire Switchboard Corpus as a function of syllable frequency rank compared with the cumulative Frequency of occurrence for words in the same corpus” (Greenberg, 1999, p. 167).

What can be inferred about the syllables used in polysyllabic vs. monosyllabic words from the data presented? One would have to remain agnostic about the syllables that form polysyllabic words in English, but might be inclined to infer that the most frequently occurring syllables are constituents of the most frequently occurring words. Only 4.66% of word tokens ( $100 - (81.04 + 14.30)$ ) have more than two syllables. Since polysyllabic words occur very infrequently, the syllables that comprise them contribute very little to the total syllable count. Let  $N$  represent the total words in the corpus,  $n_i$  represent the number of syllables in the  $i^{th}$  row in Table 2.1 ( $1 \leq i \leq 6$ ), and  $f_i$  represent the fraction of word tokens that belong to the set of words of syllable count  $i$  ( $.8104 \leq f_i \leq .02$ ), then we can easily determine the number of syllables in the corpus for shorter and longer words by taking the product of  $N$  and the appropriate  $n_i$  and  $f_i$ . Doing this, we find that 87.7% of the syllables in the corpus are found in words of two syllables or less. Said another way, if we could accurately recognize syllables, we could accurately recognize most of the corpus.

Again, according to Greenberg, “This identity between word and syllable is a natural consequence of the statistical relation between the two units and thus imply that segmentation and recognition of such entities reduce to the same thing *for this most favored part of the lexicon*” (Greenberg, 1999, p. 167, emphasis mine). You might object that “this most favored part of the lexicon,” is “favored” precisely because it is dominated by closed-class words that are low in semantic content. And you would be right. Given the way symbol error rate is computed, the failure to recognize an open-class word is no more serious than the failure to recognize a function word (see Section 1.07). On the other hand, increased accuracy in the recognition of function words and the important open-class words that are also monosyllabic, though perhaps not an end in itself, is better than not recognizing them accurately

Further, the previous description of pronunciation variability is not cause for despair. As might be expected, the most common words vary in pronunciation most frequently. But infrequently occurring words tend to vary least from the canonical pronunciation. What is more, these infrequently occurring words, usually referents to noun phrases, are also high in semantic content (Greenberg, 1999).

This is good news for ASR. The embedded training procedure for an LVCSR begins with a recorded acoustic signal, a transcription, and a pronunciation lexicon. The pronunciation lexicon is a list of words found in the transcript and their pronunciations. The important point for our purposes is that these pronunciations are considered canonical, having been gotten from some easily available source, like the Carnegie-Mellon pronunciation dictionary (Jurafsky and Martin, 2008). Hamalainen et al. (2007) report that the recognition of monosyllabic words is five times that of polysyllabic words. This seems to

follow from the lack of pronunciation variability among polysyllabic, presumably high semantic content, words.

Given the importance of syllables, Greenberg's observations about the difference between the canonical pronunciation of syllables and the pronunciation of syllables in conversational speech are worth noting:

- The onsets of syllables closely match canonical pronunciations
- Codas are often deleted
- Deviations from canonical pronunciation in the nucleus tend to be substitutions.

Greenberg et al. conclude that “the data described in this paper are incommensurate with traditional segmental models of spoken language. ... The variation observed suggests that the *syllable*, rather than the phoneme is the basic organizational unit of spoken language at the sub-word level. Moreover, prosody ... appears to play a major role with respect to the phonetic specification of pronunciation; accent's influence is differentially distributed across the syllable, both in terms of its magnitude and mode of realization” (Greenberg et al, 2002, pp. 42-43).

Though there is a good bit of information in the literature about why syllables are a more attractive unit for speech processing than the phoneme, there appear to be only four published studies of the actual use of syllables in the acoustic models of complete speech recognition systems: Ganapathiraju et al (1997a, 1997b), Sethy and Narayanan (2003), and Hamalainen et al. (2007). One reason is clear, as Ganapathiraju et al. (1997a) point out. There are too many syllables to encode as bi or trisyllables. Suppose any syllable could be paired with any other. Then, if there are 10,000 syllables, there are 100 million bisyllables. This would require a lattice too large to search in real time.

So, for reasons of computational efficiency, Ganapathiraju et al. used only those syllables that occurred at least 20 times in their training corpus. This reduced the number of syllables in Switchboard from 9023 to 2419. This meant, of course, that not all words in the corpus would be covered by the reduced syllabary. These were covered by a supplemental phone lattice. In practice, “800 syllables covered almost 90% of the training data. The remaining 10% were replaced by the underlying phone representation” (Ganapathiraju 1997b, unpaginated). This resulted in three categories of representation: words with an all syllable representation, words with a mixed syllable and phone representation, and words represented entirely by phones. When a corpus of monosyllabic words represented as monosyllables was tested against the same corpus represented as triphones, the syllabic representation achieved a 2.4% reduction in word error rate (WER) (Ganapathiraju 1997a), (Ganapathiraju 1997b).

Sethy and Narayanan interpret Greenberg et al.’s (1999) work to mean that the syllable is relatively insulated “from pronunciation variations arising from addition and deletions of phonemes as well as *coarticulation*” [emphasis mine]. (Sethy and Narayanan 2003, unpaginated). Then why is 2.4% reduction in word error rate so small? One reason is that the triphone model, a phone with its left and right context, appears to capture a good deal about coarticulation, whereas a syllable bigram captures either the preceding or the following context, but not both.<sup>38</sup> Nevertheless, Sethy and Narayanan report a 15% WER for a syllable-trained context-free recognizer compared to a 26% WER for a triphone system. Unfortunately, this 73% improvement is on a limited, 4000 word speech sample (Sethy and Narayanan, 2003).

---

<sup>38</sup> I owe these observations to Charles Wooters and Caroline Smith of my Ph.D. committee.

Hamalainen et al. (2007) attempted to replicate Sethy and Narayanan's results, but using The Spoken Dutch Corpus<sup>39</sup> speech corpus as well as TIMIT. They compared a mixed-triphone/syllable model against two triphone models, one initialized with canonical transcriptions, one initialized with manual transcriptions. They note that in TIMIT 63% of the words are monosyllabic, and almost 86% consist of one or two syllables. This distribution proves problematic. As noted above, Hamalainen et al. report that the probability of error on monosyllabic words is five times that of multisyllabic words. This appears to be a result of frequency: "a large proportion of monosyllabic words are function words that tend to be unstressed and (heavily) reduced" (Hamalainen et al. 2007, p.6). As a result, it seems that they behave as clitics with respect to adjacent words. So, to construct an example, *have* in (1) could reduce to *ve* and cliticize with the adjacent *I* forming (2) which can be further reduced to *been* as in "been there done that."

(1) I have been

(2) I've been

(3) been<sup>40</sup>

Nevertheless, with TIMIT anyway, the mixed syllable/triphone model resulted in a 28% reduction in WER over an acoustic model built with triphones alone. This reduction disappears when using canonical pronunciations, rather than the manually transcribed pronunciations found in TIMIT. A likely explanation is that the introduction of syllables

---

<sup>39</sup> *Corpus Gesproken Nederlands* "is a database of contemporary standard Dutch spoken by adults in The Netherlands and Belgium. It consists of nearly 9 million words (800 hours of speech), of which approximately two thirds originate from The Netherlands and one third from Belgium" (Hamalainen et al. 2007, p.1)

<sup>40</sup> I owe (3) to William Croft of my committee who pointed out that *I've gone* probably reduces in speech to *gone*.



results in a loss of context information on either side of the syllable, context information not present in canonical pronunciations.

### **2.3 Syllables in the Language Model**

There is very little published work on the use of syllables in the language model. Since the primary job of a recognizer is to transform speech to text, it is the language model that provides the collection of probabilities for textual (word or syllable) sequences. Further, the speech recognizers include language models only because acoustic models perform so poorly. If there were a perfect alignment between the expected acoustic input and the actual input, the language model would be unnecessary, except, of course, to deal with homophones. Still, there are specific applications that do not need word sequences in the language model. I will consider three.

The team at the University of Colorado's Center for Spoken Language Research, also responsible for the SONIC LVCSR (SONIC, 2007), has experimented with the use of syllable lattices in a children's reading tracker. Reading trackers use speech recognizers to determine if a child has read a pre-defined text correctly. The key here is that the text is pre-defined. The task of the tracker is to determine if the child's speech input aligns with text, what is referred to as "forced alignment" in the speech recognition literature. Notice that what I have been calling the primary job of a speech recognizer, the transformation of acoustic input to text, is not necessary in this case. The text already exists. The tracker's job is to find out if the child has read it correctly.

One problem with reading trackers geared to children is the great variability in their speech (Bolanos, et al., 2007b). As a result, the development of accurate acoustic models is difficult. Analysis of children's reading corpora show that most repetitions and repairs occur

at the syllable level (“when his ow- owners got him a puppy” (Bolanos, et al. 2007b, unpaginated)). This makes syllables a promising recognition unit for this purpose. A system developed at the University of Colorado’s Center for Speech and Language Research consisted of 2314 syllables for American English. Each syllable had one or more phones associated with its pronunciation. The acoustic model consisted of a lattice generated from a set of 55 context-dependent phones. The NIST syllabifier was used to generate multiple syllabifications based on multiple pronunciations. The language model consisted of a trigram syllable lattice generated from the multiple pronunciations. The system was augmented by a machine learning algorithm known as a *support vector machine*, a technique used in recent years to classify phones (Bolanos, et al., 2007a). The Colorado system reduced the combined false-acceptance/false-rejection error-rate by 18% over systems using word-based language models (Bolanos, et al., 2007b).

Syllables have also been used in the language model of a system that recognizes spoken names. The application is important in automated directory assistance and caller identification. Spoken names are particularly difficult for speech recognition systems, especially in a country like the United States where names are of multiple ethnic origins. Each name can have multiple pronunciations, as anyone who has ever called a roll in the United States will attest. As with the children’s speech tracker, the task is not to output text, but rather to determine if the input matches a stored name. Sethy and Narayanan (1998) developed a name recognition system that used syllables in both the acoustic and language models. The arguments for syllables and details of the system are almost identical to those we have already examined, right down to the use of the NIST syllabifier. The authors report

a 19.6% improvement in recognition accuracy over a conventional context-dependent triphone system.

Possibly the most relevant experiment with syllables in the language model is that reported in Schrumpf, et al. (2005). The authors are primarily concerned with the problem of indexing audio files in order to make efficient retrieval possible. That is, each audio file is assigned a name. Speaking the name allows for retrieval of the audio files. A currently available application of such a system is SYNC, developed by Microsoft for FORD (SYNC, 2010). Since words are typically used as the indexes, words that are not part of the recognizer's vocabulary inevitably occur. This requires updating both the vocabulary of the recognizer and material already in the audio database. The use of syllables as indexes for recorded speech is one alternative, both because they are smaller than words, and because a modest set of syllables, used as components, can generate a much larger set of words.

A further benefit is that by using an estimate of similarity, a syllable-based language model can correct an incorrect hypothesis generated by the acoustic model. Suppose, for instance, that the hypotheses for the word *Beslan*<sup>41</sup> include *best lan* and *bess line*. Were each of the hypotheses represented as syllables—*best-lan*, *bess-layn*—each would share a syllable with the query word, now syllabified as *bes-lan*. Since *best* deviates from *bes* and *layn* deviates from *lan*, the degree of almostness can be calculated. The authors experimented with the Boston University Radio Speech Corpus and, for the language model, 100,000 words drawn from a corpora of news wire services. They decomposed these words using the same NIST syllabifier (NIST 2007) that is part of the current study. The best performance was a syllable recognition rate of 87.7%, occurring, however at 30.07 times real time. This

---

<sup>41</sup> The example is from Schrumpf et al. 2005.

work is very promising, especially since the time can be reduced through parallelization. The authors take care to point out that syllable-based language models are useful only in special purpose environments, like indexing, since the syllable-based output is not human readable.

Besides using a syllable language model, these three examples share a further feature. The applications described have little in common with LVCSR. The final two are not LVCSR systems at all. This underscores how little research has been done on the use of syllables in the language model of LVCSR systems. The reason for this is that the language model is the collection of word sequence probabilities. The output of the recognizer is stitched together from the contents of the language model. If the language model consists of syllable sequence probabilities, it will have to undergo further processing before it is useful. One possibility is to use a parser to transform the syllable output back to words. This would have to be probabilistic, of course, since the collection of syllables in the output will not necessarily map to a sequence of words. This is exactly what is attempted in Experiment 2.1. As will be explained in Section 4, this is not the best approach. Sections 5 and 6 argue for better ways of handling syllable output.

## **2.4 Equivalence Classes in Speech Recognition**

As noted above, whatever the discourse rules are for speech in general (recognizing that they differ by language, of course), they are simpler in human-machine interaction. There is, for example, no reason to distinguish between the polite form of a request in English (“I would like ...”), the considerably less polite form (“I want...”), and the most direct form of all (“Give me ...”). From the machine’s point of view, if there is such a thing, all three are identical, in essence, they are members of an equivalence class, WANT. We discuss the formation of equivalence classes in Section 5. For now, we attribute the initial

observation that collections of words and phrases might be profitably grouped together to researchers at the University of Colorado's Center for Spoken Language Research. Their work is also an extensive use of a knowledge-base to augment speech recognition, an investigation that is not part of this study, but a topic for further research (Pellom, et al., 2001; Hacıoglu and Ward, 2002).<sup>42</sup>

Before ending this section, it should be mentioned that collapsing collections of words into equivalence classes is superficially like an old problem in artificial intelligence and computational linguistics research, indeed in linguistics proper, that of reducing the details of language through the search for language universals. Before the 1960s, this was not always a respectable enterprise. Forty years ago, Fillmore (1968, p. 1) recalled "a Linguistic Institute lecture ... in which it was announced that the only really secure generalization on language that linguists are prepared to make is that 'some members of some human communities have been observed to interact by means of vocal noises.'" Times change. The knowledge representation enterprise among artificial intelligence researchers in the 1970s and 1980s can be viewed as a quest for universals. The classic exposition is Roger Schank's conceptual dependency theory developed in the 1970s. It provides a collection of primitives out of which meaning is constructed. Thus, in conceptual dependency theory, there are four basic frameworks: ACTs (actions), PPs (objects), AAs (modifiers of actions),

---

<sup>42</sup> The system described in Pellom et al. (2001) and Hacıoglu and Ward (2002), is far more ambitious than the current study. It contains, among many other components, a speech recognizer and synthesizer, an audio server that handles incoming calls, a database for information lookup, a scoring mechanism that goes far beyond SER, a language understanding subcomponent that sends the output of the recognizer onto a parser that then maps its output to slot and filler scheme (see Luger, 2009), and a dialogue manager that handles the interaction between the user and the other subcomponents. As such, its only point of intersection with our own study is the hypothesis that words and phrases can be collapsed into equivalence classes. Our goals are considerably more modest than the text comprehension which appears to be the goal of Pellom and colleagues. We argue only that we can improve the SER of a current recognizer by including a syllable language model and post-processing concept component.

and PAs (modifiers of objects). All actions fall into the class ACT. They include, for example, ATRANS, PTRANS, and PROPEL that, in turn, collect meanings related to giving, going, and pushing. The result, according to Luger (2009), is that sentences with the same meaning can be represented internally with the same structure. Shank's system, in turn, has a familial relationship to Fillmore's proposals on case structure grammars. His use of "covert categories" to include members which lack "morphemic realizations but whose reality can be observed on the basis of selectional constraints and transformational possibilities" (p.7) can be read in the same spirit as Schank's work—an attempt to simplify the stunning complexity of human language.

Unlike both Schank and Fillmore, we make no claims about cognitive structure or linguistic universals. We simply observe that the full range of human language, the subjunctive, for instance, as noted above, is not necessary in all situations. In particular, when speaking to a machine in a restricted domain, one might get by with quite a severely scaled back language. Researchers into natural language processing and speech recognition in particular can either attempt to build a model that does justice to human language—Schank's apparent attempt—or to restrict the language used to the task at hand. The latter is the concept hypothesis, further explored in Section 5.

At this point one might object that a system using a severely restricted language model is no longer a large vocabulary system. But this is not the case. The acoustic model used in Experiment 2 (Section 4.0) was trained on the Macrophone corpus, consisting of about 200,000 utterances from 5000 speakers (Macrophone 2010). It is only the language model that is restricted to a few hundred words. The language model serves as a specific context for a larger language represented by the acoustic model. Viewed this way, the language

model is analogous to discourse context set up by conversants, while the acoustic model is analogous to the full language shared by the speakers.

### 3.0 Experiment 1, Perplexity: A First Look at Syllables in the Language Model<sup>43</sup>

As pointed out in Section 1, the current study is an investigation of alternative language models and a post-processing feature called the *concept model*. An operational LVCSR with a fully-trained acoustic model is the starting point for the investigation summarized in this thesis. A language model, as already noted, is a statistical representation of word frequencies. Its use flows from the non-controversial observation that all languages have syntactic, morphological, semantic, and pragmatic patterns that make some word sequences more probable than others. The use of language models in speech recognition systems is to bring to bear, implicitly, linguistic information when generating hypotheses about which sequences best map to a given acoustic signal (Holmes and Holmes, 2001). The experiment described in this section compares the perplexity of the syllable and word language models over two corpora, a base-line large corpus and a smaller, more specialized corpus that will be used in the experiments described in Sections 4 and 5.

Perplexity computes the inverse probability of randomly long and randomly chosen word sequences. Absent an empirical test of a language model, it can be used as an *a priori* estimate of how well a given language model predicts a given corpus. This deserves some explanation. Given a language model, why would we not just test it, that is, install it in a recognizer and see how it works? The problem is that constructing a realistic test set is no

---

<sup>43</sup> The discussion of perplexity and entropy is adapted from Jurafsky and Martin (2009). It is included here to provide background in information theory necessary for the results reported in Section 3.1.3 and the discussion in 3.1.4.



simple procedure. It could take days to collect and transcribe recordings and then to gather data and evaluate how the recognizer performed. Perplexity offers a short cut. It lets us computer-generate test sentences, and compute their probabilities given our language model. Good perplexity scores do not functionally predict good symbol error rate scores. That is, generating an artificial test set and computing its probability based on a training set does not necessarily predict how well a language model will perform in practice. It is, however, an indication, the way clouds in the sky tell us we should probably carry an umbrella, not precisely that it will rain.

In the experiment described below, the test sentences are randomly chosen from the training transcript for the language model and then removed. This gives us a test set and a training set. We then compute the perplexity for each of the sentences in the training set and average the results. This serves as an *a priori* estimate of how well a training set predicts a test set. It is not perfect, of course. Since the test set and training set are drawn from the same transcript, the fit will be better than, say if the training set were drawn from *Wuthering Heights* and the test set from the *New York Times*. But this is as it should be. We wouldn't expect the fit between a Victorian novel and a contemporary newspaper to be especially good. The training set and the test set ought to be somewhat alike without being identical (since that would be cheating).

For our purposes, perplexity is of interest for two reasons. First, since perplexity can be viewed as the weighted average branching factor of a language model, it provides a nice intuitive justification for the syllable hypothesis overall. Imagine driving from Albuquerque to Santa Fe. Though there are many decisions to make, suppose that once you get on the freeway, you reach three decision points ("bear left," "turn right," etc. for each of the three

decision points). Point A offers two choices, B four, and C two choices. Then the average branching factor for your trip is 3.<sup>44</sup> Manning and Schutze (1999) offer this analogy: “A perplexity of  $k$  means that you are as surprised on average as you would have been if you had to guess between  $k$  equiprobable choices at each step” (p. 78). Second, perplexity is closely related to cross-entropy as understood in information theory.<sup>45</sup> It has a pedigree that extends back to Shannon’s original work in the 1950s and, so, pre-dates the architecture of the current generation of speech recognizers by four decades (Jurafsky & Martin 2009).

Perplexity is defined as the  $N$ th inverse root of the probability that a language model assigns to a sequence of words:

$$PP(W) = p(w_1 w_2 \dots w_n)^{-1/n} \quad (1)$$

where:

- $W$  is a sequence of  $n$  words,  $w_1 w_2 \dots w_n$
- $p(w_1 w_2 \dots w_n)$  is the probability that a language model assigns to that sequence
- $n$  is the number of words in the sequence

What is really meant by the probability of a sequence of words,  $W = w_1 w_2 \dots w_n$ ?<sup>46</sup>  $p(W)$  is simply a probability distribution over sequences of words that describes how frequently a given sequence occurs as an utterance. So, if the  $p(\textit{hello}) = .001$ , we are asserting that about one of every 1000 utterances is the sequence *hello*. The question is how could we compute

---

<sup>44</sup>  $(3 + 4 + 2)/3$

<sup>45</sup> In fact, very closely related as we will see. Manning and Schutze (1999) doubt the motives of the ASR community. “We suspect that speech recognition people prefer to report the large non-logarithmic numbers given by perplexity mainly because it is much easier to impress funding bodies by saying ‘we’ve managed to reduce perplexity from 950 to 540’ than by saying ‘we’ve reduced cross-entropy from 9.9 to 9.1 bits’ (p. 78).

<sup>46</sup> Bearing in mind Chomsky’s famous 1969 assertion in an essay on Quine that “it must be recognized that the notion ‘probability of a sentence’ is an entirely useless one, under any known interpretation of this term” (quoted in Jurafsky and Martin, p. 83).

such a probability? The answer is that we cannot in the general case, but we can use a corpus of utterances, along with some results from probability theory, to estimate it.<sup>47</sup>

$p(W)$  can be written as:

$$\begin{aligned} p(W) &= p(w_1 w_2 \dots w_n) \\ &= p(w_1) * p(w_2 | w_1) * p(w_3 | w_1, w_2) \dots p(w_n | w_1, w_2, \dots, w_{n-1})^{48} \quad (2) \\ &= \prod_{i=1}^n p(w_i | w_1 \dots w_{i-1}) \end{aligned}$$

So,

(1) can be rewritten as:

$$\begin{aligned} PP(W) &= \sqrt[n]{\frac{1}{p(w_1 \dots w_{i-1})}} \\ &= \sqrt[n]{\frac{1}{\prod_{i=1}^n p(w_i | w_1 \dots w_{i-1})}} \quad (3)^{49} \end{aligned}$$

---

<sup>47</sup> See Appendix A for a tutorial on probability theory

<sup>48</sup> See Equation 1.10

<sup>49</sup> In practice, we estimate the perplexity of  $W$  using a given language model, whether unigram, bigram, trigram, or some higher order. Here is (2) rewritten to compute the perplexity of a word string with a bigram language model:

$$PP(W) = \sqrt[n]{\frac{1}{\prod_{i=1}^n p(w_i | w_{i-1})}}$$

There is a problem here. Suppose one of the component probabilities in the denominator is 0. That is, suppose one of the bigrams does not appear in the training set, resulting in a 0 denominator. The technique used to avoid the problem is called smoothing. The perplexity generator used in this experiment uses the Katz backoff algorithm which has the effect of distributing the total probability mass to N-grams with 0 counts (Huang et al. 2001, Jurafsky and Martin 2009).

Notice that the higher the probability of a word string, the lower its perplexity. This is the basic relationship to keep in mind. What does this have to do with the weighted average branching factor? A simple way to see this is to consider the task of recognizing a sequence of upper case letters. If each of the letters occurs with equal frequency, the probability of any given letter is  $1/26$ . Therefore,

$$\begin{aligned}
 PP(W) &= p(w_1 w_2 \dots w_n)^{-1/n} \\
 &= \left(\frac{1}{26}\right)^n^{-1/n} \\
 &= \left(\frac{1}{26}\right)^{-1} \\
 &= 26
 \end{aligned}
 \tag{4}$$

Suppose we know that the letter *E* occurs 75 times more frequently than any of the other 25 letters. Now, let the probability of any letter other than *E* be denoted,  $x$ . Then the probability of *E* is 75 times the probability of any other letter, i.e.,  $75*x$ . Since the probability of either *E* or any other letter is 1, we can write:  $75*x + 25*x = 1$ , and  $x = .01$ . Since any letter,  $w_i$  in equation (4), is either *E* or one of the other 25 letters, the probability of any letter at all is just the probability of *E* plus the probability of that other letter, giving:

$$p(w_i) = .75 + .01 = .76 = 76/100$$

Substituting  $76/100$  for each  $p(w_i)$  in (4) gives a perplexity of  $100/76 = 1.316$ . Notice that the branching factor remains the same. At every fork in the road, we still have 26 possibilities. But since *E* is 75 times more likely at every fork, the perplexity is dramatically reduced.

Now for entropy, which we informally define as the number of bits it takes to encode a piece of information. If  $X$  is a random variable ranging over whatever we are trying to predict, the complete collection of which we designate by  $Q$ , then its entropy is:

$$H(X) = -\sum_{x \in Q} p(x) \log_2 p(x) \quad (5)$$

We offer a simple example borrowed from Jurafsky and Martin (2009) who, in turn, borrowed it from a classic text on information theory, Cover and Thomas (1991). Suppose we want to place a bet on a race in which there are eight horses without going to the track. What is the minimal amount of information we need to send to our bookie to tell him which horse we want to bet on? Since there are 8 horses, each can be represented in three bits: 000, 001, 010, ..., 111. In fact,  $H(X)$  is three if the probability of our betting on each horse is the same, namely,  $1/8$ . Since

$$\log_2 \frac{1}{8} = -3$$

Then,

$$\begin{aligned} H(X) &= -\sum_{x=1}^8 p(x) \log_2 x \\ &= -8 (-3/8) = 3 \end{aligned}$$

Now suppose that the probability that we will bet on various horses differs according to this table:

Horse 1	$\frac{1}{2}$
Horse 2	$\frac{1}{4}$
Horse 3	$\frac{1}{8}$
Horse 4	$\frac{1}{16}$
Horse 5	$\frac{1}{64}$
Horse 6	$\frac{1}{64}$
Horse 7	$\frac{1}{64}$
Horse 8	$\frac{1}{64}$

Then:

$$\begin{aligned}
 H(X) &= -\frac{1}{2}\log_2 \frac{1}{2} - \frac{1}{4}\log_2 \frac{1}{4} - \frac{1}{8}\log_2 \frac{1}{8} - \frac{1}{16}\log_2 \frac{1}{16} - 4\left(\frac{1}{64}\log_2 \frac{1}{64}\right) \\
 &= 2
 \end{aligned}$$

We can encode our horse names averaging 2 bits by encoding the more probable horses with shorter bit strings:

Horse 1	0
Horse 2	10
Horse 3	110
Horse 4	1110
Horse 5	111100
Horse 6	111101
Horse 7	111110
Horse 8	111111

But, of course, we are concerned with the entropy of sequences of words, the full collection of which forms a language,  $L$ . This requires only a slight modification to (5):

$$H(W) = -\sum_{W \in L} p(W) \log_2 p(W) \quad (6)$$

We can divide the entropy of a sequence by the number of words in the sequence to arrive at the entropy rate:

$$H_{rate}(W) = \frac{1}{n} H(W) \quad (7)$$

Given some simplifying assumptions, it can be shown that (6) can be rewritten as :

$$H(W) = -\frac{1}{n} \log_2 p(w_1 \dots w_n)$$

Now we see that perplexity and entropy are intimately related, namely the antilog of  $H(W)$  is:

$$p(w_1 \dots w_n)^{\frac{1}{n}}$$

But this expression is perplexity,  $PP(W)$  as defined in (1). Since the antilog is an inverse of an inverse:<sup>50</sup>

$$\text{antilog}_2(H(W)) = 2^{h(w)} = p(w_1 \dots w_n)^{-\frac{1}{n}} = PP(W) \quad (8)$$

So the perplexity of a word sequence is 2 raised to the power of its entropy. We will return to entropy in section 3.1.4.

### 3.1.0 Materials

The materials used in this experiment are:

1. Two Corpora

- a. Appendix C

- b. Air Travel Information System (ATIS) (Hemphill et al., 2009)

Both the corpus in Appendix C and the ATIS corpus were collected using the Wizard-of-Oz protocol (WOZ).<sup>51</sup> See Appendix C for a full description. The WOZ protocol is common in the development and training of LVCSR systems. The idea is to have a person, the wizard, carry on a conversation (by keyboard and monitor) with one or more persons at another location who have not been told that they are interacting with a human. The resulting transcript, is cleansed of the wizard's responses, leaving only what engineers hope is a typical human-computer interaction in the problem domain. The text in Appendix C forms the basis for the language models used in Sections 3, 4, and 5. The conversants in this case were temporary workers hired by Next It to help construct a language model for an LVCSR (Next It, 2008; Huang et al., 2001).

---

<sup>50</sup>  $\text{Log}_b x$  is the inverse of raising  $b$  to the  $x^{\text{th}}$  power. So,  $x = \text{Log}_b b^x$ . The antilog is the inverse of the log. So,  $\text{antilog}_b(x) = \text{antilog}_b(\text{log}_b b^x) = b^x$  (Weisstein, E., 2009a, 2009b)

<sup>51</sup> There is some possibility that the WOZ protocol was not used in creating the Appendix C transcript. Rather Appendix C is a transcription of Next It employees asked to simulate a conversation with a virtual travel agent (See Appendix C; Charles Wooter, personal communications 3/7/2010). The mode of collection makes little difference in this case. The role of the transcript was, in effect, to bootstrap a system that did not yet exist. As a system is used, more complete language modeling transcripts can be generated. These were used to get the system running in the first place. For this study, a common language model useable across every experiment was all that was needed. Appendix C served this purpose.



In the case of ATIS, forty-one subjects, all employees of Texas Instruments, were given instructions like the following, “Plan travel arrangements for a small family reunion. First pick a city where the get-together will be held. From 3 different cities (your choice), find travel arrangements that are suitable for the family members who typify the ‘economy,’ ‘high class,’ and ‘adventurous’ life styles” (Hemphill, et al, 2009, p. 3). The subjects operated a system simulating a virtual travel agent. The simulator consisted of a monitor, both head-mounted and desk-mounted microphones, and a computer running an X Windowing system.<sup>52</sup> Hemphill and colleagues collected 1041 utterances over an eight week period. Further details about recording and transcriptions may be found in Hemphill et al. (2009).

	Word Types	Word Tokens	Syllable Types	Syllable Tokens
Appendix C	482	5782	537	8587
ATIS	1604	219009	1314	317578

2. Software developed for this project as well as software available from the the National Institute of Standards (NIST, 2007), the Center for Speech and Language Processing of the University of Colorado (SONIC, 2007), and Next IT Corporation (Next IT, 2008).

	Function	Source
Syllabifier	Syllabifies words	NIST
Sspell	Phonetically transcribes words	SONIC
Language model generating tools	Builds language model and computes perplexity	Next IT
PrplxTst.py, Prplx.py	Controls NIST, SONIC, Next IT software	Appendix I

---

<sup>52</sup> X Windows is a client-server windowing protocol commonly found on unix/linux machines.

### 3.1.2 Specific Techniques

The technique used is to create a test and training set from the same collection of utterances. The fraction of the collection used in the test set is a parameter. The results reported here use 10% of the collection in the test set and the remaining 90% in the training set. Specifically, for each of twenty times, a file of utterances is read into a Python list. 10% of these utterances are randomly chosen to be part of the test set. A language model is generated using the remaining 90% of the utterances and the perplexity of the test set, given the language model, is computed. The system computes the mean, median, and standard deviation of the twenty runs. These computations are done for both word and syllable language models using unigram, bigram, trigram, and quadrigram language models.

As a baseline against which all of N-gram models are normed, we first compute the perplexity of an unweighted language model, that is one in which any word/syllable has the same probability as any other. We did this by using the derived expression for perplexity given in equation (8). A language model in which all words/syllables have the same probability is the language model in which the number of word/syllable types equals the number of word/syllable tokens. Finding this set is easy enough. We simply sort the words in the transcript and eliminate duplicates. This leaves us with a transcript in which there is a single entry for each word type. Since the Appendix C transcript has 482 word types, the probability of any word being chosen at random is  $1/482$ . By (8), the perplexity of the language model can be written as:

$$\begin{aligned}
 PP(W) &= p(w_1 \dots w_n)^{-\frac{1}{n}} \\
 &= \left[ \left( \frac{1}{482} \right)^n \right]^{-\frac{1}{n}} \\
 &= 482
 \end{aligned}$$

By a similar computation, the perplexity of the unweighted language model for the Atis transcript is 1604. This means simply that were the system to guess a word, having only the unweighted language model at its disposal, the probability of guessing the correct word would be 1/482 using the Appendix C transcript and 1/1604 for the much larger Atis transcript.

### 3.1.3 Results

Tables 3.1, 3.2 show the results of perplexity testing over different N-gram sizes for word and syllable language models. Tables 3.3 and 3.4 shows the means normed by the perplexity of the unweighted—the flat—language model. This technique treats the flat language model as the base line, giving it a value of 1. The other values are then fractions of the base line. This allows us to appreciate the value of information in the language model—the more information, the better, though the rate of improvement drops for quadrigrams.

**Table 3.1: *Perplexity, Word Language Model***

	Flat LM	N = 1	N = 2	N = 3	N = 4
Mean Appendix C	482	112.56	39.44	31.44	30.42
Median, Appendix C	482	110.82	38.74	28.96	29.31
Std Dev Appendix C	0.0	13.61	7.21	5.82	4.05
Mean ATIS	1604	191.84	41.35	31.51	31.43
Median ATIS	1604	191.74	38.93	28.2	30.19
Std Dev ATIS	0.0	.57	8.64	7.6	4.32

**Table 3.2: *Perplexity, Syllable Language Model***

	Flat LM	N = 1	N = 2	N = 3	N = 4
Mean Appendix C	537	177.99	35.96	22.26	21.04
Median, Appendix C	537	177.44	35.81	22.71	20.75
Std Dev Appendix C	0.0	1.77	8.49	3.73	3.53
Mean ATIS	1314	231.13	22.42	14.91	14.11
Median ATIS	1314	231.26	21.15	14.74	13.2
Std Dev ATIS	0.0	.097	4.36	1.83	1.79

**Table 3.3: *Mean Perplexity of Normed Word Language Models***

	N = 1	N = 2	N = 3	N = 4
Mean Appendix C	23.35%	8.18%	6.52%	6.31%
Mean ATIS	11.95%	11.96%	1.96%	1.95%

**Table 3.4: Mean Perplexity of Normed Syllable Language Models**

	N = 1	N = 2	N = 3	N = 4
Mean Appendix C	33.14%	6.69%	4.15%	3.92%
Mean ATIS	17.58%	1.71%	1.13%	1.07%

To clarify, in Table 3.3, the cell indicated by the column labeled “N = 1” and row labeled “Mean Appendix C,” is computed by dividing 112.56 by 482, where 112.56 is the mean perplexity for word unigrams, and 482 is the number of distinct words in Appendix C. The data in the cell tells us that using unigrams resulted in a perplexity of 23.35% of the perplexity that would have resulted had we used a flat language model. The other cells in Tables 3.3 and 3.4 are computed in the same fashion.

#### **3.1.4 Discussion**

The least surprising feature of the results is that the perplexity of both the word and syllable models for both corpora drop sharply as the N-gram size increases up to 3, beginning with the use of bigrams (as indicated in Tables 3.3 and 3.4). As Jurafsky and Martin (2009, p. 97) observe, “the more information the N-gram gives about the word sequence, the lower the perplexity.” For sizes  $> 3$ , the improvement is less dramatic.

For the purposes of the syllable hypothesis, the most important item is the relative perplexity of the words and syllables for bigrams and trigrams, which is reproduced in Table 3.5.

**Table 3.5: Perplexity of Word and Syllable Models for Bigrams and Trigrams**

	N = 2	N = 3
Mean Appendix C, Words	36.36	19.81
Mean Appendix C, Syllables	30.65	11.71
Mean ATIS, Words	33.04	18.53
Mean ATIS, Syllables	18.37	9.97

The perplexity of the syllable language models are considerably less than their word counterparts. This is most dramatically true for ATIS bigrams, where the drop is nearly 45%. Put another way, a recognizer equipped with an ATIS bigram language model would have almost half as many branches to consider on its way to guessing the next word. The relative improvement in perplexity by using a syllable language model is perhaps best distilled in the bigram and trigram columns of Tables 3.3 and 3.4. The drop in perplexity normed by the flat language model is very steep, particularly for bigrams and trigrams.

Put still another way, consider the definition of perplexity in terms of entropy, given in equation (8). Perplexity increases exponentially with entropy. Recall the horse race example. When the system included the probability of choosing a particular horse, it had less entropy. That is, we were able to encode the name of the horse with fewer bits. Systems of higher entropy have less information. The two syllable language models presented appear to encode more information than their corresponding word language models.

Unigrams do not follow this pattern. The perplexity for both of the corpora is considerably less for the word model than for the syllable model. Had we considered only Appendix C, a first pass at a hypothesis might start with pointing out one odd characteristic

of the corpus, namely that there are more syllable types (537) than word types (482). This would not be the case with a random sample of English speech. Nevertheless, the ATIS corpus follows the expected pattern, having more word types than syllable types. Following our discussion of perplexity and entropy, we are forced to conclude that there is more information in a unigram word language model than in a syllable language model. In fact, this can be computed. We start by collecting word and syllable types across both corpora. We then compute the probability of each type. Given the collection of probabilities, it is a simple matter to sum the product of log probabilities over each collection and divide by the number of types in each to produce the entropy rate. The figures for the corpora are shown in Table 3.6:

**Table 3.6: Comparison of Entropy Rates for Syllables and Words**

	Syllables	Words
Appendix C Corpus	.0139	.0117
ATIS Corpus	.0059	.0046

The entropy rate computation confirms the previous hypothesis. Though syllable N-grams,  $N > 1$ , carry more information than word N-grams, and though the larger the corpus, the more information each of its components contains, words alone embed more information than syllables alone. Said another way, one would be more likely to guess a random word than a random syllable. For  $N = 2$  or  $3$ , we can speculate that phonotactic patterns, the very patterns that would affect sequences of syllables, operate over shorter distances than do syntactic patterns. Were we to extend  $N$  to greater values, we might find the perplexity of syllables relative to words growing. For  $N = 1$ , however, neither phonotactic nor syntactic

patterns play a role. The most frequently appearing words have a higher probability than the most frequently occurring syllables. Thus in the ATIS corpus, *to* appears 12,232 times for a probability of .0559, while the syllable *tuw* appears 13,477 times for a probability of .0424. The patterns holds for the least frequently occurring word (“abbreviated”) and syllable ([ae]), with probabilities of  $4.566 \times 10^{-6}$  and  $3.149 \times 10^{-6}$  respectively. The results appear to indicate that words, when considered alone, repeat with a greater probability than do syllables.



## 4.0 Experiment 2: A Syllable Language Model

Syllables in the acoustic model have been studied extensively. Though promising theoretically, the results have been positive but not dramatic. The problem seems to be that much of the contextual information captured by syllables has already been captured through the use of triphones. This experiment and the one described in Section 5, instead, look to the relatively understudied use of syllables in the language model. It rests on a single observation. Neither humans, nor machines in most instances, need an exact transcript of conversational utterances.

Once freed from the obligation to render speech into words, the need for a word-based language model disappears. Given an appropriate metric, we hypothesize that a speech recognizer whose language model contains syllables will perform better than one which contains words<sup>53</sup>. Testing this hypothesis is the goal of the experiments described here and in Section 5. For the experiment described in this section, the metric to be used is the symbol error rate (SER, see Section 1.07). Though this is the roughest of measures, it gives us some idea of how a recognizer works in the limiting case, that is, when it is scored against a perfect transcription of what it is asked to recognize.

Experiment 2 has several variations. Experiment 2.1 compares the SER of the syllable language model with the word language model over several audio recordings and

---

<sup>53</sup> Since syllables form a reduced search space when compared with words, the hypothesis is not difficult to believe, even without supporting evidence. On this score, see the role of anticipated results in “Normal Science” in Section 1.2. Nevertheless, if the hypothesis is not difficult to believe, one has to wonder why few, if any systems, use syllables in the language model. The answer appears to be, as noted previously, that a syllable-based language model results in syllable output, not usually what is desired. In the case of this investigation, however, the output of the recognizer undergoes further (demonstrated and proposed) processing.

using several N-gram possibilities. Since a recognizer equipped with a syllable language model produces syllable output, the question arises as to how to compute the SER, given conventional word-based reference files. The approach of Experiment 2.1 is to transform the syllable output back to its word form. Since a word language model produces word output, we now have two word outputs that can be compared with the reference files to compute the SER. But this is not exactly a fair comparison, since the syllable language model has to go through an additional transformation. To guard against the possibility of the parsing program that transforms syllables to words introducing error, we construct Experiment 2.2. This time instead of transforming the syllable output to words, we syllabify the reference files. We can then run the scoring program.

Since we are about to use an LVCSR system for the first time in this study, it might be useful to review the major components. The role of the language model in speech recognition is illustrated in Figure 4.1, adapted from Young (1998).

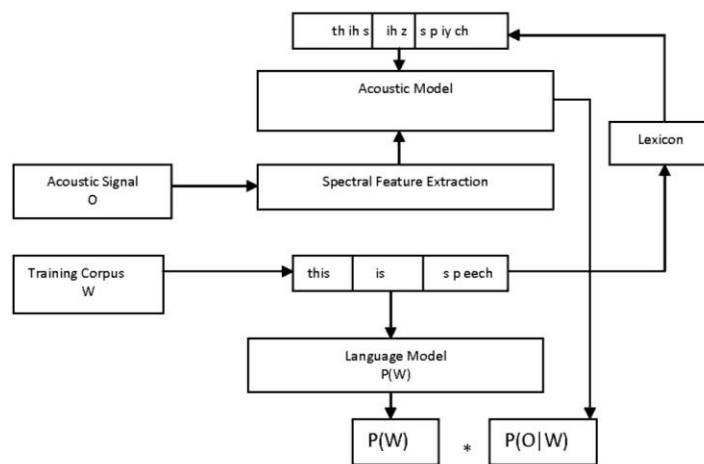


Figure 4.1 Some Components of a Recognizer

**Figure 4.1: Some Components of a Recognizer**

To understand Figure 4.1, refer to Subsections 1.5, 1.7.1, 1.7.2, 1.7.3, and 1.7.4. for a more complete discussion. We begin with the training corpus. Figure 4.1, for the sake of simplicity, makes it look as if there is a single training corpus for both the acoustic model and the language model. This is not the case. It takes many hours, sometimes days, to train an acoustic model. This study assumes a fully trained acoustic model and working recognizer. For the purposes of documentation, nevertheless, the acoustic model was trained on the Macrophone corpus, managed by the Linguistic Data Consortium and housed at the University of Pennsylvania. It consists of 200,000 utterances by 5000 speakers, distributed evenly by gender, income, and region of the United States. The ages of the speakers range from teenagers to people in their seventies. The utterances were collected by telephone in 8 kHz, 8-bit mu-law format (Macrophone, 2010). The lexicon consists of words and their phonetic transcriptions. Spectral feature extraction is the process of transforming the acoustic signal into quantized feature vectors. One step in the training process is to use the lexicon to align the feature vectors with a phonetic representation of the signal. The other step is to compute the probabilities of acoustic observations (represented as feature vectors) given a word (represented as subphone transition probabilities). The language model is a text-based collection of word sequence probabilities (e.g., given “hi” how probable is it that the next word is “there?”). Language models are considerably easier to train than acoustic models and can be freely exchanged as the application demands. In this study, we are holding the acoustic model constant and varying the language model.

Given the prior probability of word sequences computed by language model,  $P(W)$ , and the observation likelihood of an acoustic sequence given a word sequence ( $P(O|W)$ ),

using Bayes' Theorem,<sup>54</sup> we can compute the probability of a word sequence given a sequence of acoustic observations. If we replace the word N-grams of the conventional language model with syllables, and substitute a syllable-based lexicon for the word-based lexicon, it would be possible to test the hypothesis that syllables will achieve better results than words in the language models. The idea here is that the LVCSR will render an acoustic sequence into a syllable sequence.

There are eight steps in this experiment:

1. Phonetically transcribe a training transcript (see Appendix C).
2. Insert stress markings (See Section 4.6. experiment 2.3 only)
3. Syllabify the transformed transcript
4. Compute a syllable language model from the transformed transcript.
5. Run the SONIC recognizer on a selection of audio files using both the syllable and word language models.
6. Convert the output of the recognizer which is in syllables back to words (Exp 2.1 only)
7. Syllabify the reference file for the recognized audio file (Experiment. 2.2 only).

Each recording has a corresponding textual reference file. The recognizer's output is scored against the reference files. The reference files may be found in Appendix D.

8. Using *sclite* scoring software provided by the National Institute of Standards (sclite, 2009), compute the symbol error rate for the syllable and word language models.

---

<sup>54</sup> See Appendix A.

#### 4.1 Materials

The materials used in the experiment are:

1. Software developed for this project as well as software available from the the National Institute of Standards (NIST, 2007; Sclite, 2009), the Center for Speech and Language Processing of the University of Colorado (SONIC, 2007), and Next IT Corporation (Next IT, 2008).

	Function	Source
LVCSR	Speech recognition	SONIC
Syllabifier	Syllabifies words	NIST
Sspell	Phonetically transcribes words	SONIC
Language model generating tools	Builds language model and computes perplexity	Next IT
Inverse Syllabifier	Transforms syllables to words	Next IT
Sclite	Computes word error rates	NIST
Driver.py Transcribe.py, Syllabify.py LangModGen.py Recognize.py SyllablesToWords.py ChooseBest.py ScLite.py	Controls software listed above and transforms data from output format generated by one processor to the input format required by another	Appendix I

2. Eighteen audio files recorded over both standard (public switched telephone network) and mobile (cell) phones. Reference files for each audio file are found in Appendix D. The recordings were of 18 males and females, evenly distributed by sex, and recruited from employees at Next It (Next It, 2009). See Appendix D for details.

3. A training transcript found in Appendix C.

## 4.2 Specific Techniques

What follows is a more detailed discussion of the techniques used in Experiments

### 2.1 – 2.3

1. Given a file consisting of strings of words, called a transcript, write a program to produce a hashed dictionary, indexed by word, which stores the phonetic transcription of each word in SONIC's version of ARPAbet notation. The transcriptions are generated using SONIC's *spell* tool (SONIC 2007).
2. Transform the transcriptions stored in the dictionary from the phone set used by SONIC to the smaller phone set used by the NIST syllabifier. Information lost in this process must be recovered later. See step 6.
3. Pass both the transcript and the output from step 2 through the NIST syllabifier. Transform the output of this step into a lexicon, indexed by word, whose content is a list of (possibly) multiple syllabifications of each word in the transcript. When doing word recognition, this step, of course, is not necessary. The lexicon is hand-created from the original transcript.
4. Using the lexicon generated in step 3, substitute for each word in the transcript, its syllabification.

5. Transform the lexicon generated in step three from the NIST phonetic symbol set to SONIC's so that the SONIC decoder can be used. Since the function that transformed SONIC to NIST is not bijective, there is no inverse. Said another way, there is no simple way to recapture the information lost in step 2. It can be done, but with considerable effort.
6. Compute a syllable language model from the transformed transcript. The syllable language model consists of unisyllables, bisyllables, trisyllables, and quadrisyllables.
7. Exchange a syllable-based language model for the current word-based language model in the SONIC LVCSR.
8. Convert the output of the recognizer, which is in syllables, back to words. Though this seems counterintuitive, it is necessary to get a ball-park estimate of how well the syllable language model performs (Experiment 2.1).
9. As an alternative to 8, syllabify the reference file against which the output of the recognizer is scored. This is a check against the possibility that the parser that transforms sequences of syllables to words introduced error (Experiment 2.2).
10. Using *sclite*, software provided by the National Institute of Standards (SCLITE, 2009), compute the word error rate for the syllable and word language models.

### **4.3 Generating A Syllable Language Model**

In order to generate a syllable language model, we first need a training transcript. The transcript used in this experiment can be found in Appendix C, along with details of its collection. It consists of 5,782 words with an average of 5.56 characters per word. The first step is to phonetically transcribe the transcript. We did this with the *spell* utility available from SONIC. The goal is to transform the output of *spell* to a dictionary indexed by word

of SONIC transcriptions. Along the way, the phonetic symbol set used in SONIC<sup>55</sup> (see Appendix L) must be transformed to the one used by the NIST syllabifier. The most important difference is that the NIST transcription symbol set does not distinguish between word-initial consonant stops ([p] at the start of *pop*) and word-final consonant stops ([pd] at the end of *pop*). This means that each instance of a consonant transcribed as an unreleased stop closure ([pd]) must be transformed to its released equivalent:  $pd \rightarrow p$ . The syllabifier itself, as mentioned in Section 2, is available from the National Institute of Standards and implements Daniel Kahn’s widely cited dissertation (NIST, 2007, Kahn 1976). As Jurafsky and Martin point out, good language modeling toolkits are widely available. HTK from Cambridge (HTK 2009) and SRILM from SRI (SRI 2009) are examples. The language modeling toolkit that Charles Wooters built for Next IT Corporation<sup>56</sup> was used in this study.

#### 4.4 Experiment 2.1

Experiment 2.1 compares a syllable-based language model against a word-based language model. The output of the recognizer is a sequence of syllables. Each of the recognizer’s hypotheses is transformed to one or more hypothesized sequences of words. Software written for this experiment uses a word language model to choose the best hypothesis. It does this by invoking language modeling software that then computes the log-probability of each hypothesis, given the N-grams that form the language model. The software outputs the sequence of words with the highest log-probability. The log of the

---

<sup>55</sup> “...the 55-phoneme symbol set adopted by earlier versions of the CMU Sphinx-II speech recognizer” (Pellom and Hacioglu, 2005, p. 12). The SONIC symbol set can be found in Appendix L.

<sup>56</sup> Charles Wooters, of course, is on my advisory committee. He was also the Chief Technical Officer for Next IT Corporation at the time of this experiment. Next IT was precluded from using publicly available language modeling tools because it is a business. Since the research described in this document concerns the effects of using different kinds of language models, but not the language modeling software itself, it really does not matter which was used. Further research could compare the effect of using different language modeling tools.



probability of a sequence of words corresponding to the reference utterance is used instead of the probability to prevent numeric underflow and increase computational efficiency. The interested reader can refer to Jurafsky and Martin (2009, p. 313). The language model used to transform syllable sequences back to words should not be confused with the language model used in the recognizer. There, we conduct experiments with 1, 2, 3, and 4 Grams. Here, since we are just converting the output of the recognizer, i.e., the results of the experiment itself, back to a form that we can analyze, we use only a trigram language model.

#### 4.4.1 Results for Experiment 2.1

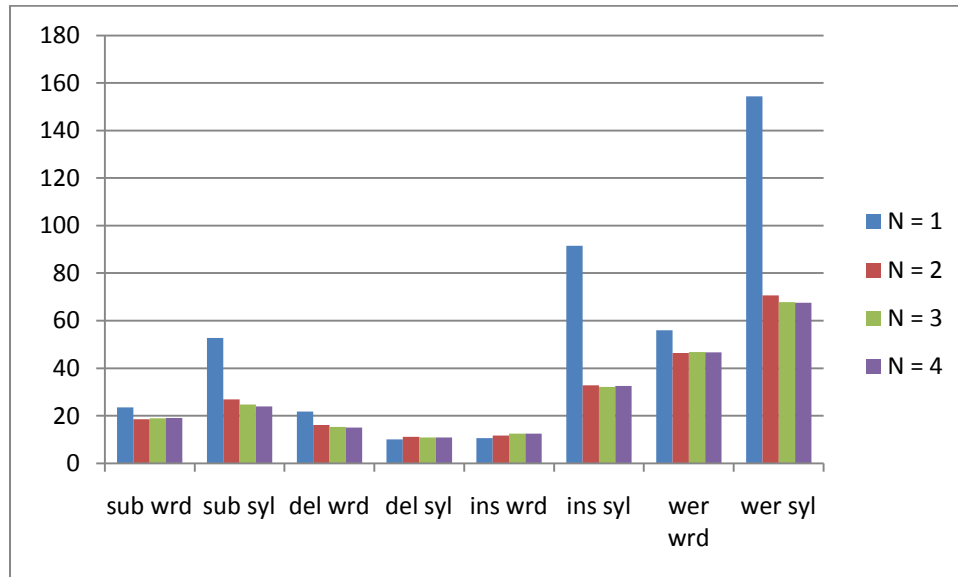
Experiment 2.1 produced eight data files, one each for N-gram = 1 through 4 for word and syllable language models. Each of the eight files consisted of error data for 18 audio files. The raw data, found in Appendix F, is summarized in Tables 4.1, 4.2, and Figures 4.2, and 4.3. The very large standard deviation is mirrored in the difference between the audio file with the best and the worst SER. This indicates the differences among speakers. For instance, with word language model trigrams, the best audio file had an 11.1% SER, while the worst was in error 86.8% of the time.

**Table 4.1: Mean Errors for Word Language Models by N-gram Size**

	Sub	Del	Ins	Mean	Median	STD	Best	Worst
N = 1	23.5	21.8	10.6	56.0	54.4	21.0	18.8	98.1
N = 2	18.5	16.2	11.7	46.4	47.8	22.7	11.1	83.0
N = 3	19.0	15.3	12.5	46.8	48.8	23.5	11.1	86.8
N = 4	19.1	15.1	12.5	46.7	48.0	23.6	11.1	86.8

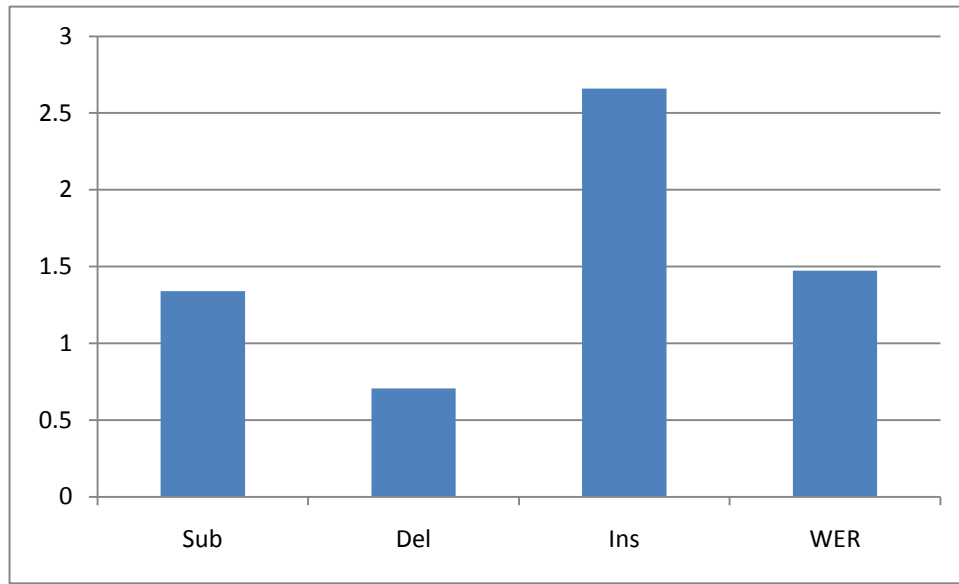
**Table 4.2: Mean Errors for Syllable Language Models by N-gram Size**  
**(Word Ref. Files)**

	Sub	Del	Ins	Mean	Median	STD	Best	Worst
N = 1	52.8	10.1	91.5	154.4	156.1	39.1	68.8	225.7
N = 2	26.9	11.1	32.8	70.7	76.0	37.4	11.1	148.6
N = 3	24.8	10.9	32.1	67.8	74.7	37.4	11.1	145.7
N = 4	24.0	10.9	32.6	67.5	72.0	37.1	11.1	148.6



**Figure 4.2: Mean Errors as a Function of N-gram Size (N = 1, 2, 3, 4)**

**In order from left to right within error type**



**Figure 4.3: Mean Ratio of Syllables to Words by Error Type**  
**Across N-gram Sizes 2, 3, 4**

#### 4.4.2 Discussion of Experiment 2.1

Of the various N-gram sizes presented, N= 1 is included only for consistency. Unigrams, of course, use no historical information at all. Guessing a token using a unigram is equivalent to the probability of simply guessing the token. Since there are fewer syllables than words in English, we would predict a lower SER for syllables than for words. This is not the case because the sample corpus is highly constrained. It has the unusual feature of containing more syllable types than word types. This accounts for the significantly higher SER error when using a syllable language model.

Figure 4.3 shows the mean ratio of the word language model to the syllable language model by error type for only the reasonable N-Gram sizes, i.e., 2, 3, 4. The syllable language model performs considerably less well than the word language model except in the single category of deletions. Though disappointing, this is not hard to understand. The reason is to

be found in point 8 of Section 4.2, namely the transformation of the syllable output of the recognizer back to words in order to compute the error rates. Specifically, the problem we were confronted with is how to compare the SER of an LVCSR using a syllable language model—and whose output are syllables—to the SER of an LVCSR using a word language model whose output is words. The approach taken in Experiment 2.1 was to convert the syllable output to words. Unless the output of the recognizer was perfect, the mapping between syllable output and words is probabilistic. As noted above, we used a parser that generated multiple hypotheses for a given utterance. We chose the best one by computing the log probability of that utterance using a trigram language model generated from the training corpus. We then output the highest log prob. The important point to notice is that technique is probabilistic. Since the LVCSR itself is probabilistic, the probability of the final output is the product of the probabilities, since the output of the LVCSR is independent of the output of the parser. If, for example, the LVCSR had a probability of correctly guessing an utterance of .8 and our technique for transforming syllables back to words had the same probability, the probability of the entire system would be .64. Experiment 2.2, which uses a different technique to compare syllable and language models, illustrates what happens when the second guess is removed from play.

#### **4.5 Experiment 2.2**

Experiment 2.2 is a replay of Experiment 2.1 but with a crucial difference in scoring. In Experiment 2.1 the word output of the word language model is compared with the syllable output transformed to words of the syllable language model. In Experiment 2.2, we take a more direct approach. The scoring software, *sclite*, available from the National Institute of Standards (SCLITE, 2009) uses a hand-produced reference file for each audio file. The word

reference file for the audio file *male9.raw* and the output of the LVCSR using trigram word and syllable language models is shown in Figure 4.4.

```

REF: (1.2,4.0) I WANT TO FLY FROM SPOKANE TO SEATTLE
REF: (5.3,8.6) I WOULD LIKE TO FLY FROM SEATTLE TO SAN FRANCISCO
HYP: (1.21,4) <BR> I WANNA FLY FROM SPOKANE TO SEATTLE
HYP: (5.36,8.61) <LG> I WOULD LIKE TO FLY FROM SEATTLE TO SAN FRANCISCO
HYP: (1.21,4) <BR> AY WAANTD TUW FLAY FRAHM SPOW KAEN TUW SIY AE DXAXL
HYP: (5.36,8.61) <LG> AY WUHDD LAYKD TUW FLAY FRAHM SIY AE DXAXL TUW SAEN FRAEN SIH
SKOW

```

**Figure 4.4: Word Reference File and Two Output Files**

Notice the timing data the tag (REF or HYP) prepended to each utterance. Scoring is a matter of aligning the reference and hypothesis for each utterance and computing the word error rate as described in Section 1.07 and Jurafsky and Martin (2009). In Experiment 2.1 the syllable output (lines 5 and 6 in Figure 4.4) undergo the additional probabilistic transformation to words. In Experiment 2.2, we syllabify the reference files before the scoring begins. So, lines 1 and 2 in Figure 4.4 are as shown in Figure 4.5.

```

REF: (1.2,4.0) AY WAANTD TUW FLAY FRAHM SPOW KAEN TUW SIY AE DXAXL
REF: (5.3,8.6) AY WUHDD LAYKD TUW FLAY FRAHM SIY AE DXAXL TUW SAEN FRAEN SIH SKOW

```

**Figure 4.5: Syllable Reference File**

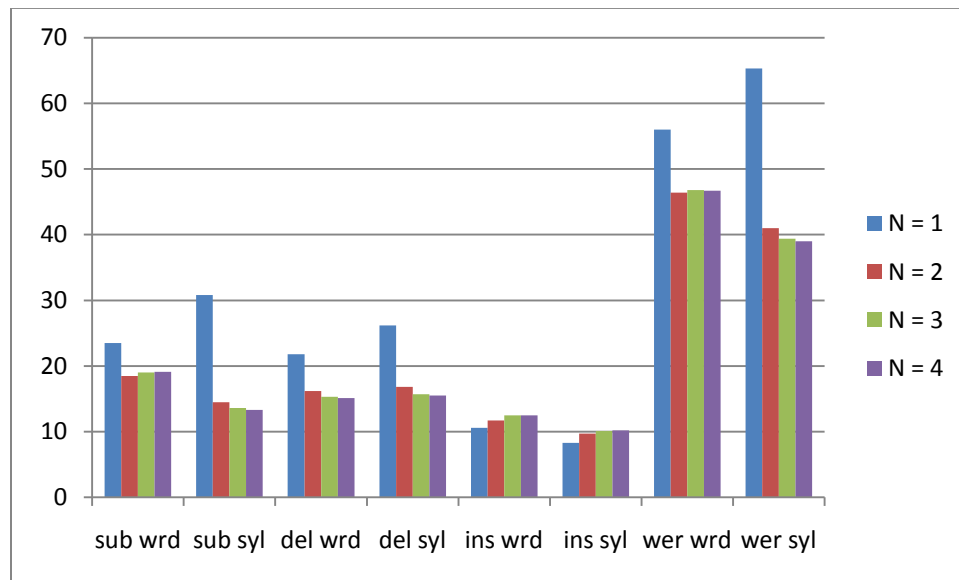
Having eliminated the probabilistic transformation of syllable output to word output, the syllable language model performs significantly better than the word language model, as shown in the next subsection.

#### 4.5.1 Results from Experiment 2.2

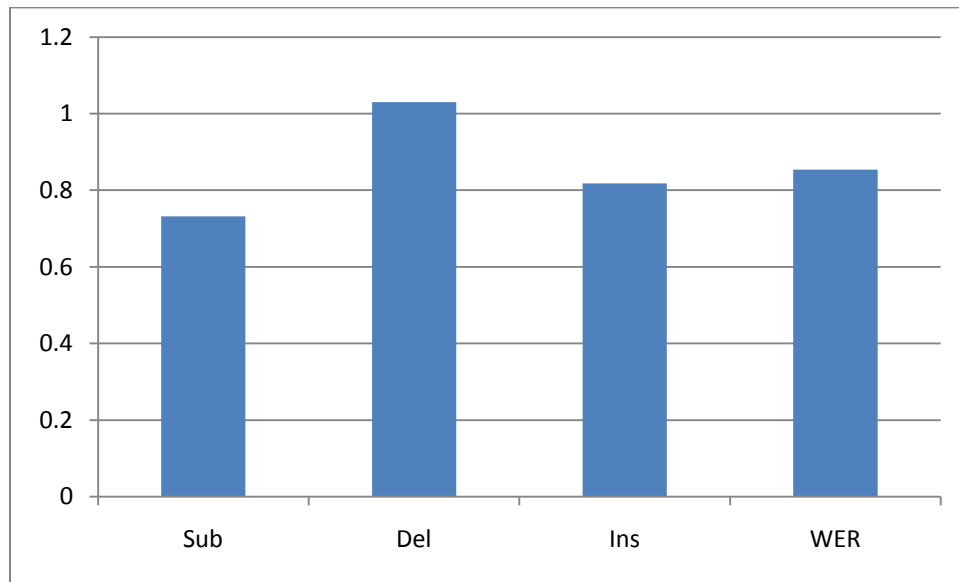
The results from Experiment 2.2 are shown in Table 4.3 and 4.4 and Figures 4.6, and 4.7. The raw data can be found in Appendix F.

**Table 4.3: Mean Errors for Syllable Language Models by N-gram Size**  
(Syllable Reference Files)

	Sub	Del	Ins	Mean	Median	STD	Best	Worst
N = 1	30.8	26.2	8.3	65.3	62.4	21.5	28.0	104.9
N = 2	14.5	16.8	9.7	41.0	38.8	22.8	8.3	87.8
N = 3	13.6	15.7	10.1	39.4	38.1	23.0	8.3	84.1
N = 4	13.3	15.5	10.2	39.0	37.1	23.6	0.0	86.6



**Figure 4.6: Mean Errors as a Function of N-gram Size,  $N = 1, 2, 3, 4$**   
In order from left to right within error type



**Figure 4.7: Mean Ratio of Syllables to Words by Error Type**  
**Across N-gram Sizes 2, 3,4**

#### 4.5.2 Discussion of Results for Experiment 2.2

As expected the syllable language model performs considerably better than the word language model when the scoring for syllables is done using a syllable reference file. We can get a sense of just how good by comparing Figure 4.3 to Figure 4.7. Syllables outperform words in every category except deletions, where the two language models are near parity. Table 4.4 shows the mean improvement by category over the three realistic N-Grams sizes (2, 3, 4).

**Table 4.4: Mean Improvement By Category Across N-gram Sizes 2, 3, 4**

	Sub	Ins	Del	SER
%	36.9	- 2.9	22.31	17.32

This data suggests that the hypothesis with which we began has been confirmed: An LVCSR equipped with a syllable language model will perform better than one equipped with a word language model.

The only thing that remains is to try to explain why the syllable language model performs less well than the word language model when  $N = 1$ . The reason is really quite simple and has to do with the idiosyncratic nature of the training transcript. It contains more syllable tokens than word tokens (8587 vs. 5782). This is expected. Though most common words are monosyllabic, not all of them are. In all but the most specialized transcripts, we would expect to find more syllable tokens than words. On the other hand, the training transcript contains more syllable *types* than word types<sup>57</sup>. In other words, the sample space is larger and, in the case of unigrams, the language model offers no historical data to the recognizer to guide its hypotheses. All other things being equal, the recognizer performs less well when offered a larger sample space and no historical data.

#### 4.6 Stress Markings and the NIST Syllabifier, Experiment 2.3

It is well-known that stress plays a role in human syllabification (see Section 2). What was less clear at the onset of this study is the role that stress plays in the NIST syllabifier. The short documentation that comes with the syllabifier is quite informal but does offer guidance on how to insert stress marks into the transcribed words that form its input. The syllabifier, for example, accepts either of the following as input to the word, *adult*:

---

<sup>57</sup> Whether this is an idiosyncratic feature of this transcript or would be characteristic of any random sample of speech is, of course, an empirical question. Odds are that it is idiosyncratic. The ratio of word syllable types to word types in the Appendix C corpus is 1.114, whereas the ratio in the much larger (and presumably more representative) ATIS corpus is .819. See subsection 3.1.0.



ax d ah l t

'0 ax d '1 ah l t

“\0” or “\1” indicates level of lexical stress, the higher the number the more stress. The syllabifier produces different output for each input string, as shown in Figures 4.8 and 4.9.

```
Enter ASCII phone string:
Basic pron is /# [ '0 ax ] [ d '0 ah l t ] #/
No. of prons = 3
They are:
# Pronunciation ..... Rate Lects
1 /# [ '0 ax ] [ d '0 ah l t ] # / <3 0
2 /# [ '0 ax d ] [ '0 ah l t ] # / >2 0
3 /# [ '0 ax [ d ] '0 ah l t ] # / >3 0
```

**Figure 4.8, Sample Output for Input Without Lexical Stress**

```
Enter ASCII phone string:
Basic pron is /# [ '0 ax ] [ d '1 ah l t ] #/
No. of prons = 1
They are:
# Pronunciation ..... Rate Lects
1 /# [ '0 ax ] [ d '1 ah l t ] # / >0 0
```

**Figure 4.9, Sample Output for Input With Lexical Stress**

The question is whether a syllable language model whose syllables were produced with the help of lexical stress would perform better than one without stress markings. That is, the question being asked is if a syllabification were undertaken with additional information, namely stress markings, would it perform better than one without that

information? The experiment used to answer that question is described in the following sections.

#### 4.6.1 Materials

The materials used in this experiment were the NIST syllabifier, available from the National Institute of Standards (NIST, 2007), an on-line pronunciation dictionary, available from Carnegie-Mellon University (CMU, 2009), and the same training transcript used to produce the syllable language model described above. See Appendix C for the full text of the transcript along with details of its capture.

#### 4.6.2 Technique

We hand-extracted every multisyllabic word found in the training transcript used in this study. For each word in the transcript, we looked up the canonical pronunciation in the CMU pronouncing dictionary. The dictionary provides lexical stress markings. We then constructed two input files, one with the stress markings included, one without the stress markings. We used each of these files as input to the NIST syllabifier and compared the outputs.

#### 4.6.3 Results

There are 482 word types in the training transcript, of which 44 are polysyllabic. Of these, all but three were found in the CMU pronouncing dictionary. These three words (*GOIN'*, *WASHTUH*, *WASHTUCK*).<sup>58</sup> The transcription software handled all three words. We encoded lexical stress based on residency in Washington State and status as a speaker of Standard American English. The syllabifier produced a different set of syllabifications on 41

---

<sup>58</sup> *WASHTUCK* refers to a town in Washington State. *WASHTUH* is a disfluency included in the transcript that directly precedes *WASHTUCK*.

of the 44 polysyllabic words. The number of candidate syllabifications varied from 1 to 3. In each of the 41 different syllabification sets, the number of candidate syllabifications for the non-stress-marked words exceeded those for the stress-marked words. Nevertheless, the NIST syllabifier offers a best-choice syllabification, what it calls “The basic pron.” In every case, the best-choice syllabification was identical in the stress-marked and non-stress-marked file.

#### **4.6.4 Discussion**

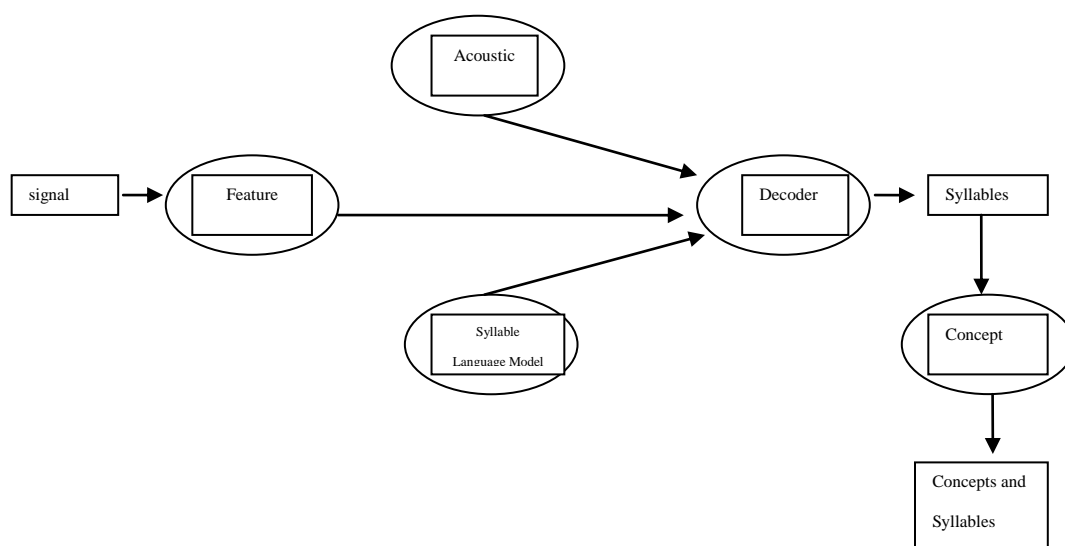
Not surprisingly, the inclusion of lexical stress constrained the output of the syllabifier. What was not expected was that the recommended pronunciation is exactly the same in the stress and no stress cases. The implication for the current study is that the inclusion of lexical stress in the input to the syllabifier will not affect the performance of the resulting language model. The reason for this has to do with the way the syllabified language model is constructed. The software written for this study extracts the single recommended syllabification from the output of the syllabifier to construct the language model. Since the recommendation is the same, whether stress markings are included or not, the resulting language model would be identical. There is no need to build one using the stress-marked candidates.

The result, however, points to an investigation that could be conducted in the future. Instead of trying to make the syllabifier produce better (recall that “better” is an empirical question) syllabifications, we could instead use the larger set of syllabifications produced by the non-stressed input, under the assumption that they may preserve non-standard pronunciations and, so, syllabifications. For example, the non-stressed output for [ax d ah l t ], produces a syllabification in which [d] is ambisyllabic, presumably to account for dialects

that stress the initial vowel: [ax [ d ] ah l t ]. So, considering multiple syllabifications, instead of the single best syllabification, would lead to a richer set of N-grams and, possibly, a better performing recognizer.

## 5.0 Experiment 3: A Concept Component

We now arrive at the third experiment. We will test the hypothesis that syllables, when passed through a concept component, will deliver improvements over those demonstrated in Experiment 2. A follow-up study will examine the effects of sending the output of this component off to an expert system. The expert system will generate a response. See Section 6 for a discussion of future research. We limit ourselves here to a proof of concept, namely that a recognizer can profitably move from an acoustic signal to a string of concepts, bypassing words altogether. Figure 5.1 shows the complete system. Experiment 3 will rate the output of the concept component, comparing it to that of a recognizer equipped with a syllable language model.



**Figure 5.1: The Complete System**

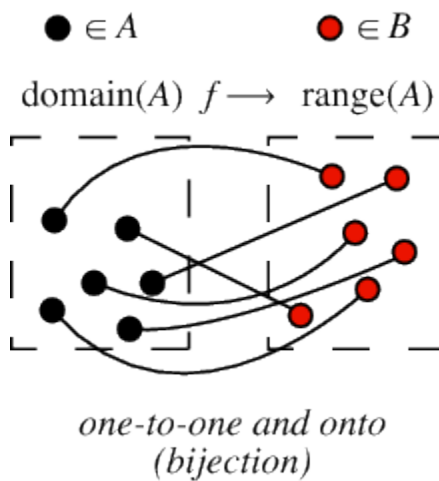
The question of what is intended by the word *concept* naturally arises. Hacıoglu and Ward (2002, p. 2) take an operational view when they define concepts as “classes or phrases with the same meaning” (p. I-226). The crucial phrase here is “same meaning,” something

they never define. This lack of specificity is perplexing, since the term has a long history in cognitive science (see Rosch, 1978; Weiner and De Palma, 1993). In fact, this looseness opens the door to the kind of personal intuition associated with formalist linguistics.

Hacioglu and Ward have mapped concepts of their own invention to “respective values” (p. I-226) outside of any experimental setting. The difference between this use of personal intuition and that employed in formalist linguistics is that the current use is purely operational. It makes no claim about cognition. Speech recognition systems are engineering artifacts. The sole criterion is how well the artifact transforms the required inputs into the desired outputs under specified constraints. So, a Boeing 737 transforms jet fuel into the movement of an aluminum capsule at subsonic speeds. Though it is constrained by the laws of physics, it is under no obligation to be covered in feathers and flap its wings. So also with Hacioglu and Ward’s (and our own) use of the term *concept*.

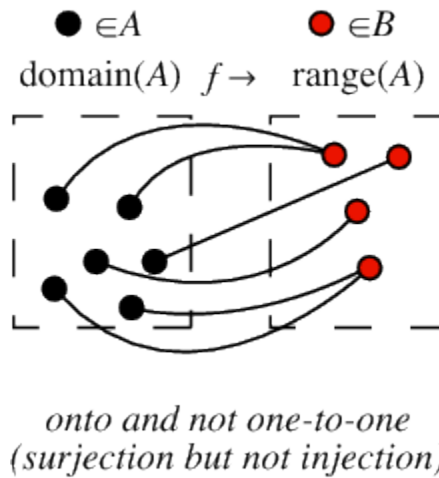
Consider another example, this one closer to home. Collecting words and phrases “of the same meaning” into a higher order classification scheme is very similar to calling NIST syllabifier-generated segments *syllables*. Though the syllabifier is based on supposed rules of syllabification that apply cross-linguistically, our only concern is whether the classification scheme produces a better result—*better* defined in the usual ways—than an LVCSR with a word language model. We make no claim about the psycholinguistic relationship of strings of sounds to mental or linguistic constructs, whether syllables, words, or concepts in the technical sense. The single criterion is whether the construction works, that is, does it produce a more robust speech recognition system, where *robust* is defined by the task at hand, i.e., SER adapted to syllables in Experiment 2 or adapted to concepts in Experiment 3?

There is one important sense in which segmenting words into syllables is different than collecting words and phrases “of the same meaning” into concepts. Mapping from words to their constituent syllables is bijective. That is, it is an invertible function. Recall that a function, or map, is bijective if it is both one-to-one and onto. A function,  $f$ , has an inverse,  $f^{-1}$ , if and only if it is bijective (Barile, 2009). The transformation shown in Figure 5.2 is bijective (from Weisstein, 2009c).



**Figure 5.2: Bijection**

Give me a word and I’ll give you a sequence of syllables. Give me a sequence of syllables and I’ll give you a word. On the other hand, mapping from words and phrases “of the same meaning” to concepts is a surjective, or an onto function. Though it maps each element from a domain (the words) onto distinct elements from a range (the concepts), there is no inverse. Give me a word or phrase and I’ll give you a concept. But give me a concept, and I won’t be able to tell you the unique word or phrase that produced it. Figure 5.3 shows a surjective function (from Weisstein 2009d).



**Figure 5.3: Surjection**

Another way of saying this is that information is lost in the process of mapping from words and phrases to concepts. To take a simple example, *I would like*, *I want*, *Give me* could all be reduced to the concept *QUERY*, following Pieraccini et al. (1991). Thus, whether a speaker conformed to the discourse rules governing conversations in restaurants in Standard American English by using *I would like*, or ignored them by using *I want* is not recoverable. The only criterion is whether collapsing three phrases into a single concept helps or hurts the performance of the system. In short, will the waiter bring the pizza? We could offer more complex examples, but all reduce to the same observation, namely that in *certain simplified contexts*, much of the complexity of natural language is redundant.

One such context is a call center. The invented dialog that follows shows how such a system might work.

**System:** How can I help you?

**Caller:** I'd like to fly to Spokane on November first, returning a week later.

**System:** You want to go to Spokane?



**Caller:** Yes, on November first.

**System:** I'm sorry. Would you give me the exact date of your return?

**Caller:** I want to return on November eighth.

Notice that the system does not have to render the entire sentence into text, nor does it have to extract the meaning (whatever that means) of the sentence. The words *fly*, *Spokane*, *November*, *first*, and *eighth* are all that are necessary for the system to proceed. *Fly to* could be part of a larger structure that we might call *GO*. It could include, say, *book a ticket to*, *travel to*, and so on. By not requiring a full-text rendition, the problem of speech recognition has become simpler. There are many ways to perform this kind of classification scheme. Of course, relationships among the concepts are still necessary to extract intent. The larger structure, *GO*, along with the words *Spokane*, *November*, *first* would have to trigger, for example, a display of available flights. See Section 6.0, Conclusions and Further Research.

## 5.2 The Concept Model

Any intelligent system, whether human or machine, must limit the space through which it searches to determine the intent of its (perhaps virtual) language partner. A simple personal example should make this clear. Not long ago, I tried to translate the text of the legal document describing my grandmother's dowry from Italian to Standard American English. The difficulty I encountered was not with Italian, which I read moderately well. The difficulty was with the script in which the document was written. Because I am not fluent, it was difficult for me to use contextual cues to hypothesize the Italian version of many words whose representation in script was unclear to me. After some effort, I engaged a native speaker to help me with the project. Though the script was also difficult for her to read, there were many fewer possibilities--given her native command of the language--for

each indeterminate word. In effect, Signora Barbanti was working with a reduced search space.

The approach taken in the current study is exactly the same, namely to limit the recognizer's search space by making the recognizer's target larger. We did this in Experiment 2 by asking it to recognize a comparatively small number of syllables instead of a very large number of words. Experiment 3 pushes that technique one step further. Here we go beyond the idea that words that mean the same thing can be grouped into equivalence classes, to the observation that phrases themselves, whatever their usefulness in the full range of English speech are redundant in a travel reservation system. Thus the words and phrases *ticket, to book a flight, to book a ticket, to book some travel, to buy a ticket, to buy an airline ticket, to depart, to fly, to get to*, all taken from the reference files for the audio used in this study, all describe what someone wants in this constrained context. With respect to a single word, we collapse morphology and auxiliary words, used to denote person, tense, aspect, and mood, into a base word. So, *fly, flying, going to fly, flew, go to, travelling to*, are grouped, along with certain formulaic phrases (*book a ticket to*), in the equivalence class, *GO*.

The justification for this kind of *a priori* reclassification is quite simple. Whatever processes selected primates with a capacity for language is probably not recoverable (Croft 2000). Nevertheless, it appears uncontroversial to assert that human language is used in a variety of contexts--to give instructions, to recount events, to communicate inner states, and so on. Much of this usage, we must assume, is conventional. When ordering in an American restaurant, we conventionally use the subjunctive (*I would like*), whereas Italians more frequently use the indicative (*Prendo, [I take]*). Further, it seems reasonable to assume that the conventions themselves depend on our having a conversational partner, the waiter in this

case, who, as a member of our speech community, expects a certain, i.e., formulaic, sequence of sounds. But what if the context is severely constrained and our partner is not a member of any speech community, but is, rather, a machine? Expecting a recognizer to transcribe the full range of English speech in the grammatically/semantically/pragmatically reduced context of making airline reservations, is a bit like leasing the space shuttle to travel between Philadelphia and New York. The distinctly 19<sup>th</sup> century train will get you there very quickly with the chance of mishap greatly reduced. We could constrain the speaker, the approach apparently taken by Pellom, et al. (2001), or we could do it ourselves by providing an appropriate set of mappings between words/phrases and concepts.

Table 5.1 shows how a set of commonly used words and phrases can be reduced to the WANT equivalence class. The words and phrases were hand-extracted from the same training transcript used in Experiment 2 (see Appendix C). Three other similarly constructed equivalence classes, BE, GO, RETURN can be found in Appendix E.

**Table 5.1: *WANT Equivalence Class***

<b>WANT</b>	
<b>Word Map</b>	<b>Syllable Map</b>
Buy	b_ay
can i	k_ae_n ay
can I have	k_ae_n ay hh_ae_v
could I	g_eh_td
could I get	k_uh_dd ay g_eh_td

Table 5.1 Continued

<b>WANT</b>	
<b>Word Map</b>	<b>Syllable Map</b>
I like	ay l_ay_kd
I need	ay n_iy_dd
I need	ay n_iy_dd
I wanna	ay w_aa n_ax
I want	ay w_aa_n_td
I would like	ay w_uh_dd l_ay_kd
I'd like	ay_dd l_ay_kd
I'd like to have	ay_dd l_ay_kd t_uw hh_ae_v
I'm planning on	ay_dd l_ay_kd t_uw hh_ae_v
looking for	l_uh k_ix_ng f_ao_r
Need	n_iy_dd
Wanna	w_aa n_ax
Want	w_aa n_td
We need	w_iy n_iy_dd
We want	w_iy w_aa_n_td
We would like	w_iy w_uh_dd l_ay_kd
We'd like	w_iy_dd l_ay_kd
We'll need	w_iy_l n_iy_dd
Would like	w_uh_dd l_ay_kd

Using equivalence classes, the sentence from the transcript:

I WANT TO FLY TO SPOKANE (1)

becomes:

WANT GO SPOKANE (2)

Actually, this is somewhat of a simplification. Experiment 2 uses a syllabified transcript, so utterance (1) is really:

ay w\_aa\_n\_td t\_uw f\_l\_ay t\_uw s\_p\_ow k\_ae\_n (3)

With concepts inserted, this becomes:

WANT GO s\_p\_ow k\_ae\_n (4)

### 5.3 Experiment 3.1

Experiment 3.1 is intended as a proof of concept. Will the reduction of word strings to equivalence classes provide a SER sufficient to justify building a probabilistic concept language model and other interfaces necessary to send the output of a concept component on to an expert system? The experiment has four steps:

1. Develop a set of equivalence classes from the training transcript used in Experiments 1 and 2. These are found in Appendix E.
2. Map the equivalence classes onto the reference files used to score the output of the syllabifier. That is, for each distinct syllable string that appears in one of the four equivalence classes, substitute the name of the equivalence class for the syllable string. Do this for each of the 18 reference files. Thus, for example, WANT is substituted for every occurrence of *ay w\_uh\_dd l\_ay\_kd* (*I would like*) .

3. Using the technique of step 2, map the equivalence classes onto the output of the recognizer when using a syllable language model for N-gram sizes 1 through 4. That is, map the equivalence class names on each of the 72 output files (4 X 18) generated by the recognizer.
4. Determine the SER of the output in step 3 with respect to the reference files in step 2,

### 5.3.1 Materials

Materials required for this experiment consist of:

1. Software developed for this project as well scoring software available from the National Institute of Standards (Sclite, 2009)

Name	Function	Source
Sclite	Computes word error rates	Sclite
MakeConceptHypFiles.py	Inserts concepts into reference	Appendix
MakeConceptRefFiles.py	files and hypothesis files	I

2. Reference files for eighteen human-computer monologues. See Appendix D for a description of how the recordings were made.
3. Reference files with mapped equivalence classes.
4. Seventy-two output files generated by using a syllable language model and each of four N-gram sizes.
5. Output files from step 3 with inserted equivalence classes.

### 5.3.2 Results from Experiment 3.1

The summary results for Experiment 3.1 are shown in Table 5.2. The full results can be found in Appendix G.

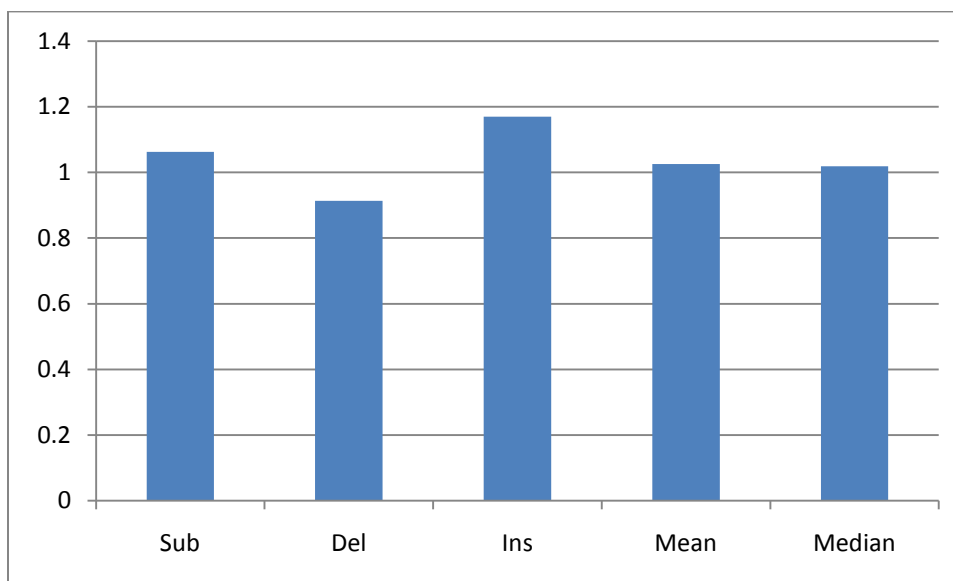
**Table 5.2: Errors for Concept Language Model**

	Sub	Del	Ins	Mean	Median	STD	Best	Worst
N = 1	32.8	22.3	12.0	67.1	63.7	21.6	25.0	105.6
N = 2	14.7	15.6	11.0	41.3	38.6	22.8	10.0	93.1
N = 3	14.8	15.0	10.8	40.6	39.2	22.3	0.0	87.5
N = 4	14.4	14.9	11.0	40.3	38.1	23.2	0.0	91.7

For convenience, the results from Experiment 2, where the recognizer was equipped with a syllable language model, are shown in Table 5.3. Figure 5.4 shows the mean ratio of errors generated by the recognizer using a syllable language model and the concept component to those generated by the recognizer with the syllable language model alone.

**Table 5.3: Mean Errors Experiment 2, Syllable Language Model**

	Sub	Del	Ins	Mean	Median	STD	Best	Worst
N = 1	30.8	26.2	8.3	65.3	62.4	21.5	28.0	104.9
N = 2	14.5	16.8	9.7	41.0	38.8	22.8	8.3	87.8
N = 3	13.6	15.7	10.1	39.4	38.1	23.0	8.3	84.1
N = 4	13.3	15.5	10.2	39.0	37.1	23.6	0.0	86.6



**Figure 5.4: Ratio of Concept Runs to Syllable Runs by Error Type**

### 5.3.2 Discussion

One of the first things to notice in Table 5.2 are the large standard deviations, averaging almost 50% of the mean SER across all four N-gram sizes, suggesting that the data is quite dispersed, though, of course, it could well be normally distributed. This is in fact the case. If we eliminate the three worst-scoring and three best-scoring speakers, the standard deviation drops by half. Contributing to the large standard deviation is the striking difference between the most and least successfully recognized speakers. Male 9 is recognized perfectly in runs with N-gram size of 3 and 4, while Male 4 is well over twice the mean in both cases. The size of the standard deviation may be a simple artifact of the way the utterances were collected. There was no attempt to weight the utterances by length, for example. Thus Male 7, consisting of a single utterance, contributes the same weight to the mean as does Male 8 with seven utterances. Though the addition of a concept language model performs slightly less well than the syllable language model alone, there are two



caveats. The first is that the concept model indicated in Table 5.1 and available in its entirety in Appendix E, is itself an experiment. That is, the only criterion for choosing the concept model shown in Appendix E is empirical. We began with the concept model found in Appendix E, modeled very loosely on Ward (2009). We attempted to gather words into equivalence classes that convey the roughly the same information, as explained above. The equivalence classes will undergo iterative refinements, with the sole criterion being does iteration B perform better than iteration A. No cognitive claim is made.

For example, the concept model was constructed from the same training transcript used in Experiment 2. In that transcript there were many instances of the BE equivalence class shown in Table 5.4. In the reference files for the recorded input, there is not a single instance of a member of the BE equivalence class. The reason for this may have to do with how the training transcript and the recordings were constructed. The training transcript, as noted, was built from transcribed conversations of hired temporary workers who were told they were conversing with a computer. The transcript is one end of a dialog. The recordings were made by volunteer software engineers who were solicited for help by e-mail. They are monologues. The recordings have more of the feel of human-machine speech. There are imperatives (*List all flights*), fewer subjunctive requests (*I would like*), and no references to companions (*Me and my husband want to go...*). In short, the match between the training transcript and the reference files is not a good one. This means in practice that the strength of the concept model, its relatively larger target—several syllables can be collapsed to a single equivalence class—remains underexploited.

Second, the choice of equivalence class members was made to maximize the target size. In many cases, entire phrases are mapped to a single equivalence class. Thus, *to buy an*

*airline ticket*, becomes GO. But the rules of the mapping, chosen in order to generate the worst-case results, required that the entire phrase had to appear in the output file.

Misrecognition of a single syllable, say *air*, would cause the equivalence class not to be inserted, and would result in errors for each missed syllable. Given the mismatch between the training transcript and the recordings and this rather rigid mapping scheme, an increased mean SER of just 1.175% seems remarkable. I conclude that with a larger or more closely matched training transcript along with probabilistic mapping, the addition of a concept language would result in a more robust LVCSR than the syllable language model alone.

**Table 5.4: The BE component of the Concept Model**

<b>BE</b>	
<b>Word Map</b>	<b>Syllable Map</b>
Be	b_iy
goin' to be	g_ow ax_n b_iy
Going to be	g_ow ix_ng t_uw b_iy
I am	ay ae_m
I have	ay hh_ae_v
I'll be	ay_l b_iy
I'll have	ay_l hh_ae_v
I'm	ay_m
I'm going	ay_m g_ow ix_ng
I'll	ay_l
Is	ih_z

Table 5.4 Continued

<b>BE</b>	
<b>Word Map</b>	<b>Syllable Map</b>
It's	ih_ts
It's goin' be like	ih_ts g_ow ax_n b_iy
it'll be	ih dx_ax_l b_iy
There are	dh_eh_r aa_r
There going to be	dh_eh_r g_ow ix_ng t_uw b_iy
There should be	dh_eh_r sh_uh_dd b_iy
There will be	dh_eh_r w_ih_l b_iy
There'll be	dh_eh r_ax_l b_iy
There's	dh_eh_r_z
there's	dh_eh_r_z
there's going to be	dh_eh_r_z g_ow ix_ng t_uw b_iy
they'll	dh_ey_l b_iy
We have	w_iy hh_ae_v
We will be	w_iy w_ih_l b_iy
We'll be	w_iy_l b_iy
we'll	w_iy_l
Will be	w_ih_l b_iy
Will have	w_ih_l hh_ae_v

## 5.4 Experiment 3.2

Researchers in LVCSR systems have long-recognized that different LVCSR systems perform differently on the same data. For instance, in 1997 the National Institute of Standards conducted benchmark tests for several speech recognition systems. The two systems that performed best, one from Raytheon's BBN Technologies and one from Carnegie-Mellon, posted respective word error rates of 44.9% and 45.1%. Given this .2% difference in WER, it is perhaps surprising that of 5919 segments with errors, only BBN posted errors in 738 of them, while only CMU-ISL posted errors in 755 segments. This suggested to NIST researchers that it might be useful to develop a post-processing system that would align the outputs of different recognizers and vote, according to various criteria, on the correct segment. The resulting system, known as ROVER (Recognizer Output Voting Error Reduction) applied to the output of four LVCSR systems resulted in an 11.8% relative improvement of the best-performing system. That is, the BBN system went from a WER of 44.9% without ROVER to a WER of 39.7% with ROVER (Fiscus, 1997).

Could ROVER be profitably used within a single system, specifically within SONIC? As it happens, SONIC provides the ability to output the not just the best hypothesis per utterance, but the N-Best. Before we begin, it seems important to point out that, though *N-Best* is the conventional usage, and the manner of generating the N-Best hypotheses is not mentioned in the SONIC manual (Pellom and Hacıoglu, 2005), the output of the recognizer is almost certainly an approximation. Multiple hypotheses can be visualized as a (not necessarily complete) graph, with a collection of vertices representing words, and a collection of weighted arcs representing the probabilities of transitioning from one word to another. This graph is known as a lattice in the speech recognition literature. The N-best

hypotheses are just  $N$  paths through the lattice. One might even go a step further and conjecture that the problem of finding the most probable path through the lattice (and, therefore, the  $N$  most probable paths) is NP-complete, given its informal similarity to the Travelling Salesperson Problem.<sup>59</sup> At least one research group has shown that the problem of finding the *most* probable parse using a stochastic regular grammar is NP-hard (Casacuberta and Higuera, 2000).<sup>60</sup> Knight (1999) argues that the simplest forms of statistical models for machine translation are NP-complete. Tillmann (2006) observes that the same dynamic programming decoding and alignment algorithms used in machine translation are also used in speech recognition. Though the details are difficult, the observation is not surprising. In both cases, the problem is one of transforming a signal from one type (acoustic in the case of speech recognition, target language in the case of machine translation) to a signal of another type (text and source language respectively). The significance of this discussion for our problem is that hypotheses 2 through  $N$  might contain information overlooked in hypothesis 1, simply because generating the actual most probable hypothesis is computationally intractable.

Figure 5.5 shows the N-Best hypotheses for a single utterance (from audio file labeled *Male1*, which contained a total of five utterances. See Appendix D), followed by the

---

<sup>59</sup> The designation *NP-complete*, informally means that a given problem is just as hard as a very large group of similar problems that are known to be intractable. The execution time for solutions to these problems grow unreasonably fast with growth of the input data. See Garey and Johnson (1979) for the classic discussion.

<sup>60</sup> A regular grammar is a rule-based formalism that is equivalent to the finite state automata described in Appendix B as well as to regular expressions. One use of regular grammars/expressions is to formalize text searches. Another is in morphological parsing. Regular grammars are a restricted form of the familiar context free grammars used to describe the syntax of both programming and natural languages. Regular grammars have rules of the form, for example,  $S \rightarrow 0S$  and  $S \rightarrow 0$ , a grammar capable of generating strings of one or more zeroes. Stochastic regular grammars are just regular grammars with probabilities associated with each rule. Informally, NP-hard problems belong to a set of problems whose members are as hard to solve as NP-complete problems (Jurafsky and Martin 2009; Garey and Johnson 1979; Hopcroft, Motwani, and Ullman 2001).

syllabified reference for that utterance. At the very bottom of the figure is the word version of the reference. The numbers in parentheses are log probabilities. These are followed by the recognizer's confidence rating.

ay w_aa_n_td ax t_r_ae v_ax_l f_r_ah_m s_p_ow k_ae_n s_iy ae dx_ax_l	(-4867.515137)	20
ay w_aa_n_ax t_r_ae v_ax_l f_r_ah_m s_p_ow k_ae_n s_iy ae dx_ax_l	(-4870.780762)	13
ay w_aa_n_td t_r_ae v_ax_l f_r_ah_m s_p_ow k_ae_n s_iy ae dx_ax_l	(-4893.363281)	4
n_ow ay w_aa_n_td ax t_r_ae v_ax_l f_r_ah_m s_p_ow k_ae_n s_iy ae dx_ax_l	(-4906.576172)	1
ah_m ay w_aa_n_ax t_r_ae v_ax_l f_r_ah_m s_p_ow k_ae_n s_iy ae dx_ax_l	(-4916.541016)	0

**Figure 5.5: N-Best Hypotheses**

The task in Experiment 3.2 is to insert concepts, as in Experiment 3.1, and then score the hypotheses, choosing the 1-Best utterance among the candidate utterances, repeating this process for every utterance for every speaker. This required four steps:

1. Generate SONIC's N-Best hypothesis for each utterance within each of the same eighteen audio files used in the previous experiments
2. Collect and format the output of the recognizer as input to ROVER
3. Collect and format the output of ROVER as input to sclite, the National Institute of Standards scoring program.
4. Run sclite on the results

#### **5.4.1 Materials**

Materials used in this experiment consists of:

1. Software developed for this project as well as scoring software and post-processing software available from the National Institute of Standards

<b>Name</b>	<b>Function</b>	<b>Source</b>
Sclite-2	Computes word error rate	NIST
ROVER	Chooses the best among several hypotheses	NIST
MakeConceptHypFiles.py	Inserts concepts into the N-Best hypotheses of the recognizer	Appendix I
CreateNBestFileClass.py	Determines how many N-best files there are	Appendix I
MakeConceptsClass.py	Inserts concepts into the output of the recognizer	Appendix I
MakeCTMClass.py	Transforms the output of the recognizer to Concatenation Time Marked (CTM) format defined by NIST and necessary for ROVER.	Appendix I
RunRoverClass.py	Controls the execution of ROVER	Appendix I
CollectVotesClass.py	Collects the output of ROVER into a single file.	Appendix I
ScliteTst.py	Controls Sclite-2	Appendix I
nBestTst.py	Controls all software in this experiment except Sclite-2 and ScliteTst	Appendix I

2. Input and output files to the software in item 1, above:

<b>Description</b>	<b>Function</b>	<b>Source</b>
Reference files for 18 human-computer monologues	Used in scoring	Appendix D
Reference files with mapped equivalence classes	Used in scoring	Mappings found in Appendix E
72 hypothesis files obtained using a syllable language model and N-gram sizes 1 - 4	Input to software that inserts equivalence classes	Recognizer
72 hypothesis files described above, but with inserted equivalence classes	Input to ROVER and other software developed for this experiment	MakeConceptsClass.py

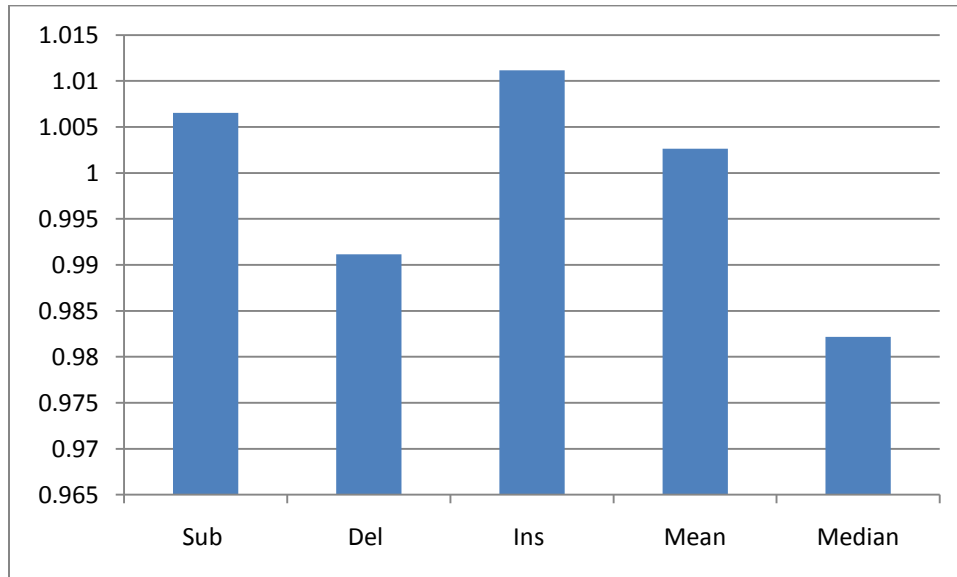
### 5.4.2 Results

The summary results for Experiment 3.2 are shown in Table 5.5. The full results can be found in Appendix H. Figure 5.6 shows the ratio of mean errors across all four N-gram sizes in the N-Best (Experiment 3.2) test to those in the 1-Best test (Experiment 3.1)



**Table 5.5: Mean Errors Experiment 3.2, Concept Language Model**  
**(ROVER Chosen Best)**

	Substituted	Deleted	Inserted	Mean	Median	STD	Best	Worst
N = 1	32.7	22.1	12	66.8	63.8	21.1	25	102.8
N = 2	15.5	15.2	10.9	41.6	37.8	23.2	5	94.4
N = 3	14.9	14.9	11.1	41	37.4	22.6	5	91.7
N = 4	14.1	15	11.3	40.4	37.4	23.1	0	91.7



**Figure 5.6: Ratio of Mean Errors Experiment 3.2 to Mean Errors Experiment 3.1**

### 5.4.3 Discussion

Though, as Figure 5.6 indicates, there is a very slight improvement in median SER from the 1-Best to the N-Best experiments, all values are insignificantly different. We conclude that post-processing of the sort that ROVER does probably will not decrease SER

and does not merit further investigation. Put another way, the differences that NIST researchers uncovered among recognizers is not present in the same recognizer among multiple hypotheses for the same utterance. The clear implication is that despite the computational complexity of traversing the lattice, SONIC does a good job of approximating the optimal hypothesis, *optimal*, of course, defined probabilistically in terms of the information stored in the recognizer.

## 6.0 Conclusions and Future Research

Experiments 2 and 3 are proofs of concept. A speech recognition system equipped with a syllable language model and a concept component will perform better in a tightly constrained domain than a conventional recognizer. Given that the hypotheses worked out as expected, there are several directions that we might take this research:

1. The syllable output of the recognizer has sequences of syllables that comprise partially formed concepts. These were passed over when we inserted concepts. By treating the syllable to concept mapping as probabilistic rather than as a surjective function, we might develop a more robust recognizer. I sketch the contours of this approach in Section 6.1.
2. In experiment 3, we inserted concepts into the best and N best paths through the lattice. We then scored each of these. What we did not do is to determine if the pieces of the target references files are contained in the lattice, but missed in output generation. A future project is to examine the entire lattice by hand in an attempt to determine if some alternative mechanism to the Viterbi algorithm could output enough information to fulfill a request to a machine travel agent.
3. The syllable/concept output of the concept component might be reformatted as input to an expert system that could generate a response. The Next IT Corporation, that sponsored much of the effort that went into Experiment 1, have such an expert system and are willing to help with such an experiment.
4. The system was trained on a relatively small corpus and asked to recognize a small number of questions. Given that the responses look promising, we could re-run

- experiments 2 and 3 (and possibly, point 2, above) on a much larger dataset, say the Air Travel Information System Spoken Language Systems Pilot Corpus of human-computer speech (Hemphill et al., 2009).
5. Casacuberta and Higuera (2000) argue that although the probability of a parse string can be computed by a stochastic grammar in linear time, the problem of finding the single string that is the most probable parse is NP-hard (Casacuberta and Higuera, 2000). Knight (1999) and Tillman (2006) argue that the general machine translation problem is NP-complete and that dynamic programming algorithms can be analyzed as shortest path algorithms respectively. No one appears to have made an explicit argument that speech recognition is reducible to a known NP-complete problem, say the Travelling Salesperson Problem where the task is, given known distances between any two cities in a set of cities, construct the shortest tour that visits each city in the set, starting and ending at the same city. This is a possible area for further investigation.
  6. Just as the recognizer can generate the N-Best hypotheses, the NIST syllabifier can generate multiple syllabifications. The software constructed for this project chooses only the best syllabification. It is possible for ROVER to choose the best among candidate syllabifications. This would allow us to determine if (potentially) different syllabifications could produce a reduced WER. So, just as we ran ROVER to choose among the N-Best hypotheses, we might invoke it earlier to choose among the N-Best syllabifications. This would allow us to determine if (potentially) different syllabifications could produce a reduced symbol error rate.

## 6.1 The Contours of a Probabilistic Concept Component

Recall that the output of a speech recognizer is the most probable sequence of words given an acoustic event, where *acoustic event* is short-hand for a speech wave form whose amplitude has been sampled and quantized (i.e., digitized) at precise intervals. In the case of Experiment 2, the output is the most likely string of syllables given an acoustic event. Following Jurafsky and Martin (2009), we can treat the acoustic event,  $O$ , as if it were a collection of sequential observations, each of the same size:

$$O = o_1, o_2, \dots, o_t \quad (6.1)$$

Likewise, the string of syllables can be represented as:

$$S = s_1, s_2, \dots, s_n \quad (6.2)$$

So, the output of the recognizer is:

$$\text{hyp}(S) = \text{argmax}_{S \in L} P(S|O) \quad (6.3)$$

where the subscript  $S \in L$  simply means that  $S$  is one of the  $L$  hypothetical strings generated by the recognizer. Equation 6.3 is read, “The hypothesized syllable string is the one with the greatest probability given the sequence of acoustic observations.” In the case of concepts, we want to output the most likely concept string given a sequence of syllables, specifically that sequence of syllables whose probability given an acoustic observation is maximized in 9.3:

$$\text{hyp}(C) = \text{argmax}_{C \in M} P(C|S) \quad (6.4)$$

where, like in 6.4,  $C \in M$  means that  $C$  is one of the  $M$  hypothetical concepts strings generated by the recognizer. We would like to transform  $P(C|S)$  into something that is more directly computable. To do so, we invoke Bayes’ Theorem:

$$p(x|y) = \frac{p(y|x)*p(x)}{p(y)} \quad (6.5)$$

$p(x)$  is commonly referred to as *the prior probability of  $x$* .  $p(y|x)$  is referred to as *the likelihood of  $y$  given  $x$* <sup>61</sup>.

Rewriting 6.4 using 6.5 gives:

$$hyp(C) = \operatorname{argmax}_{C \in M} P(C|S) = \operatorname{argmax}_{C \in M} \frac{P(S|C) * P(C)}{P(S)} \quad (6.6)$$

The first thing to notice is that the denominator on the right hand side is identical for every candidate concept string. Since we don't care about the probabilities themselves, but only the concept string with the highest probability, it can be eliminated, giving:

$$hyp(C) = \operatorname{argmax}_{C \in M} P(C|S) = \operatorname{argmax}_{C \in M} P(S|C) * P(C) \quad (6.7)$$

Equation 6.7 describes what I will call the Probabilistic Concept Component (PCC).

At this point, it is instructive to place the equations for the output of the recognizer above the equation for the output of the PCC:

$$hyp(S) = \operatorname{argmax}_{S \in L} P(S|O) \quad (6.8)$$

$$hyp(C) = \operatorname{argmax}_{C \in M} P(C|S) \quad (6.9)$$

Using Bayes' Theorem, 6.8 may be rewritten as:

$$hyp(S) = \operatorname{argmax}_{S \in L} P(S|O) = \operatorname{argmax}_{S \in L} P(O|S) * P(S) \quad (6.10)$$

Both of the terms on the rightmost side of 6.10 represent the probabilities of sequences of events. Given a sequence of syllables how probable is a sequence of acoustic events? And, how probable is a sequence of syllables? The first term on the rightmost side of the equation,  $P(O|S)$ , is the output of the acoustic model. The second term is the output of the syllable language model. The  $\operatorname{argmax}$  function, itself, is the Viterbi algorithm (See Appendix B).

---

<sup>61</sup> See Appendix A

Each term in 6.10 has an analog in the PCC represented in equation 6.7. But how do we operationalize equation 6.4?

It turns out that computing the most probable concept string given a syllable string is very similar to another problem in computational linguistics, one that has a long history, namely mapping a string of part-of-speech tags to a string of words. The problem, of course, is that some words can be classified in multiple ways. To take a simple example, whether a word is a verb or a noun (“*book* a flight” vs. “I read the *book*”), a complementizer or determiner (“I thought *that* you booked *that* flight”) often depends on context. In the world of Bayesian inference, it depends probabilistically on what word or words preceded it. Stochastic part-of-speech tagging has been explored since the mid-sixties. Stolz, et al. (1965), Bahl and Mercer (1976), and DeRose (1988) are a few early efforts, all cited by Jurafsky and Martin (2009) before presenting their own HMM implementation of part-of-speech tagging. It is important to point out that the HMM used in the part-of-speech tagging problem assumes an already tagged corpus. A likely candidate is the Brown corpus, collected at Brown University in the 1960s (Jurafsky and Martin, 2008). Similarly, the PCC task assumes an already tagged training corpus. Such a corpus, of course, does not exist. Constructing it is part of the research being proposed in this section.

As they point out, part-of-speech tagging is a sequence classification task. The observation is a sequence of words. The task is to assign to that sequence a corresponding sequence of part-of-speech tags. Because the mapping from a word to a tag is ambiguous, the assignment is probabilistic. Though the mapping from syllable strings to concepts is not ambiguous—as the designer of the concept tags, I will make sure that this is not the case—the formalism that Jurafsky and Martin present for the likelihood of word string given a tag

string is not directly applicable. Equation 6.9 is theirs, changed slightly to be consistent with the previous notation:

$$\text{hyp}(t_1^n) = \text{argmax}_{t_1^n} P(w_1^n | t_1^n) P(t_1^n) \quad (6.9)$$

This is read as “the hypothesized tag string of length  $n$  is that tag string whose product of a word string of length  $n$  given the tag string (the likelihood) and the probability of the tag string (the likelihood) is the highest.” The thing to notice is that both the word sequence and the tag sequence are the same length. This is not the case with concepts and syllables where  $\text{len}(C) \leq \text{len}(S)$ . This poses a potential problem for the simplifying assumption Jurafsky and Martin offer for Equation 6.9.

The equation is too difficult to compute since it requires intermediate computation for all strings of length  $k$ ,  $1 \leq k \leq n$ . To avoid this, Jurafsky and Martin assume that the probability of a word appearing depends only on its part-of-speech tag and is independent of the words and their tags around it. The assumption of independence allows us to transform the likelihood factor in Equation 6.9 to:

$$P(w_1^n | t_1^n) \approx \prod_{i=1}^n P(w_i | t_i) \quad (6.10)$$

If Equation 6.10 is approximate for part-of-speech tagging, it might appear even less accurate for concept-tagging. Though we might not know which tag maps to a word we do know that at least one of them does. But notice that syllables do not necessarily map to distinct concepts. Many different concepts could begin with the same syllable, AIR and AIRPLANE, for example. In fact, this is not a problem. The simplifying assumption for Figure 6.10 rewritten for concept tagging is:

$$P(S|C) \approx \prod_{i=1}^n P(s_i | c_i) \quad (6.11)$$



But notice that Equation 6.11 is computing the likelihood of a single syllable given a concept, not the other way around. That is, which of a list of candidate syllables most likely maps to a given concept. So, the simplifying assumption appears as reasonable for syllables given concepts as it does for words given part-of-speech tags.

Jurafsky and Martin also make a simplifying assumption with respect to the prior factor in Equation 6.9, namely that the probability of a tag appearing depends only on the previous tag<sup>62</sup>, not the entire tag sequence:

$$P(t_1^n) \approx \prod_{i=1}^n P(t_i | t_{i-1}) \quad (6.12)$$

This is easily pressed into service of concept tagging without the caveats that produced Equation 6.11:

$$P(C) \approx \prod_{i=1}^n P(c_i | c_{i-1}) \quad (6.13)$$

Combining equations 6.7, 6.11, and 6.13, we have a way to compute an hypothetical concept sequence, given an observed syllable sequence:

$$\text{hyp}(C) = \operatorname{argmax}_{C \in M} P(C|S) \approx \operatorname{argmax}_{C \in M} \prod_{i=1}^n P(s_i | c_i) * \prod_{i=1}^n P(c_i | c_{i-1}) \quad (6.14)$$

### 6.1.1 Computing the Prior Probability

In this section we turn our attention to operationalizing the rightmost term in Equation 6.14,  $\prod_{i=1}^n P(c_i | c_{i-1})$ , namely how to compute the prior probability of concept bigrams. Suppose our training corpus contains the concepts RETURN and FLIGHT and we want to compute the probability of the sequence RETURN FLIGHT, that is, we want to compute  $P(\text{FLIGHT}|\text{RETURN})$ . We do this by counting the number of appearances of RETURN FLIGHT in the corpus and dividing it by the number of bigrams that begin with

---

<sup>62</sup> This simplification is self-imposed. This is simply the bigram assumption for a Markov chain. We can look farther back in history and use trigrams or even quadrigrams. Jurafsky and Martin do this very thing (2008, p. 149).

RETURN. Imagine having cards with all bigrams beginning with RETURN printed on the front and placed in a hat. The probability of pulling the card reading RETURN FLIGHT from the hat is just the number of RETURN FLIGHT cards in the hat divided by the sum of the number of cards in the hat whose bigram begins with RETURN. Expressed more precisely:

$$P(c_n | c_{n-1}) = \frac{\text{num}(c_{n-1}c_n)}{\sum_c \text{num}(c_{n-1}c)} \quad (6.15)$$

Since the number of bigrams beginning with  $c_{n-1}$  is equivalent to the number all bigrams beginning with  $c_{n-1}$  and followed by anything else, Equation 9.15 can be simplified:

$$P(c_n | c_{n-1}) = \frac{\text{num}(c_{n-1}c_n)}{\text{num}(c_{n-1})} \quad (6.16)$$

### 6.1.2 Computing the Likelihood

Now we want to operationalize the concept likelihood factor in Equation 6.14,  $\prod_{i=1}^n P(s_i | c_i)$ , namely the probability that if we see a given concept string, it will be associated with a given syllable string. Suppose we want to compute the probability of the syllable [eh r] given the concept AIR. We proceed exactly as we did in Section 6.1.1. We count the instances of AIR (i.e.,  $\text{num}(c_i)$ ) in the corpus. We then count how many times it is the concept tag for [eh r] (i.e.,  $\text{num}(c_i, s_i)$ ). Dividing the latter term by the former, we get:

$$P(s_i | c_i) = \frac{\text{num}(c_i, s_i)}{\text{num}(c_i)} \quad 6.17$$

The association, of course, is probabilistic and depends upon having a corpus of syllables that has previously been tagged with concepts.

### 6.1.3 Computing the Most Likely Concept Sequence

In computing the most likely concept sequence given a syllable sequence, we are engaged in what is known as a decoding task. The most widely used algorithm for decoding tasks and HMM models is the Viterbi, an instance of a dynamic programming algorithm, a technique discovered in the sixties<sup>63</sup>. A hidden Markov model is defined by a set of states, a transition probability matrix, a sequence of observations, a sequence of observation likelihoods and start and end states. The transition probability matrix corresponds to the prior probabilities computed in Equation 6.16. The observation likelihoods correspond to the likelihood probabilities computed in equation 6.17. The Viterbi algorithm moves from the start state to the end state, computing the concept sequence whose probability is maximized given a syllable sequence. It computes partial probabilities on the way to the end, saving backward pointers to previous concepts. When the program ends, it has computed the probability of the most likely concept string given the syllable string and has retained a collection of pointers that allow the algorithm to emit that concept string.

The preceding discussion adapts a well-understood statistical technique, hidden Markov models, to an apparently new arena. In fact, the newness is more apparent than actual. HMM part-of-speech tagging hypothesizes a tag sequence given a word sequence. My adaptation hypothesizes a concept sequence given a syllable sequence. The thing to notice is that in both cases we are mapping sequences of symbols to sequences of symbols that have have a probabilistic relationship to one another. If the technique works for part-of-speech tagging—and it does—then it should work for concept mapping as well.

---

<sup>63</sup> See Appendix B for explanations of both Hidden Markov Models and the Viterbi algorithm.

## 6.2 Conclusion

Speech recognition has made progress since the basic architecture was designed at IBM in the 1970s. Given the very high error rates in large vocabulary continuous speech recognition, it is still very much an open question as to whether systems will ever achieve anything even remotely similar to what any normally-endowed human can do. Recall McAllister's experiment (cited in Greenberg, 1999). If the content of the acoustic model is matched exactly with the input, the error rate drops to 5%. This suggests that the problem in LVCSR is not with feature extraction, but instead somewhere in the acoustic and language models. Faster computers and less expensive memory will ultimately produce better acoustic models. Whether performance will rise to near-human performance or level off asymptotically is not clear. Given the history of disappointments in artificial intelligence research in general and natural language processing in particular, caution is a virtue.

Nevertheless, one problem with LVCSR systems may be that what they were designed to solve—the rendering of speech to strings of words—is too narrowly defined, particularly given the probabilistic tools that are used. Perhaps another way of saying this is that the bar is set too high for most situations. Humans, recall, don't do a particularly good job of transcribing conversations. AI systems have usually foundered when they have either been asked to do something outside of a tightly-circumscribed domain or when they have been asked to do something that is not—again, for now—expressible as an effective procedure. Notice that the speech recognition problem takes something that is quintessentially human, effective communication through spoken language, and transforms it into something that is distinctly inhuman, namely the precise rendering of an acoustic event to strings of words.

Unlike with many AI techniques, we run into difficulties in ASR not because we are trying to shoehorn a messy problem into a formal representation, but rather because we are asking more than is necessary in most cases. Scholars have understood for many years that speech is something that humans do effortlessly, while writing is an artifact. Though what internal representations we form when we converse are beyond the scope of this study, it seems clear enough that we do not store long and precise strings of text. Just as engineers have simplified the problem of syllabification, they appear to have simplified the problem of speech recognition as well, ironically making it more difficult. Without making a cognitive claim about human internal representation of spoken language, we can agree that some sort of abstract reduction takes place, thus the notion of a paraphrase. Current LVCSR systems are designed with the goal of mapping an acoustic signal to a string of words. The accuracy of the systems are then judged by how closely the string of words matches a hand transcription of the acousting signal. Though this allows researchers to compare systems across applications, the task is distinctly non-human. We humans converse quite nicely without the capacity to transcribe speech in real-time. The hypothesis explored in this study is inspired by that simple observation, namely that if we scale back what we ask our system to do, we might ultimately achieve more human-like performance, namely the ability to respond reasonably to an acoustic signal given a context. The experiments that comprise this study and the probabilistic concept model proposed in Section 6.1 are the first part of the task. Ultimately, the output of an appropriately modified LVCSR must be sent on to an expert system and a speech synthesizer for further processing. These are subjects for further research. Nevertheless, the results presented here are a necessary first step.

## References

- Alaska Airlines (2010). *Ask Jenn*, retrieved 3/3/2010 from [www.alaskaair.com/jenn](http://www.alaskaair.com/jenn).
- Bahl, L., Mercer, R. (1976). Part of speech assignment by a statistical decision algorithm. In *Proceedings of the IEEE Symposium on Information Theory*, pp. 88-89.
- Bamford, J. (2008). *The Shadow Factory: The Ultra-Secret NSA from 9/11 to the Eavesdropping on America*. NY: Random House.
- Barile, Margherita (2009). Bijective, from *MathWorld—A Wolfram Web Resource*, created by Eric W. Weisstein. <http://mathworld.wolfram.com/Bijective.html>.
- Barlow, M., Kemmer, S. (2000). *Usage-Based Models of Language*. Cambridge: CSLI Publications, distributed by Cambridge University Press.
- Benfey, C. (2008). *The New York Times*. Literary Devices, 4/20/2008. Retrieved 2/27/10 from: <http://www.nytimes.com/2008/04/20/books/review/Benfey-t.html>.
- Bhaskararao, P., Eady, S., Esling, J. (1991). Use of triphones for demisyllable-based synthesis. *Proceedings of the 1991 International Conference on Acoustics, Speech, and Signal Processing*, 517-520.
- Bolanos, D., Ward, W. Van Vuuren, S., Garrido, J. (2007a) SVM based posterior probabilities for syllable confidence annotation. Paper received from Wayne Ward, Center for Spoken Language Research, University of Colorado, 10/10/07 (personal communication).
- Bolanos, D., Ward, W. Van Vuuren, S., Garrido, J. (2007b) Syllable lattices as a basis for a children's speech reading tracker. *Proceedings of Interspeech*, 2007. Bonn: International Speech and Communication Association (unpaginated).

- Browman, C., Goldstein, L. (1990). Tiers in articulatory phonology, with some implications for casual speech. In J. Kingston and M. Beckman (Eds.), *Papers in Laboratory Phonology I: Between the Grammar and Physics of Speech*. NY: Cambridge University Press.
- Bureau of Labor Statistis (2010). *Travel Agent: Occupational Outlook Handbook*. Accessed from: <http://www.bls.gov/oco/ocos124.htm#outlook> 2/27/2010.
- Bybee, J. (2006). From Usage to Grammar: The Mind's Response to Repetition. *Language*, 82 (4), 711-33.
- Bybee, J. (2003). *Phonology and Language Use*. Cambridge: Cambridge University Press.
- Cannarozzi, G. String alignment in dynamic programming. Retrieved 8/20/09 from: <http://www.biorecipes.com/DynProgBasic/code.html>.
- Casacuberta, F., de la Higuera, C. (2000). Computational Complexity of Problems on Probabilistic Grammars and Transducers. In P. Adriaans, H. Fernau, M. van Zaanen (Eds.), *Grammatical Inference: Algorithms and Applications*. Berlin: Springer-Verlag.
- Chomsky, N. (1956). Three models for the description of language. *IRE Transactions On Information Theory*. IT-2, 113-124.
- CMU (2009). *The Carnegie Mellon Pronouncing Dictionary*. Accessed 7/22/09 from: <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>.
- Cohen, P., Johnston, M., McGee, D., Oviatt, S., Clow, J., Smith, I. (1998). The efficiency of multimodal interaction: a case study. *Proceedings of ISLIP-98*, Sydney, pp. 249-252.
- Cormen, T., Leiserson, C., Rivest, R. (1991). *Introduction to Algorithms*. Cambridge: The MIT Press.

- Cover, T., Thomas, J. (1991). *Elements of Information Theory*. Hoboken, NJ: John Wiley & Sons.
- Croft, William. (2000). *Explaining Language Change*. Harlow, England: Longman Linguistics Library.
- Croft, W., Cruse, D. (2007). *Cognitive Linguistics*. Cambridge: Cambridge University Press.
- Cutler, A., Norris, D. (1988) The role of strong syllables in segmentation for lexical access. *Journal of Experimental Psychology: Human Perception and Performance*, 14 (1), 113-121.
- Cutler, A., Dahan, D., Donselaar, W. (2007). *Prosody in the Comprehension of Spoken Language: A Literature Review*. *Language and Speech*, 40(2), 141-201.
- De Palma, P., Weiner, E. (1992). *Riddles: Accessibility, Parallelism and Knowledge Representation*. *Proceedings of CoLing-92: The Fourteenth International Conference on Computational Linguistics*, Nantes, France.
- De Mori, R., Brugnara, F. (1996) HMM models of speech recognition. In R. Cole, J. Mariani, H. Uszkoreit, A. Zaenen, V. Zue (Eds.), *Survey of the State of the Art in Human Language Technology*. National Science Foundation, European Commission. Retrieved 11/30/07 from: <http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html>.
- DeRose, S. (1988). Grammatical category optimization by statistical optimization. *Computational Linguistics*, 14, 31-39.
- Elliot, L., Stinson, M., McKee, B., Everhart, V., Francis, P. (2001). College Students' Perceptions of the C-Print Speech-to-Text Transcription System. *Journal of Deaf Studies and Deaf Education*, 6 (4), 285-298.



- Eisner, J. (2002). An interactive spreadsheet for teaching the forward-backward algorithm. In *Proceedings of the American Association for Computational Linguistics on effective Tools and Methodologies for Teaching NLP an CL*, pp. 10-18.
- Everitt, K., Harada, S., Landay, J. (2007) Disambiguating Speech Commands using Physical Context. *Proceedings of 9<sup>th</sup> International Conference on Multimodal Interfaces*, 247-254.
- Fillmore, C. (1968). The case for case. In E. Bach and R. Harms (eds.) *Universals of Linguistic Theory*. NY: Holt, Rinehart and Winston.
- Fiscus, J. (1997). A post-processing system to yield reduced word error rates: Recognizer output voting error reduction (ROVER). *Proceedings IEEE Workshop on Automatic Speech Recognition and understanding (ASRU 97)*.
- Fiscus, J., Garofolo, J., Le, A., Martin, A., Pallett, D., Przybocki, M., Sanders, G. (2004). Results of the Fall 2004 STT and MDE Evaluation. In *Proceedings of the Fall 2004 Rich Transcription Workshop (RT-04)*, November 2004.
- Franklin, A. (1991). Millikan's published and unpublished data on oil drops. *Historical studies in the physical sciences*, 11, pp. 185-201.
- Ganapathiraju, A., Hamaker, J, Picone, J, Ordowski, M., Doddington, G. (1997a) *Syllable-Based Large Vocabulary Continuous Speech Rrecognition*. Baltimore: Center for Language and Speech Processing, Johns Hopkins University. Technical report retrieved 10/6/07 from <http://www.clsp.jhu.edu/>.
- Ganapathiraju, A., Goel, V., Picone, J., Corrada, A., Doddington, G., Kirchof, K., et al. (1997b) Syllable – A Promising Recognition Unit for LVCSR. Baltimore: Center for

- Language and Speech Processing, Johns Hopkins University. Technical report  
retrieved 10/6/07 from <http://www.clsp.jhu.edu/>.
- Garey, M., Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. NY: W.H. Freeman and Co.
- Genesis (2010). Retrieved 3/4/2010 from: <http://quod.lib.umich.edu/k/kjv/browse.html>
- Goslin, J. Frauenfelder, U. A comparison of theoretical and human syllabification. *Language and Speech*, 44 (4), pp 409-436.
- Godfrey, J., Holliman, E. & McDaniel, J. (1992) SWITCHBOARD: Telephone Speech Corpus for Research and Development. In *Proceedings of ICASSP-92*, Vol. 1, 517-520.
- Goldberg, R, Riek, L. (2000) *A Practical Handbook of Speech Coders*. Boca Raton, FL: CRC Press.
- Goslin, J. and Frauenfelder, U. (2000). A comparison of theoretical and human syllabification. *Language and Speech*, Vol 44 (4), 409-436.
- Goronzy, S. (1998) *Robust Adaptation to Non-Native Accents in Automatic Speech Recognition*. Berlin: Springer.
- Greenberg, S. (1999) Speaking in shorthand—A syllable-centric perspective for understanding pronunciation variation. *Speech Communication*, 29, 159-176.
- Greenberg, S. (2001) From here to utility—Melding insight with speech technology. *Proceedings of the 7<sup>th</sup> European Conference on Speech Communication and Technology* (pp. 2485-2488). Berlin: Springer.
- Greenberg, S., Carvey, H., Hitchcock, L., Chang, S. (2002) Beyond the phoneme: A juncture-accent model of spoken language. *Proceedings of the 2<sup>nd</sup> International*

- Conference on Human Language Technology Research*. San Francisco: Morgan Kaufmann.
- Hacioglu, K., Ward, W. (2002) A Concept Graph Based Confidence Measure. *IEEE International Conference on Acoustics and Signal Processing*, (I-225- I-228).
- Hall, T. (2006) English syllabification as the interaction of markedness constraints. *Studia Linguistica*, 60 (3), 1-33.
- Hamalainen, A., Boves, L., de Veth, J., ten Bosch, L. (2007) On the utility of syllable-based acoustic models for pronunciation variation modeling. *EURASIP Journal on Audio, Speech, and Music Processing*. 46460, 1-11.
- Hemphill, C., Godfrey, J., Doddington, G. (2009). The ATIS Spoken Language Systems Pilot Corpus. Retrieved 6/17/09 from:  
[http://www ldc.upenn.edu/Catalog/readme\\_files/atis/sspcrd/corpus.html](http://www ldc.upenn.edu/Catalog/readme_files/atis/sspcrd/corpus.html)
- Hindle, Donald (1983) Discourse Organization in Speech and Writing. In M. Siegel, T. Olson (eds.), *Writing Talks*. Upper Montclair NJ: Boynton-Cook.
- Hogg, R., Tanis, E. (1988) *Probability and Statistical Inference*. NY: Macmillan.
- Holmes, J., Holmes, W. (2001). *Speech Synthesis and Recognition*. London: Taylor & Francis.
- Hopcroft, J., Motwani, R., Ullman, J. (2001). *Introduction to Automata Theory, Languages, and Computation*. Boston: Addison Wesley.
- Huang, X., Acero, A., Hsiao-Wuen, H. (2001). *Spoken Language Processing: A Guide to Theory, Algorithm, and System Development*. Upper Saddle River, NJ: Prentice Hall.
- Iliad (2010). From The Catalog of the Captains, Book II. Samuel Butler translation.  
 Retrieved 3/4/2010 from: <http://classics.mit.edu/Homer/iliad.2.ii.html>,

- Jelinek, F. (1997) *Statistical Methods for Speech Recognition*. Cambridge: The MIT Press.
- Johnson, K. (2005) Speaker normalization in speech perception. In D. Pisoni & R. Remez (eds.), *The Handbook of Speech Perception*. Oxford: Blackwell. Pages 363-389.
- Jurafsky, D., Martin, J. (2000) *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ: Prentice Hall.
- Jurafsky, D., Martin, J. (2008) *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Upper Saddle River, NJ: Prentice Hall.
- Kahn, D. (1976) *Syllable-based generalizations in English phonology*. Bloomington: Indiana University Linguistics Club.
- Keating, P., Byrd, D., Flemming, E. & Todaka, Y. (1994) Phonetic analyses of word and segment variation using the TIMIT corpus of American English. *Speech Communication* 14, 131-142.
- Kenstowicz, M. (1994). *Phonology in Generative Grammar*. Oxford: Blackwells.
- King James (2010). *Genesis* I, 1-19. Retrieved 3/4/2010 from:  
<http://quod.lib.umich.edu/k/kjv/browse.html>
- Kingsbury, P., Strassel, S., McLemore, C., MacIntyre, R. (1997). *CALLHOME American English Lexicon (PRONLEX)*. Philadelphia: Linguistic Data Consortium. Retrieved 6/4/08 from:  
<http://www ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC97L20>.

- Kiyota, Y., Kurohashi, S., Misu, T., Komatani, K., Kawahara, T., Kido, F. Dialog navigator: A Spoken Dialog Q-A System based on Large Text Knowledge Base. *Proceedings of the 41st Annual Meeting of the Association for Computational Linguistics*, 149-152.
- Koehn, P. (2010). *Statistical Machine Translation*. Cambridge: Cambridge University Press.
- Kohler, K. (1966) Is the syllable a phonological universal? *Journal of Linguistics* 2, 207-208.
- Knight, K. (1999). Decoding Complexity in Word-Replacement Translation Models. *Computational Linguistics*, 25 (1), pp. 607 – 615.
- Kuhn, T. (1970). *The Structure of Scientific Revolutions*. Chicago: The University of Chicago Press.
- Luger, G. (2005) *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. New York: Addison Wesley.
- Luger, G. (2009) *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. New York: Addison Wesley.
- Macrophone (2010). Linguistic Data Consortium, Philadelphia. Retrieved 3/9/2010 from: <http://www ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC94S21>.
- Manning, C., Schutze, H. (1999). *Foundations of Statistical Natural Language Processing*. Cambridge: MIT Press.
- Marchand, Y., Adsett, C., Damper, R. (2007). Evaluating automatic syllabification algorithms for English. *Proceedings of the 6<sup>th</sup> International Conference of the Speech Communication Association*.
- Moore, R. (1998). Speech pattern processing. In K. Ponting (ed.), *Computational Models of Speech Pattern Processing*. Berlin: Springer.

Next IT. (2008). Retrieved 4/5/08 from: <http://www.nextit.com>.

Next IT. (2010). Retrieved 2/27/10 from:

<http://www.alaskaair.com/as/www2/Help/Site/askJenn.asp>.

Newell, A. (1990). *Unified Theories of Cognition*. Cambridge: Harvard University Press.

Newell, A. and Simon, H. (1976). Computer science as empirical inquiry: symbols and search. *Communications of the ACM*, 19(3), 113-126.

NIST. (2007) Syllabification software. National Institute of Standards: NIST Spoken Language Technology Evaluation and Utility. Retrieved 11/30/07 from:

<http://www.nist.gov/speech/tools/>.

NIST-IAD. (2008). Speech Group, Current Evaluations. Retrieved 5/23/08 from:

<http://www.nist.gov/speech/>.

NIST-RT. (2008). Rich Transcription Evaluation Project. National Institute of Standards, Information Access Division. Retrieved 5/23/08 from:

<http://www.nist.gov/speech/tests/rt/>.

Ong, Walter (1984). *Orality and Literacy: The Technologizing of the Word*. London: Methuen.

Pellom, B., Hacıoglu, K. (2005). *SONIC: The University of Colorado Continuous Speech Recognizer*, Technical Report TR-CSLR-2001-01. Boulder: The University of Colorado.

Pellom, B., Ward, W., Hansen, J., Cole, R., Hacıoglu, K., Zhang, J., Yu, X., Pradhan, S. (2001). University of Colorado Dialog systems for travel and navigation. Proceedings of the first international conference on human language technology research, pp. 1-6.

- Pieraccini, R., Levin, E., Lee, C. H. (1991). Stochastic representation of conceptual structure in the ATIS task. *Proceedings of the DARPA Speech and Natural Language Workshop*, pp. 121-124.
- Pitt, M.A., Dilley, L., Johnson, K., Kiesling, S., Raymond, W., Hume, E. and Fosler-Lussier, E. (2006). *Buckeye Corpus of Conversational Speech*. Columbus, OH: Department of Psychology, The Ohio State University.
- Pointing, K. (Ed.) (1998) *Computational Models of Speech Pattern Processing*. Berlin: Springer.
- Rabiner, L., Juang, B. (1993) *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice Hall.
- Risen, J., Lichtblau, E. (2005). Bush lets US spy on callers without courts. *The New York Times*, 12/16/2005. Retrieved 2/27/2010 from:  
<http://www.nytimes.com/2005/12/16/politics/16program.html>.
- Rocca, I, Johnson, W. (1999) *A Course in Phonology*. Malden, MA: Blackwell.
- Rosch, E. H. (1978). Principles of Categorization. In E. Rosch and B. Lloyd (eds.), *Cognition and Categorization*, 27-48. Hillsdale, NJ: Lawrence Erlbaum Associates.
- Sanh-Hun, Choe. (2007) In Myanmar, fear is ever present. *The New York Times*, 10/21/2007.
- Schmidt-Nielsen, A., Crystal, T. (2000). Speaker Verification by Human Listeners: Experiments Comparing Human and Machine Performance Using NIST 1998 Speaker Evaluation Data. *Digital Signal Processing*, 10, 249-266.
- Schrumpf, C., Larson, M., Eickler, S. (2005). Syllable-based Language Models in Speech Recognition for English Spoken Document Retrieval. *Proceedings of the 7th*

*International Workshop of the EU Network of excellence DELOS on Audio-Visual Content and Information Visualization in Digital Libraries*, pp. 196-205.

Sclite. (2009). Retrieved 5/22/09 from: <http://www.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm>.

Selkirk, E. (1999). The syllable. In J. Goldsmith (ed.), *Phonological Theory: The Essential Readings*, 328-350. Malden, MA: Blackwell.

Sethy, A., Narayanan, S., Parthasarthy, S. (1998) A syllable-based approach for improved recognition of spoken names. *Advances in Phonetics: Proceedings of the International Phonetic Sciences Conferences*, pp. 109-113. Stuttgart: F. Steiner.

Sethy, A., Narayanan, S. (2003) Split-lexicon based hierarchical recognition of speech using syllable and word level acoustic units. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 1, pp. 772-775.

Shannon, C.E. (1951). Prediction and entropy of printed English. *Bell System Technical Journal*, vol. 30, pp. 50-64.

Singh, Simon (1999). *The Code Book*. NY: Doubleday.

Sladek, J., Zschorn, A., Hashemi-Sakhtsari, A. (2004). Speech-To-Text Transcription in Support of Pervasive Computing. *Proceedings of the 2002 Conference on Pervasive computing*, 25, 3-8.

SONIC (2010). SONIC: Large vocabulary continuous speech technology. Retrieved 3/8/2010 from: [http://techexplorer.cusys.edu/show\\_NCSum.cfm?NCS=258626](http://techexplorer.cusys.edu/show_NCSum.cfm?NCS=258626).

SYNC (2010). Official Ford Sync Site. Retrieved 3/9/2010 from: <http://www.fordvehicles.com/innovation/sync/>.



- Selkirk, E. (1999). The syllable. In J. Goldsmith (ed.), *Phonological Theory: The Essential Readings*, 328-350. Malden, MA: Blackwell
- Sommer, B. (1970) An Australian language without CV syllables. *International Journal of American Linguistics*, 36 (1), 57-58.
- Stewart, D. Ming, J., Smith, F. (1998) Automatic syllabification with application to continuous speech recognition. *Proceedings of the International Phonetic Sciences Conference*, 109-113.
- Stolz, W.S., Tannembaum, P.H., Carstensen, F.V. (1965). A stochastic approach to the grammatical coding of English. *Communications of the ACM*, 8(6), 399-405.
- Sudkamp, T. (2006). *Languages and Machines: An Introduction to the Theory of Computer Science*. Boston: Pearson/Addison-Wesley.
- Tanenbaum, A., Woodhull, A. (2006). *Operating Systems: Design and Implementation*. Upper Saddle River, NJ: Pearson/Prentice-Hall.
- Tillmann, C. (2006). Efficient Programming Search Algorithms for Phrase-Based SMT. *Workshop on Computationally Hard Problems and Joint Conference in Speech and Language Processing*, NYC, pp. 9-16.
- Tomasello, M. (2003). Introduction: Some Surprises for Psychologists. In M. Tomasello (ed.) *The New Psychology of Language: Cognitive and Functional Approaches to Language Structure*, Vol. 2. Lawrence Erlbaum Associates.
- Trappe, W., Washington, L. (2006). *Introduction to Cryptography with Coding Theory*. Upper Saddle River, NJ: Pearson-Prentice Hall.
- Walker, M., Litman, D., Kamm, C, Abella, A. (1997) PARADISE: A Framework for Evaluating Spoken Dialogue Agents. In *Proceedings of the 35th Annual Meeting of*

- the Association for Computational Linguistics and Eighth Conference of the European Section of the Association for Computational Linguistics, Spain, 271-280.*
- Waller, J. (2004). *Einstein's Luck: The Truth behind Some of the Greatest Scientific Discoveries*. Oxford: Oxford U. Press.
- Ward, W. (2009). Personal communication, based on Pellom (2001) and Hacıoglu and Ward (2002).
- Ward, W. (1994). Extracting information from spontaneous speech. *Proceedings of the Third International Conference on Spoken Language Processing (ICSLP)*.
- Waterman, D.A. (1986). *A Guide to Expert Systems*. Reading, MA: Addison-Wesley.
- Weiner, E. & De Palma, P. (1993). Some Pragmatic Features of Lexical Ambiguity and Simple Riddles. *Language and Communication*, 13 (3), 183-193.
- Weisstein, Eric W. (2009a). Antilogarithm, from *MathWorld—A Wolfram Web Resource*.  
<http://mathworld.wolfram.com/Antilogarithm.html>.
- Weisstein, Eric W. (2009b). Logarithm, from *MathWorld—A Wolfram Web Resource*.  
<http://mathworld.wolfram.com/Logarithm.html>.
- Weisstein, Eric W. (2009c). Bijection, from *MathWorld—A Wolfram Web Resource*.  
<http://mathworld.wolfram.com/Bijection.html>
- Weisstein, Eric W. (2009d). Surjection, from *MathWorld—A Wolfram Web Resource*.  
<http://mathworld.wolfram.com/Surjection.html>.
- Weizenbaum, J. (1976). *Computer Power and Human Reason*. San Francisco: W. H. Freeman.

Young, S. (1998) Acoustic modeling for large vocabulary continuous speech recognition. In

K. Ponting (ed.) *Computational Models of Speech Pattern Processing*. Berlin:

Springer.

Zechner, K. (1998). Automatic Construction of Frame Representations for Spontaneous

Speech in Unrestricted Domains. *Proceedings of the 17<sup>th</sup> Annual Meeting of the*

*Association for Computational Linguistics, 1448-1452.*

## Appendices

Appendix A	A Derivation of Bayes' Theorem .....	151
Appendix B	Finite State Automata, Hidden Markov Models, Dynamic Programming.....	169
Appendix C	The Training Transcript.....	184
Appendix D	Reference Files .....	209
Appendix E	Four Equivalence Classes Used in Experiment 3 .....	213
Appendix F	Results, Experiments 2.1 and 2.2 .....	217
Appendix G	Output of Sclite for Experiment 3.1 .....	229
Appendix H	Output of Sclite for Experiment 3.2 .....	233
Appendix I	Software Written for the Study.....	237
Appendix J	Word-Transcription Lexicon .....	294
Appendix K	Word-Syllabification Lexicon .....	325
Appendix L	SONIC Symbol Set (from Pellom and Hacıoglu, 2005, p. 12) .....	356

## Appendix A A Derivation of Bayes' Theorem<sup>64</sup>

### A.1 Some Preliminaries

We begin with five definitions:

- Definition A.1, Random Experiment  
A *random experiment* is one in which “the outcome cannot be predicted with certainty” (p. 1).
- Definition A.2, Sample Space  
The *sample space*,  $S$ , is the set of all possible outcomes of a collection of random experiments.
- Definition A.3, Event  
If  $S$  designates a sample space and  $A$  is a subset of  $S$ , then  $A$  is an *event*.
- Definition A.4, Relative Frequency  
Suppose we conduct a random experiment  $n$  times. If event  $A$  is a set of outcomes, we can count the number of times event  $A$  occurs, denoted  $|A|$ . Then  $(|A| / n)$  is referred to as the *relative frequency* of event  $A$  in the  $n$  random experiments.
- Definition A.5, Probability  
The relative frequency tends to stabilize as  $n$  grows in size. If we associate a number,  $p$ , with event  $A$  that is approximately equal to the value around which the relative frequency stabilizes, that number is called the *probability* of  $A$ , denoted  $P(A)$ . Said another way, “ $P(A)$  represents the fraction of times that the outcome of a random

---

<sup>64</sup> The material in this appendix is intended to make the several discussions of Bayes' Theorem clearer. A fuller discussion is available in any text on probability theory. The definitions and examples are borrowed/adapted from Hogg and Tanis (1988). Since Hogg and Tanis are the only source, citations are omitted. Page numbers are included only when material is quoted.

experiment results in the event A in a large number of trials of that experiment” (p. 4).

The following example illustrates these definitions.

Example A.1 Suppose we have a fair six-sided die that we roll repeatedly. The sample  $S = \{1, 2, 3, 4, 5, 6\}$ . Let  $A = \{1, 2\}$ , i.e, the outcome of a roll is either 1 or 2.  $P(A) = 2/6 = 1/3$ .

Can this be verified experimentally? The following table indicates the result of a computer simulation.

n	A	A /n
50	16	.32
100	34	.34
250	80	.32
500	163	.326

In fact, probability can be defined more precisely using set theory. Here are a few useful definitions from set theory.

- Definition A.6, Set  
A *set* is an unordered collection of objects.
- Definition A.7, Union  
The *union* of two sets, A and B, is the set that contains those elements that occur in either A or B. The union of two sets is denoted  $A \cup B$ .
- Definition A.8, Intersection

The intersection of two sets, A and B, is the set that contains those elements that occur in both A and B. The intersection of two sets is denoted  $A \cap B$ .

- Definition A.9, Complement  
If A is a set of elements drawn from S,  $A'$ , read *A complement*, denotes the set of all elements of S with the elements of A removed.
- Definition A.10, Subset  
If every element of set A is also an element of set B, then A is a subset of B, written  $A \subset B$
- Definition A.11, Null Set  
The null set is the set containing no elements, written  $\emptyset$ . For all sets X,  $\emptyset \subset X$
- Definition A.12, Cardinality  
The number of elements in a set is called its cardinality. The cardinality of set A is denoted  $|A|$ .

We can now define probability more precisely.

- Definition A.13  
Probability is a function that maps a number  $P(A)$  to each event A in the sample space S such that the following properties are satisfied:
    1.  $P(A) \geq 0$
    2.  $P(S) = 1$
    3. If  $A_1, A_2, A_3, \dots, A_k$  are events and  $A_i \cap A_j = \emptyset$ ,  $i \neq j$ ,  $i, j < k$ , then
 
$$P(A_1 \cup A_2 \cup \dots \cup A_k) = P(A_1) + P(A_2) + \dots + P(A_k)$$
- for each positive integer,  $k$

Less formally, item 1 tells us that probability can never be negative, since it is defined as the number of times an event occurs divided by the number of trials. Item two says that probability of the sample space is 1. Since the sample space contains all possible outcomes, one of the outcomes has to occur for every random experiment. The sum of these outcomes divided by the number of outcomes is 1. Item 3 tells us that set union is closely related to addition. Suppose we have a six-sided fair die. If  $A = \{1, 2\}$  and  $B = \{3, 4\}$ , the probability

that you will roll a 1,2,3,4 is just the probability that you will roll a 1 or 2 plus the probability that you will roll a 3 or 4, namely  $2/3$ .

Several basic theorems follow from the definitions. The proofs are quite simple and may be found in Hogg and Tanis (2008).

- Theorem A.1  
For each event A,  $P(A) = 1 - P(A')$
- Theorem A.2  
 $P(\emptyset) = 0$
- Theorem A.3  
If events A and B exist such that  $A \subset B$ , then  $P(A) \leq P(B)$
- Theorem A.4  
For each event A,  $P(A) \leq 1$
- Theorem A.5  
If A and B are any two events, then  
 $P(A \cup B) = P(A) + P(B) - P(A \cap B)$

Notice that this meets the conditions of item 3 in Definition A.6. If  $A \cap B$  is the null set, then its probability is 0.

- Theorem A.6  
If A, B, and C are any three events, then  
 $P(A \cup B \cup C) = P(A) + P(B) + P(C) - P(A \cap B) - P(A \cap C) - P(B \cap C) + P(A \cap B \cap C)$

Theorem A.6 can be generalized to any number of events.

### Example A.2

A man is meeting three friends who are flying in from cities A, B, and C. We know the following probabilities, where  $P(Q)$  is the probability that any flight from city Q is on time.



$$P(A) = .93$$

$$P(B) = .89$$

$$P(C) = .91$$

$P(A \cap B) = .87$ , i.e., the probability that the any flight from both city A and city B is on time.

$$P(B \cap C) = .85$$

$$P(A \cap C) = .86$$

$$P(A \cap B \cap C) = .81$$

What is the probability that at least one of the flights is on time? That is what is the probability that a flight from city A, or a flight from city B, or a flight from city C is on time?

We are seeking  $P(A \cup B \cup C)$ . The result may be obtained by inserting the above probabilities directly into the right hand side of Theorem A.6, obtaining .96.

## A.2 Conditional Probability

So far we have been considering events that are subsets of the sample space  $S$ , for example, the probability that a 1 or 2 will be rolled using a fair, six-sided die. In this case, the event is the  $\{1,2\}$ , while  $S = \{1,2,3,4,5,6\}$ . In problems of conditional probability—of which Bayes's Theorem is the most obvious instance in speech recognition—we want to define the probability an event  $A$  considering only those outcomes of a random experiment that are elements of a prior event  $B$ . Consider the following example.

Example A.3 Suppose we are given a list of 20 student IDs that contain no information about gender or state of residence. We are also given the following summary table:

	Male (M)	Female (F)	Total
Washington (W)	5	8	13
Oregon (O)	3	4	7
Total	8	12	20

If one student is selected at random, the probability that that student lives in Oregon is given by  $P(O) = 7/20$ . Now suppose we are interested only in Oregon residents who are also male. The time the probability is  $3/8$ , because we are limiting ourselves first to males. We denote this by  $P(O|M)$ , read “the probability of O given M, or the probability that a student is from Oregon given that he is male.”

Notice that

$$P(O|M) = \frac{3}{8} = \frac{|O \cap M|}{|M|} = \frac{|O \cap M|/20}{|M|/20} = \frac{P(O \cap M)}{P(M)}$$

This leads to a definition of conditional probability.

#### Definition A.14, Conditional Probability

If  $P(B) > 0$ , the conditional probability of an event A, given that an event B has occurred, is given by:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

Multiplying both sides by  $P(B)$  gives us the multiplication rule:

## Definition A.15, Multiplication Rule

$$P(A \cap B) = P(B) * P(A|B)$$

Or

$$(A \cap B) = P(A) * P(B|A)$$

The following example illustrates the use of conditional probability.

## Example A.5

Suppose we are given an opaque box containing 3 red marbles and 7 green marbles. We are asked to close our eyes and choose two marbles at random. What is the probability that the first marble chosen was red and the second green. Let R be the event defined by the choice of a red marble and G be the event defined by the choice of the green marble. Then,

$$P(R) = 3/10$$

Now, having chosen a marble, there are only nine marbles left in the box, so

$$P(G|R) = 7/9$$

The probability of picking a red marble first followed by a green marble is given by the probability of the intersection of the event R and the event G:  $P(R \cap G)$ .

Invoking Definition A.15, we have:

$$P(R \cap G) = P(R) * P(G|R) = \frac{3}{10} * \frac{7}{9} = \frac{7}{30}$$

Notice that the results are the same if the green marble is chosen first. That is, the probability of first choosing a red marble followed by a green marble equals the probability of first choosing a green marble followed by a red marble and both are designated as  $P(R \cap G)$

The multiplication rule can be extended to three events by using the fact that sets are associative under intersection, namely that  $A \cap B \cap C = (A \cap B) \cap C$

For convenience, let  $A \cap B = F$

Then,

$$P(A \cap B \cap C) = P[(A \cap B) \cap C]$$

$$P(F \cap C) = P(F) * P(C|F)$$

$$= P(A \cap B) * P(C|P(A \cap B))$$

$$\text{But } P(A \cap B) = P(A) * P(B|A)$$

So,

$$P(A \cap B \cap C) = P(A) * P(B|A) * P(C|P(A \cap B))$$

The general formula for more than three events can be proven by mathematical induction.

For k events, it looks like this:

$$P(A_1 \cap A_2 \cap \dots \cap A_k) =$$

$$P(A_1) * P(A_2|A_1) * P(A_3|P(A_1 \cap A_2)) * \dots * P(A_k | P(A_1 \cap A_2 \cap \dots \cap A_{k-1}))$$

To see how this works in practice, consider the following example

### Example A.6

Four cards are to be dealt from a deck of ordinary playing cards. What is the probability of the four cards being a spade, a heart, a diamond, and a club, in that order?

Call the four events, S, H, D, C.

$$P(S \cap H \cap D \cap C) = P(S) * P(H|S) * P(D|(S \cap H)) * P(C|(S \cap H \cap D))$$

The four terms on the right hand side mean, in order:

- The probability that a spade is drawn from a deck of 52 cards. Since there are 13 spades (2 through Ace),  $P(S) = 13/52$
- The probability that a heart is drawn from a deck from which a spade has been removed. Since there are now only 51 cards,  $P(H|S) = 13/51$ .
- The probability that a diamond is drawn from a deck after both events S and H have occurred. Since events S and H each resulted in removing a card from the deck,  $P(D|(S \cap H)) = 13/50$ .
- The probability that a club is drawn from a deck after events S, H, and D have occurred. Since events S, H, and D each resulted in removing a card from the deck,  $P(C|(S \cap H \cap D)) = 13/49$ .

So,

$$P(S \cap H \cap D \cap C) = 13/52 * 13/51 * 13/50 * 13/49$$

### A.3 Independent Events

Given a pair of events, it is important to know if the occurrence of one of them changes the probability of the occurrence of the other. In a coin flipping experiment where we are interested in the order of heads and tails on two consecutive flips, the sample space is

$$S = \{HH, HT, TH, TT\}.$$

The probability of each of the outcomes in the sample space is  $\frac{1}{4}$ . Now, let's define three events:

$$C = \{TT\}$$

$$B = \{\text{heads flipped first}\} = \{HH, HT\}$$

$$A = \{\text{tails flipped first}\} = \{TH, TT\}$$

By Theorem A.5,  $P(A) = P(HT) + P(TT) - P(HT \cap TT) = \frac{1}{4} + \frac{1}{4} - 0 = \frac{1}{2}$ .

But if we are told that  $C$  has already occurred, since  $C$  is a subset of  $A$ , there is only one possibility left. So,  $P(A|C) = 1$ . More formally,

$P(A|C) = P(A \cap C)/P(C)$ . But  $A \cap C = C$ . So  $P(A|C) = P(C)/P(C) = 1$ . In effect, the prior occurrence of  $C$  affected the probability of  $A$ .

Now suppose we are told that  $B$  has occurred.

$$P(A|B) = P(A \cap B)/P(B). \quad A \cap B = \{HT\}. \quad \text{Its probability is } \frac{1}{4}. \quad \text{So, } P(A|B) = (1/4)/(1/2) = \frac{1}{2}.$$

Notice that the prior occurrence of  $B$  did not affect the probability of  $A$ . When this is the case, we say that the two events are independent. That is, two events are independent if the occurrence of one of them has no effect on the probability that the other will occur. That is two events are independent if at least one of the following equalities holds:

$$P(A|B) = P(A) \text{ if } P(B) > 0 \text{ or } P(B|A) = P(B) \text{ if } P(A) > 0.$$

It is easy to show that the second of the equalities also holds using the data from the example.

Now, from the definition of conditional probability (Def. A.14), we have the following relationships:

$$P(A|B) = \frac{P(A \cap B)}{P(B)}$$

$$P(B|A) = \frac{P(B \cap A)}{P(A)}$$

So,

$$P(A \cap B) = P(A|B) * P(B) = P(A) * P(B) \quad (1)$$

$$P(B \cap A) = P(B|A) * P(A) = P(B) * P(A) \quad (2)$$

Because intersection and multiplication are commutative, (2) can be rewritten:

$$P(A \cap B) = P(A) * P(B)$$

Which leads to following definition:

Definition A.16, Independence

Two events, A and B, are said to be *independent* if and only if

$$P(A \cap B) = P(A) * P(B)$$

The definition of independence can be extended to more than two events as follows.

Definition A.17

Any number of events A, B, C, ... N are said to be *mutually independent* if and only

if both of the following conditions hold:

- a. Each pair meets the definition of independence
- b.  $P(A \cap B \cap C \cap \dots \cap N) = P(A) * P(B) * P(C) * \dots * P(N)$

#### Example A.7

A crucial component of an airliner has three levels of redundancy built in. If level A fails, control is passed to level B. If level B fails, control is passed to level C. The probability of failure of each component is .02. The levels are mutually independent. What is the probability that the system will not fail.

Let F be the event that all systems fail.  $P(F) = P(A \cap B \cap C)$ . F' is the event that the system will not fail.  $P(F') = 1 - P(F) = 1 - .02 * .02 * .02 = .999992$

#### Example A.8

Suppose we have a bowl with three red balls, two blue balls, and 4 green balls. We are to draw three balls, replacing each after it has been drawn. Clearly, drawing one ball does not affect the drawing of the next, since the first ball is replaced. Then the probability of drawing a red, blue, and green ball or two reds and a green is given by:

$$P(\{RBG\} \text{ or } \{RRG\}) = 3/9 * 2/9 * 4/9 + 3/9 * 3/9 * 4/9$$

### A.4 Bayes' Theorem

Bayes' theorem, named after an 18<sup>th</sup> century English cleric challenged to defend the gospels, is concerned with how the probability of a past event affects the conditional probability of a current event. Consider the following example. Suppose we have three bowls, A, B, C each containing marbles as follows:



- A contains two red and four white marbles
- B contains one red and two white marbles
- C contains five red and four white marbles.

The bowls are organized in such a way that the probability of selecting each is given by:

- $P(A) = 1/3$
- $P(B) = 1/6$
- $P(C) = 1/2$

In this experiment we select a bowl and then draw a marble from that ball. What is the probability of drawing a red marble,  $P(R)$ ? Clearly, the outcome is affected by which bowl we select, the probability of selecting a red marble from, C being higher ( $5/9$ ) than that of either B ( $1/3$ ) or A ( $2/6$ ). So, selecting a red marble is dependent first on which bowl we select and then on the probability of selecting a red marble from that bowl. Notice that once we have selected a bowl, the probability of selecting a red marble is independent of the bowl selection. Further, the bowl selections are also independent.

Though this is intuitively obvious, let's refer to the definition of conditional probability and represent the probability of selecting a red marble from bowl A like this:

$$P(R|A) = \frac{P(R \cap A)}{P(A)}$$

The probability of selecting a red marbles from bowls B and C may be represented in a similar fashion.

So,

$$P(R \cap A) = P(R|A) * P(A)$$

But since the R and A are independent,  $P(R \cap A) = P(R) * P(A)$ . We can do this for each bowl, giving:

- $P(R \cap A) = P(R) * P(A)$  (1)
- $P(R \cap B) = P(R) * P(B)$  (2)
- $P(R \cap C) = P(R) * P(C)$  (3)

Since we want a red marble from either A or B or C, we can write the probability this way:

R is the union of sets whose probability we have represented in 1, 2 , and 3:

$$R = (R \cap A) \cup (R \cap B) \cup (R \cap C)$$

Since these sets are non-intersecting, we have:

$$\begin{aligned} P(R) &= P(R \cap A) + P(R \cap B) + P(R \cap C) \\ &= P(A) * P(R|A) + P(B) * P(R|B) + P(C) * P(R|C) \quad (3) \\ &= 1/3 * 2/6 + 1/6 * 1/3 + 1/2 * 5/9 = 4/9 \end{aligned}$$

Since we were given the probabilities of choosing the various bowls and the number of marbles by color in each bowl, we have been able to compute:

- The probability of drawing a red marble from bowl A
- The probability of drawing a red marble from bowl B
- The probability of drawing a red marble from bowl C
- The probability of drawing a red marble from bowl A or bowl B or bowl C

Suppose we change things around a bit. Someone tells us that he has drawn a red marble.

What is the probability that this red marble was drawn from bowl A? That is, instead of computing  $P(R|A)$  and so forth, we want to compute  $P(A|R)$ . Said another way, what is the probability of bowl A having been chosen if we know that a red marble was previously selected.

From the definition of conditional probability, we have:

$$\begin{aligned}
 P(A|R) &= \frac{P(A \cap R)}{P(R)} = \frac{P(A) * P(R|A)}{P(R)} \\
 &= \frac{P(R|A)*P(A)}{P(A) * P(R|A) + P(B) * P(R|B) + P(C) * P(R|C)} \quad (4)
 \end{aligned}$$

We can compute  $P(B|R)$  and  $P(C|R)$  in the same fashion, noting that the denominator remains the same.

$$P(B|R) = \frac{P(R|B)*P(B)}{P(R)} \quad (5)$$

and

$$P(C|R) = \frac{P(R|C)*P(C)}{P(R)} \quad (6)$$

Notice that when the given probabilities are plugged into equations (4), (5), and (6), the resulting probabilities agree with our intuition, namely, that if a red marble is observed, the probability that it came from bowl C is higher since there is a greater fraction of red marbles in bowl C than in either bowl A or B:

$$P(A|R) = 2/8$$

$$P(B|R) = 1/8$$

$$P(C|R) = 5/8$$

$P(R)$  is usually called the prior probability.  $P(A|R)$ ,  $P(B|R)$ , and  $P(C|R)$  are called the conditional probabilities.

It is easy to generalize the foregoing. Since choosing a bowl is an event, that is a set of trials, the sample space,  $S = A \cup B \cup C$ . We can think of events A, B, C as mutually exclusive and exhaustive partitions of the sample space. So, for a sample space, S, with m partitions, we have:

$$S = B_1 \cup B_2 \cup \dots \cup B_m$$

where each  $B_i$  is a mutually exclusive partition. Suppose the prior probabilities of B with respect to the partitions are positive. That is  $P(B_i) > 0$ ,  $i = 1, \dots, m$ . In bowl world, this simply means that S represents of the choice of bowl 1, the choice of bowl 2, and so on.

Suppose A is an event that occurs with each of the B mutually exclusive partitions:

$$A = (B_1 \cap A) \cup (B_2 \cap A) \cup \dots \cup (B_m \cap A)$$

Then,

$$P(A) = \sum_{i=1}^m P(B_i \cap A) = \sum_{i=1}^m P(B_i) * P(A|B_i) \quad (7)$$

Notice that equation (7) is just equation (3) generalized to m bowls. If  $P(A) > 0$ , the definition of conditional equality tells us that:

$$P(B_k|A) = \frac{P(B_k \cap A)}{P(A)} \quad (8)$$

Using the definition of conditional probability and replacing  $P(A)$  in (8) with (7) we get the general form of Bayes Theorem:

$$P(B_k|A) = \frac{P(B_k) * P(A|B_k)}{\sum_{i=1}^m P(B_i) * P(A|B_i)}$$

With just two independent events, X and Y, the simple form is:

$$P(X|Y) = \frac{P(Y|X) * P(X)}{P(Y)}$$

The following example illustrates the use of Bayes theorem for two events.

#### Example A.9

A part gotten from supplier X has a 2% failure rate. The same part gotten from supplier Y has a 3% failure rate. A manufacturer purchases 40% of its parts from X and 60% from Y.

What is the probability that a part selected at random from the mixed parts will fail?

We know:

$$P(F|X) = .02$$

$$P(F|Y) = .03$$

$$P(X) = .4$$

$$P(Y) = .6$$

The event that a part will fail can be expressed like this:

$$F = (F \cap X) \cup (F \cap Y)$$

So,

$$P(F) = P(F \cap X) + P(F \cap Y)$$

$$= P(X) * P(F|X) + P(Y) * P(F|Y)$$

$$= .4 * .02 + .6 * .03 = .026$$

Now, given that a part failed, what is the probability that it was purchased from supplier X?

Here's where Bayes's theorem comes into play:

$$P(X|F) = \frac{P(F|X) * P(X)}{P(F)} = \frac{.02 * .4}{.026} = .3076$$

#### Example A.10

The example concerning three bowls and marbles of various colors that opens this section illustrates the use of Bayes' Theorem with more than two events.

## Appendix B Finite State Automata, Hidden Markov Models, Dynamic Programming

### B.0 Finite State Automata<sup>65</sup>

Bayes Theorem is easy enough to compute for discrete events. I provide several examples in Appendix A. It becomes more difficult when we are trying to compute the probability of a *sequence* of events. One technique used is known as a hidden Markov model (HMM), a machine learning mechanism that makes Bayesian inferences for chains of events. When combined with the Viterbi algorithm that performs the remarkable feat of approximating exponentially complex computations in polynomial time, we have the distinctive architecture of statistical speech recognition systems.

One of the key ideas in computer science is that of a finite state automata (FSA). The FSA belongs to a class of abstract machines whose best-known member is the Turing Machine, an abstraction of a computer. While a Turing Machine, according to the Church-Turing hypothesis, can model any process defined with suitable specificity, an FSA models only a very restricted subset of these, known as *regular languages*<sup>66</sup>. An FSA consists of a set of states, including well-defined start and end states, a set of rules for moving between states, a set of acceptable tokens, and an input string. If, when the input string is presented to the FSA, the machine moves from the start to the end state, the input string is said to a member of the language accepted by that particular FSA. Figure B.1 shows a newspaper vending modeled as an FSA, where the states indicate how much money is still needed to

---

<sup>65</sup> See Appendix A for a primer on probability theory. The material in Appendix A is based on Sudkamp (2006), Jurafsky and Martin (2009), and Luger (2009).

<sup>66</sup> FSAs are an important component of the theory of computer science. This is a very short and informal introduction. For more detail, see Jurafsky and Martin, Chapter 2. For a full, formal, yet readable presentation, see Sudkamp (2006).

open the box. The tokens are the legal U.S. coins; the rules are arcs that tell what happens after a specific coin is inserted; and the input string is a sequence of coins.

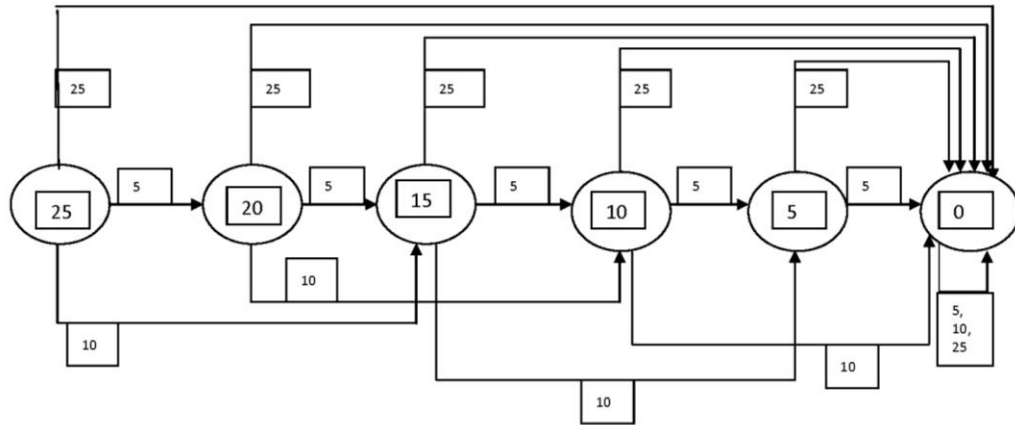
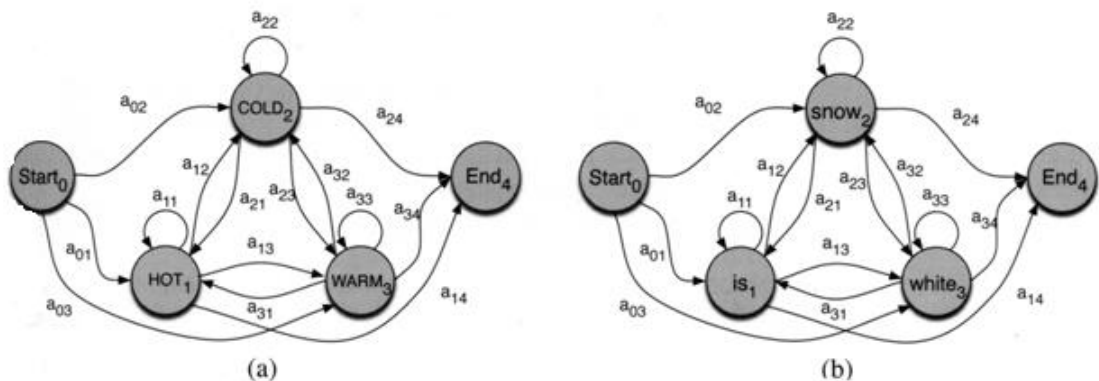


Figure B.1, Finite State Automata Modeling a Newspaper Vending Machine

An FSA whose arcs are mapped to a number, indicating a cost or weight associated with that arc, is called a *weighted finite state automata*. When that cost or weight is the probability of taking a certain path, the weighted finite state automata is known as a *Markov chain*. Figure B.2 shows two Markov chains borrowed from Jurafsky and Martin (2009, p. 175). The labels associated with each arc are assumed to be associated with probabilities. By presenting the Markov chain B.2b with a word string, say, “is snow white,” we can evaluate the probability of that sequence occurring (given a corpus, of course, and a mechanism for assigning probabilities).





## Figure B.2, Markov Chains

### B.1 Hidden Markov Models

Markov chains embody a simplifying assumption, namely that the probability of being in a current state depends only on the probability of having passed through  $N$  previous states, where  $N$  is known as the order of the Markov chain (or model). We call this the *Markov Assumption*. Since the arcs represent the probability of going from one state to another, the clear implication is that the probabilities associated with all arcs leaving a state sum to 1. Luger (2009, p. 374) offers a precise definition of a Markov Chain (or Model):

A graphical model is called an (observable) Markov model if its graph is directed and the probability of arriving at any state  $s_t$  from the set of states  $S$  at a discrete time  $t$  is a function of the probability distributions of its being in previous states of  $S$  at previous times. Each state  $s_t$  of  $S$  corresponds to a physically observable situation. An observable Markov model is first-order if the probability of it being in the present state  $s_t$  at any time  $t$  is a function only of its being in the previous state  $s_{t-1}$  at the time  $t-1$ , where  $s_t$  and  $s_{t-1}$  belong the set of observable states  $S$ .

The following example also comes from Luger (2009). Suppose we have weather data for a particular time, say noon, at a particular location. The weather in our system can only be in one of four discrete states,  $s_1$ :sunny,  $s_2$ :cloudy,  $s_3$ :foggy,  $s_4$ :rainy. Having observed the weather for many days we summarize our findings in a table that shows the probability of weather changing from one discrete state to another. It should be clear that the tabular format is equivalent to the graphical representation found in figures B.1 and B.2. That is, given a graphical representation, we can construct an equivalent table and given a tabular representation, we can construct an equivalent graph.

	s <sub>1</sub>	s <sub>2</sub>	s <sub>3</sub>	s <sub>4</sub>
s <sub>1</sub>	0.4	0.3	0.2	0.1
s <sub>2</sub>	0.2	0.3	0.2	0.3
s <sub>3</sub>	0.1	0.3	0.3	0.3
s <sub>4</sub>	0.2	0.3	0.3	0.2

**Figure B.3, Tabular Representation of a Markov Chain**

Suppose that today is sunny. We can determine the probability of next five days being sunny, sunny, cloudy, cloudy, rainy. To do this, we have to appeal to probability theory, specifically conditional probability. Suppose O is the observation sequence s<sub>1</sub>, s<sub>1</sub>, s<sub>1</sub>, s<sub>2</sub>, s<sub>2</sub>, s<sub>4</sub>, and M is the Markov model that stores the probabilities. Then we want to find the probability of O using M. We notate this relationship P(O,M). What we have here is a chain of conditional probabilities, where the probability of each subsequent state depends on (and only on) the previous state. Expanded, we have:

$$P(O,M) = P((s_1, s_1, s_1, s_2, s_2, s_4), M)$$

Since each state, by the Markov Assumption, is dependent only on the previous state, the transitions are independent of one another. So, the joint probability represented by chain of conditional probabilities is calculated by multiplying them:

$$P(O,M) = P(s_1) * P(s_1|s_1) * P(s_1|s_1) * P(s_2|s_1) * P(s_2|s_2) * P(s_4|s_2)$$

Notice that if today is sunny, the probability of today being sunny is one. Further, notice that the probability of P(s<sub>1</sub>|s<sub>1</sub>) and, in general, the P(s<sub>i</sub>|s<sub>j</sub>) is just the probability of moving from state i to state j. These can be read directly from the table. So,

$$P(O,M) = 1 * .4 * .4 * .3 * .3 * .3 = .00432$$

When a sequence of events is directly observable, like a sequence of words (or weather states), whose probabilities we are trying to compute, Markov chains are a useful model. In the case of speech recognition, what we are trying to compute—a string representing what was said—is not directly observable. What *is* observable is an acoustic signal. The output is the most probable sequence of words, call it a *string*, given that event. Since the output is not directly observable, it is said to be hidden.

When the output is not directly observable but is, in fact, probabilistically related to a state, we have what is called a *hidden Markov model* or HMM. Thus, in our recognizer acoustic events are directly observable. Syllables are hidden since they are probabilistically related to acoustic events. We can extend this further but developing a Probabilistic Concept Component (See Section 9). In the Probabilistic Concept Component, syllables are directly observable. Concepts are hidden since they are probabilistically related to syllables.

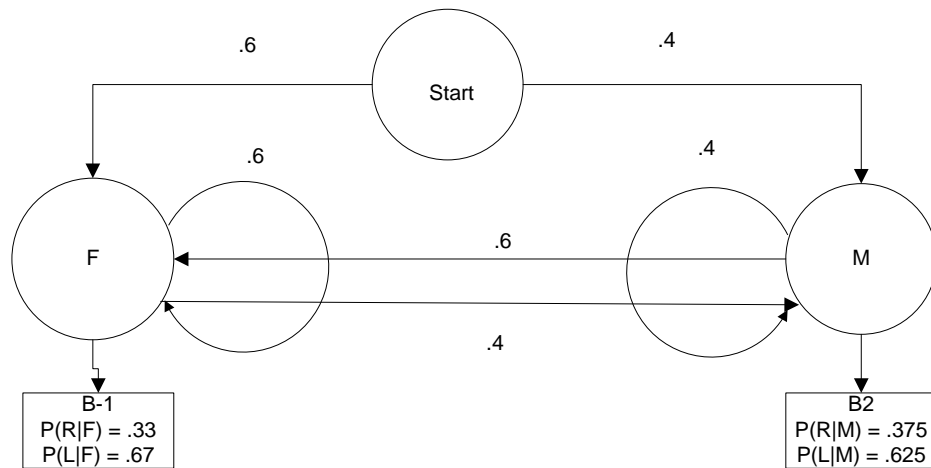
An HMM requires a further simplifying assumption called *Output Independence*: each observation is probabilistically related to the state of the model and not on any other observations or states. As we will see, this makes it easier to compute the joint probability of chains of conditional probabilities.

Consider the following table that provides (invented) data on student gender (M or F) and handedness (L or R).

	Male (M)	Female (F)	Total
Left (L)	5	8	13
Right(R)	3	4	7
Total	8	12	20

**Figure B.4, Handedness/Gender Data**

We can represent this data with a first-order Hidden Markov Model, as shown in Figure B.5.



**Figure B.5, Hidden Markov Model**

The model has three hidden states, Start, F, and M. There are transitional probabilities between three states that are directly computable from the data in the table. Suppose that the data is represented on cards with handedness on the face side and gender on the back (or hidden) side. Since there are twelve female students in the deck of twenty students, the probability of selecting a female student at random is .6. Notice that in this simple problem, the probability of selecting one gender or another does not depend on the state of the machine. So, the probability of selecting a card representing a female is the same whether the model is in state Start, state F, or state M. It is assumed that we also know the *observation likelihoods*, represented in B1 and B<sub>2</sub>. These can be computed directly from the table. For example, since three of the eight males are right-handed, the probability of choosing a card representing a right-handed student given that he is male student is  $3/8 = .375$ . Just to demonstrate that our reading of the table adheres to the definition of conditional

probability<sup>67</sup>, we notate the probability of selecting a right-handed student, given that he is male as  $P(R|M)$ . By the definition of conditional probability,

$$P(R|M) = \frac{P(R \cap M)}{P(M)} \quad (1)$$

$P(R \cap M)$  is the probability of both events  $R$  and  $M$  occurring. Since three of eight males are right-handed out of a total of twenty students this is  $3/20$ .  $P(M) = 8/20$ . So, the right-hand side of (1) is  $3/8 = .375$ . Notice, finally, that the sum of the probabilities of the transitions from any state is 1 as is the sum of observation likelihoods for a given hidden state.

Now, suppose we choose three cards at random, recording the handedness information found on the front of each card and replacing that card before choosing the next. The sequence of handedness indicators recorded is our observation sequence. What is the probability of having drawn three lefties?

The technique will be clearer if we answer a simpler question first. Suppose we know a sequence of hidden states, say FFM, what is the probability that these correspond to the handedness sequence LLL? In an HMM, each state produces a single observation, and, according to the output independence assumption, each output depends only on the state that produces it. In the language of probability theory, the observations are independent events. To compute their joint probabilities, we need only multiply the probability of each state producing each observation.

If  $Q$  is a set of states  $q_1, q_2, \dots, q_t$  and  $O$  is a sequence of observations  $o_1, o_2, \dots, o_t$  then we can compute the likelihood of the observation sequence given the state sequence as

---

<sup>67</sup> See Appendix A

$$P(O|Q) = \prod_{i=1}^t P(o_i|q_i) \quad (2)$$

$$P(\text{LLL}|\text{FFM}) = P(\text{L}|\text{F}) * P(\text{L}|\text{F}) * P(\text{L}|\text{M}) = .667 * .667 * .375 = .169$$

Yet, the basis of the HMM is that the state sequence that produced the observation sequence is hidden from view. It is nice enough that a given state sequence has a certain probability of producing a certain set of observations, but we have to remember that being in a certain state is itself a probabilistic event, one, in fact, that depends on the Markov Assumption: the probability of being in a certain state depends only on the probability of having passed through N previous states where N, in our example, is 1. The probability of being in state  $q_j$  is based on having passed through state  $q_{j-1}$  and nothing else. This means that the state transitions are independent of one another and that their joint probability can be computed by multiplying their individual probabilities. So,

$$P(Q) = \prod_{i=1}^n P(q_i|q_{i-1}) \quad (3)$$

Recall the multiplication rule for conditional probability:

$$P(O \cap Q) = P(O|Q) * P(Q)$$

$P(O \cap Q)$  represents the joint probability of generating a particular handedness sequence O while being in a certain state sequence Q. Expanding it, we have:

$$P(O \cap Q) = P(O|Q) * P(Q) = \prod_{i=1}^t P(o_i|q_i) * \prod_{i=1}^n P(q_i|q_{i-1}) \quad (4)$$

Recall that the original question was to determine the probability of drawing three consecutive lefties, LLL. The answer requires that we compute the joint probability of LLL over every possible three state sequence. That is, we want to compute the probability of LLL if MMM were the states or MMF were the states and so on for all sequences of three states. We know that if A and B are any two events, then  $P(A \cup B) = P(A) + P(B) - P(A \cap B)$  where A and B represent any legal sequence of three states. Since these are permutations

are non-intersecting,  $P(A \cap B) = 0$ . Therefore, to compute the joint probability of LLL over every possible three state sequences, we add the probability of LLL for each individual three state sequence:

$$P(LLL) = P(LLL|MMM) * P(MMM) + \dots + P(LLL|FFF) * P(FFF)$$

For the general case, this becomes:

$$P(O) = \sum_Q P(O \cap Q) = \sum_Q P(O|Q) * P(Q) \quad (5)$$

where  $P(O|Q)$  and  $P(Q)$  are as defined in (2) and (3).

This example is small enough to compute by hand.

$$\begin{aligned} P(LLL) = & P(LLL|MMM) * P(MMM) + P(LLL|MMF) * P(MMF) + \\ & P(LLL|MFM) * P(MFM) + P(LLL|MFF) * P(MFF) + \\ & P(LLL|FMM) * P(FMM) + P(LLL|FMF) * P(FMF) + \\ & P(LLL|FFM) * P(FFM) + P(LLL|FFF) * P(FFF) \end{aligned} \quad (6)$$

Using the argument in (2),  $P(LLL|FFF)$ , for example is  $P(L|F) * P(L|F) * P(L|F)$ .

Substituting the values from Figure B.3 into (6), we get:

$$\begin{aligned} P(LLL) = & (.625 * .625 * .625 * .4 * .4 * .4) + (.625 * .625 * .667 * .4 * .4 * .6) + \\ & (.625 * .667 * .625 * .4 * .6 * .4) + (.625 * .667 * .667 * .4 * .6 * .6) + \\ & (.667 * .625 * .625 * .6 * .4 * .4) + (.667 * .625 * .625 * .6 * .4 * .6) + \\ & (.667 * .667 * .625 * .6 * .6 * .4) + (.667 * .667 * .667 * .6 * .6 * .6) \\ = & .015625 + .0250125 + .0250125 + .02669334 + .0250125 + .03751875 + .04004001 + .064096048 \\ = & .259010648 \end{aligned}$$

Let's imagine that we put our hands in a hat and pulled out three cards with L or R printed on them. If the Ls and Rs were evenly distributed,  $P(LLL) = P(L) * P(L) * P(L) = .5 * .5 * .5 = .125 = 1/8$ . Put another way, there are 8 permutations of 3 card sequences.  $P(LLL) = 1/8$ .

Notice that the probability computed through our HMM is considerably higher, almost .26.

This corresponds to our intuitive notion that there are simply more Ls than R's in the table. Though tedious, this computation is reasonable for an HMM with 3 states. What happens when the states grow. We need another approach, specifically the Viterbi algorithm, the subject of the next section.

### B.3 The Viterbi Algorithm

The HMM, in the language of computer science, is a data structure, an arrangement of data. We need an algorithm to compute, say,  $P(O)$ . This is no easy matter for realistic cases. Suppose  $Q$  is a variable that hold one of two values, L or R. Then the string  $QQQ$  can have eight different values, LLL, LLR, ..., RRR. The string  $QQQQ$  can have sixteen different values and, in general, the string  $Q_1 \dots Q_t$  can have  $2^t$  different values, where  $t$  is the number of observations. Informally, a set of computations that represents the amount of work equation (5) would have to do increases in proportion  $2^t$ . If the HMM had three hidden states, the complexity of (5) would jump to  $3^t$  and, in general, the complexity of computation using a hidden Markov model is  $N^t$ , where  $N$  is the number of hidden states and  $t$  is the number of observations. An algorithm that grows at least as quickly as  $2^p$ , where  $p$  is  $> 1$ , is called *exponentially complex*. That is, if  $p$  describes some component of the problem, say the number of observations, the number of computations that must be performed to solve the problem grows exponentially with  $p$ . Such an algorithm will not terminate in reasonable time.

In order for it to terminate in reasonable time, we have to reduce its complexity so that its growth is of *polynomial complexity*, that is, where the number of computations can be represented by a polynomial. Any exponentially complex algorithm grows faster than any



algorithm of polynomial complexity. In a very important class of problems, known as NP-complete problems, the reduction from exponential to polynomial time complexity appears not to be possible (Cormen, et al. 1991). Fortunately, since the late fifties, researchers have recognized that for certain problems, an optimal solution implies the optimal solution for subproblems computed on the way to the optimal solution for the problem as a whole. If these solutions are saved, then they don't need to be recomputed. This technique is referred to as *dynamic programming* and is widely used in speech recognition.

In the sample problem presented in Section B.2, we wanted to compute the probability of drawing three cards in a row representing left-handed students. In the language of the Probabilistic Concept Component, we were trying to compute the probability of a sequence of concepts. In fact, we are asking something more difficult of the Probabilistic Concept Component. We don't want the probability of a specific sequence, we want the most probable *sequence* of concepts given a syllable string. The single probability is computed using a dynamic programming technique called *the forward algorithm*, so-called, because the probabilities are successively computed as the algorithm moves along the observation sequence. The most probable sequence is computed by modifying the forward algorithm, Hansel and Gretel fashion, so that it includes backward pointers from the probability at time  $t$  to the probability at time  $t-1$  and so on until we reach start. This modification is called, in the general case, the *forward-backward* algorithm and, when used with probabilities, the *Viterbi* algorithm.

Instead of presenting forward or forward-backward when executing the data in an HMM, I will present the related, but considerably simpler problem, of computing how closely one string of characters resembles another. This is called the *minimum-edit distance*

or, sometimes, the *Levenshtein distance*. Computing the distance itself requires the forward algorithm. Reconstructing the sequence of edits that would be required to transform a source string to a target string (where each edit contributes to the distance) requires the forward-backward algorithm. The algorithm is important in spell checkers, where, given a misspelled string, the spell checker gives, say, the five strings most closely related to the one that is misspelled. It is very closely related to algorithms that attempt to find the optimal alignment between two strings. This is the technique used in *sclite*, the program used to score the output of speech recognizers (see Section 7), to compute minimum edit distance. It has also found its way into bioinformatics where it is used to align DNA sequences.

Suppose we want to determine how difficult it would be to transform the word *intention* to the word *execution*. We can do this through a sequence of five deletions (d), substitutions (s), and insertions (i) as shown below (Jurafsky and Martin 2009, p. 74):

```

I N T E * N T I O N
* E X E C U T I O N
d s   s   i s

```

**Figure B.6, *intention -> execution***

We assign deletion and insertion a cost of 1. Since a substitution requires both a deletion and an insertion, we assign it a cost of two. The transformation is assigned a cost of 8. But how did we arrive at this solution? The technique is to construct a matrix with the rows labeled by the source string and columns labeled by the target string. Each cell in the matrix, representing the minimal cost of a subproblem, is a function of three neighboring cells: its left neighbor, its bottom neighbor, and the previous diagonal. Specifically, it stores the

minimum of the neighboring cells plus the cost of transforming the letter labeling its row to the letter labeling its column, 1 insertion, 1 for deletion, 2 for substitution.

Figure B.7 (adapted from Jurafsky and Martin 2009, p. 76) shows the filled-in matrix. Suppose we indicate each cell in the forward matrix as `forward[row][column]`, counting from 0. Thus `forward[0][0]` is blank, `forward[0][1] = #`, `forward[1][1] = 0`, etc. `Forward[2][2] = 2`. This is the cost of transforming an i to an e. We compute it by adding 0, the cost of substitution to the minimum of the surrounding cells, `forward[2][1]` (the left neighbor), `forward[1][2]` (the bottom neighbor), `forward[1][1]` (the previous diagonal). Since the previous diagonal holds a 0, the valued stored in `forward[2][2]` is 2. The minimum edit distance is found in the upper right cell, 8, as predicted. This is the forward algorithm.

n	9	8	9	10	11	12	11	10	9	8
o	8	7	8	9	10	11	10	9	8	9
i	7	6	7	8	9	10	9	8	9	10
t	6	5	6	7	8	9	8	9	10	11
n	9	8	9	10	11	12	11	10	9	8
e	4	3	4	5	6	7	8	9	10	9
t	6	5	6	7	8	9	8	9	10	11
n	2	3	4	5	6	7	8	7	8	7
i	1	2	3	4	5	6	7	6	7	8
#	0	1	2	3	4	5	6	7	8	9
	#	e	x	e	c	u	t	i	o	n

**Figure B.7, A Distance Matrix for the Forward Algorithm**

To transform the forward to the forward-backward algorithm, we store at each cell, pointers to the neighbor that might have been used to produce the value in that cell. The forward matrix with pointers is found in Figure B.8 (from Jurafsky and Martin, p. 77).

<b>n</b>	9	↓ 8	↙↖ 9	↙↖ 10	↙↖ 11	↙↖ 12	↓ 11	↓ 10	↓ 9	↘ 8	
<b>o</b>	8	↓ 7	↙↖ 8	↙↖ 9	↙↖ 10	↙↖ 11	↓ 10	↓ 9	↘ 8	← 9	
<b>i</b>	7	↓ 6	↙↖ 7	↙↖ 8	↙↖ 9	↙↖ 10	↓ 9	↘ 8	← 9	← 10	
<b>t</b>	6	↓ 5	↙↖ 6	↙↖ 7	↙↖ 8	↙↖ 9	↘ 8	← 9	← 10	← 11	
<b>n</b>	5	↓ 4	↙↖ 5	↙↖ 6	↙↖ 7	↘ 8	↙↖ 9	↙↖ 10	↙↖ 11	↙↖ 10	
<b>e</b>	4	↙↖ 3	← 4	↙↖ 5	← 6	← 7	↙↖ 8	↙↖ 9	↙↖ 10	↓ 9	
<b>t</b>	3	↙↖ 4	↙↖ 5	↙↖ 6	↙↖ 7	↙↖ 8	↙↖ 7	← 8	↙↖ 9	↓ 8	
<b>n</b>	2	↙↖ 3	↙↖ 4	↙↖ 5	↙↖ 6	↙↖ 7	↙↖ 8	↓ 7	↙↖ 8	↙↖ 7	
<b>i</b>	1	↙↖ 2	↙↖ 3	↙↖ 4	↙↖ 5	↙↖ 6	↙↖ 7	↙↖ 6	← 7	← 8	
<b>#</b>	0	1	2	3	4	5	6	7	8	9	
	<b>#</b>	<b>e</b>	<b>x</b>	<b>e</b>	<b>c</b>	<b>u</b>	<b>t</b>	<b>i</b>	<b>o</b>	<b>n</b>	

**Figure B.8, The Distance Matrix for the Forward-Backward Algorithm**

Notice, for example, that forward[2][2] holds a 2 and three arrows, pointing to each of its neighbors, indicating that the minimum extension into that cell could have come from any one of the three previous cells. Forward[6][2] holds a single down arrow indicating that it was computed only from its bottom neighbor (since  $1 + 3 < \text{either } 1 + 5 \text{ or } 2 + 4$ , the computations possible using the other neighbors). The operations that produced the path from the upper right corner to the lower left can be reconstructed based on the type of arrow. Following one of the paths (there could be many) allows us to construct an alignment path of the form shown in Figure B.6.

The Viterbi algorithm, used to compute the most probable sequence of concepts given a syllable string, is just an extension of the minimum edit distance algorithm. The most important point for our purposes is that its running time can be described with a polynomial. How? Intuitively, notice that the longer the edit strings, the larger the matrix. Suppose that one string is of length  $p$ , the other of length  $q$ , and for each cell in the matrix, we have to do  $r$  computations. For the sake of simplicity, assume all are less than some integer,  $x$ . Then the complexity of the problem grows in proportion to  $x^3$ , a polynomial. The good news for us is that the exponentially complex HMM computation done manually at the end of section B.2

can be done in polynomial time. This becomes important when the number of observations (and hidden states) grows, as they must in a real-world problem.

## Appendix C The Training Transcript

The text in this appendix forms the basis for the language models used in Sections 3, 4, and 5. It was captured using the Wizard-of-Oz protocol (WOZ) by engineers at the Next It Corporation in 2007. The WOZ protocol is common in the development and training of LVCSR systems. The idea is to have a person, the wizard, carry on a conversation (by keyboard and monitor) with one or more persons who have been told that they are interacting with a computer. The resulting transcript, is cleansed of the wizard's responses, leaving only what engineers hope is a typical human-computer interaction in the problem domain. The conversants in this case were temporary workers hired by Next It to help construct a language model for an LVCSR (Next It, 2008; Huang et al., 2001).<sup>68</sup>

I WANT TO FLY TO SPOKANE  
 I WANT TO FLY TO SPOKANE  
 I WANT TO FLY FROM SEATTLE  
 TWO PEOPLE  
 I WANT TO FLY TO SEATTLE  
 I'M FLYING FROM SPOKANE WITH THREE PEOPLE  
 NEXT THURSDAY  
 FEBRUARY EIGHTH  
 I WANT TO FLY TO SEATTLE  
 SPOKANE

---

<sup>68</sup> There is some possibility that these are transcriptions of Next It employees asked to simulate a conversation with a virtual travel agent (See Appendix D, Charles Wooter, personal communications 3/7/2010). It makes little difference in this case. The role of the transcript was, in effect, to bootstrap a system that did not yet exist. As a system is used, more complete language modeling transcripts can be generated. These were used to get the system running the first place. For this study, a common language model useable across every experiment was all that was needed. Appendix C served this purpose.

ONE  
 FEBRUARY SIXTH  
 THE NEXT DAY  
 I WOULD LIKE TO FLY TO SEATTLE FROM SPOKANE  
 SEATTLE  
 SPOKANE  
 TWO  
 TOMORROW  
 ON FRIDAY  
 I WANT TO FLY FROM SAN DIEGO TO PHOENIX  
 THREE PEOPLE AND WERE LEAVING ON THE FIFTEENTH  
 WE'LL BE RETURNING THE FOLLOWING FRIDAY  
 WANT TO FLY FROM SPOKANE TO SEATTLE UM WITH THREE PEOPLE  
 LEAVING ON THE SIXTH AND RETURNING ON FEBRUARY EIGHTH  
 UH LEAVING FEBRUARY SIXTH  
 FEBRUARY SIXTH  
 I WANT TO FLY FROM NEW YORK CITY TO ATLANTA  
 WEIRD OKAY ATLANTA GEORGIA  
 THREE PEOPLE AND WERE LEAVING ON FEBRUARY TENTH  
 THE FOLLOWING FRIDAY  
 I WANT TO FLY FROM NEW YORK CITY TO MIAMI  
 THREE PEOPLE LEAVING ON FEBRUARY TWENTY EIGHTH  
 FEBRUARY TWENTY EIGHTH  
 THE FOLLOWING WEDNESDAY  
 I WANT TO FLY FROM WASHINGTON D C FROM MEMPHIS  
 MEMPHIS  
 WASHINGTON D C  
 ONE  
 LABOR DAY  
 THANKSGIVING  
 I'D LIKE A CALL TO OR I'D LIKE A TICKET TO ARIZONA  
 PHOENIX ARIZONA  
 PORTLAND  
 EIGHT  
 ON DECEMBER TWENTY FIFTH  
 JANUARY FIRST  
 UH ME AND MY HUSBAND WANT TO GO TO LOS ANGELES  
 NO NO ASPEN  
 FROM SPOKANE TO LOS ANGELES  
 TWO  
 UM NEXT MONTH  
 NO NEXT MONTH  
 I'D LIKE TO GO TO SPOKANE WASHINGTON  
 SAN DIEGO  
 JUST MYSELF

UM ON VALENTINE'S DAY  
 THE NEXT DAY  
 I WOULD LIKE TO GO TO LAS VEGAS  
 PHOENIX  
 UM MY MOM AND MYSELF  
 NEXT YEAR  
 APRIL TWENTY NINTH  
 NEVER  
 I'M NOT  
 I AM NOT COMING BACK  
 I'M GOING TO SEATTLE AND I'M LEAVING FROM TUCSON  
 THEY'LL BE A DOZEN OF US  
 UM JUNE THIRD  
 JUNE THIRTIETH  
 I WOULD LIKE TO BOOK A TICKET FROM SAN DIEGO CALIFORNIA TO LOS  
 ANGELES  
 ME AND MY DOG  
 UH FEBRUARY SIXTEENTH  
 THE SAME DAY  
 THE SAME DAY  
 UM I WOULD LIKE TO LEAVE FROM PORTLAND AND ARRIVE AN  
 I AM TRAVELING TO SEATTLE  
 I AM LEAVING FROM UM LET'S SAY LAS VEGAS  
 UM THEY'LL BE TWO OF US  
 NO I MEAN FROM UH THE OTHER WAY AROUND I WANT TO GO FROM  
 SEATTLE TO VEGAS  
 TOO AWESOME JUNE THIRTIETH  
 I'D LIKE TO BOOK A FLIGHT FROM SPOKANE TO ONTARIO CALIFORNIA UH  
 LEAVING ON JANUARY FIFTEENTH AND RETURNING JANUARY TWENTY  
 THIRD  
 FOUR  
 JANUARY TWENTY NINTH  
 COULD I GET A FLIGHT TO ONTARIO CALIFORNIA  
 UH SEATTLE WASHINGTON  
 TWO CHILDREN ONE ADULT  
 UH JANUARY FIFTEENTH  
 SEPTEMBER EIGHTEENTH  
 I'D LIKE TO FLY TO LOS ANGELES CALIFORNIA AND UH I'LL HAVE TWO  
 PEOPLE  
 NEW HAMPSHIRE  
 NEW HAMPSHIRE  
 OCTOBER TWENTY THIRD  
 SEPTEMBER FIRST  
 I'D LIKE TO FLY FROM ADDICK ISLAND  
 I'D LIKE TO FLY TO BILLINGS MONTANA ON HALLOWEEN



BOSEMAN  
BILLINGS  
TWO CHILDREN  
JUNE FIRST  
JUNE SIXTEENTH  
I'D LIKE TO BOOK A FLIGHT FROM MOSCOW IDAHO TO ORANGE COUNTY  
CALIFORNIA ON SEPTEMBER FIFTEENTH  
MOSCOW IDAHO  
MOSCOW  
NOME  
ORANGE COUNTY  
PETERSBURG  
FOURTEEN  
UH FIRST OF SEPTEMBER  
HALLOWEEN  
I'D LIKE TO GET A TICKET LEAVING SPOKANE  
I'D LIKE A TICKET LEAVING FROM SEATTLE  
EL PASO  
THREE  
FEBRUARY SEVENTEENTH  
MARCH FIRST  
I'D LIKE TO TRAVEL FROM SEATTLE TO BOSTON LEAVING ON FEBRUARY  
SIXTH AND RETURNING ON FEBRUARY SIXTEENTH WITH TWO PASSENGERS  
BOSTON  
SPOKANE  
TWO  
EASTER WEEKEND  
THAT'S WRONG  
I'D LIKE A TICKET FROM SPOKANE WASHINGTON TO EL PASO TEXAS  
SPOKANE WASHINGTON  
ME AND MY HUSBAND  
FOURTH OF JULY  
WE WANT LEAVE JULY THIRD AND RETURN JULY SEVENTH  
I'D LIKE A TICKET FROM SEATTLE TO SPOKANE PLEASE  
UH THEY'LL BE SIX OF US WE'LL BE LEAVING ON FEBRUARY FIFTH AND  
RETURNING ON THE EIGHTH  
FEBRUARY FIFTH  
I'D LIKE TO BOOK A FLIGHT FOR FOUR PEOPLE TO FLY FROM CHICAGO TO  
MIAMI  
MARCH SIXTH TO THE SEVENTEENTH  
MARCH SEVENTEENTH  
I'D LIKE TO FLY FROM BOSTON TO NEW YORK ON APRIL SECOND AND  
RETURN ON APRIL NINTH  
NEW YORK  
JUST ME

APRIL SECOND AND I'LL RETURN ON APRIL SEVENTH  
I'D LIKE TWO TICKETS FROM PHOENIX TO SAN JOSE  
TWO  
WE'LL LEAVE ON THE SEVENTH OF JULY AND RETURN ON THE TENTH  
I'D LIKE THREE TICKETS FROM BOISE TO SAN FRANCISCO  
DECEMBER ELEVENTH UNTIL THE EIGHTEENTH  
I'D LIKE TO FLY FROM SEATTLE TO PORTLAND ON MAY EIGHTH  
IT'LL BE SEVEN OF US AND WE'LL NEED TO COME BACK ON THE SEVENTH  
I'D LIKE EIGHT TICKETS FROM LOS ANGELES TO SAN DIEGO ON THE FIFTH OF  
AUGUST UNTIL SEPTEMBER NINTH PLEASE  
UM THERE WILL BE FOUR OF US  
OCTOBER TENTH  
I WOULD LIKE TO FLY FROM SPOKANE TO SAN FRANCISCO  
SIX PEOPLE  
CHRISTMAS  
NEW YEAR'S EVE  
I WANT TO BOOK A FLIGHT FROM DALLAS TO SPOKANE ON FEBRUARY FIFTH  
JUST ME  
THE NEXT DAY  
I NEED TO FLY ON UNITED TO UH D C ON MEMORIAL DAY FOR TWO PEOPLE  
SPOKANE WASHINGTON  
ON JUNE SEVENTH  
I NEED TO FLY FROM SPOKANE WASHINGTON TO INDIANAPOLIS INDIANA ON  
THURSDAY  
ONE  
ON SATURDAY  
I NEED TO TRAVEL BY MYSELF ON BUSINESS FROM SPOKANE TO SEATTLE  
TOMORROW  
JUST MYSELF  
UH THE SAME DAY IN THE EVENING  
ON THE SAME DAY  
IN THE EVENING  
UH THE NEXT MORNING  
ON WEDNESDAY  
I NEED TO FLY WITH MY FAMILY TO DISNEYLAND  
SPOKANE WASHINGTON  
TWO ADULTS AND TWO CHILDREN  
UH THIS COMING SUNDAY AND RETURN THE FOLLOWING SUNDAY  
I NEED TO TRAVEL WITH UH FOUR PEOPLE TO DISNEY WORLD UH LEAVING  
ON THURSDAY AND RETURNING IN ONE WEEK  
DISNEY WORLD  
BOISE IDAHO  
ON SUNDAY THE TENTH OF FEBRUARY ON THE SEVENTH  
I NEED TO TRAVEL ON BUSINESS FROM TO SPOKANE WASHINGTON TO BOISE  
IDAHO TOMORROW AND RETURN ON FRIDAY

JUST MYSELF

I NEED TO TRAVEL FROM SPOKANE TO OHARE TOMORROW AND RETURN ON FRIDAY

ONE

I NEED TO TRAVEL FROM BOISE IDAHO TO SEATTLE TOMORROW I'D LIKE AN AISLE SEAT

ONE

THAT SAME NIGHT

ON FRIDAY

I NEED TO LEAVE TOMORROW MORNING FROM SPOKANE AND ARRIVE IN CHICAGO BY AFTERNOON JUST ONE OF US IS TRAVELING

ONE

ON THURSDAY

THE NEXT DAY

I'D LIKE TO GET A TICKET PLEASE

I'D LIKE TO GET A TICKET TO SEATTLE

I WANT TO FLY FROM SEATTLE

TRAVELING TO SAN DIEGO

SEATTLE

FOUR PEOPLE

FEBRUARY TWELVETH

ON EASTER

I'D LIKE TO BOOK A FLIGHT FROM SPOKANE WASHINGTON TO SAN JOSE CALIFORNIA LEAVING ON FEBRUARY FIFTEENTH AND RETURNING ON FEBRUARY TWENTY SECOND FOR TWO PEOPLE

SPOKANE

I NEED TO BOOK A FLIGHT FROM SEATTLE TO LOS ANGELES

FOUR

FRIDAY FEBRUARY FIFTEENTH

SUNDAY FEBRUARY TWENTY FOURTH

I NEED TO BOOK A FLIGHT FROM SPOKANE TO BOSTON

ONE

MARCH SEVENTEENTH

MARCH TWENTIETH

I NEED TO BOOK A FLIGHT FROM MIAMI TO CHICAGO FOR FOUR PEOPLE LEAVING ON APRIL THIRD AND RETURNING ON APRIL TENTH

APRIL THIRD

I NEED TO BOOK A FLIGHT FOR FEBRUARY TWENTY FIFTH FOR THREE PEOPLE

SAN ANTONIO

OMAHA

FEBRUARY TWENTY FIFTH

MARCH THIRD

I NEED TO BOOK A FLIGHT

I NEED TO BOOK A FLIGHT TO SAN FRANCISCO

MISSOULA

TWO

MARCH SEVENTH

MARCH TWENTY FIFTH

I'D LIKE TO FLY FROM UH SPOKANE WASHINGTON TO DES MOINES IOWA  
AND I WANT TO LEAVE ON FEBRUARY TWELVETH AND I'D LIKE RETURN ON  
UH MARCH THIRD AND THEY'LL BE FOUR OF US

I WANT TO FLY TO SPOKANE

YEAH I'D LIKE TO FLY TO UM SPOKANE

SPOKANE

CEDAR RAPIDS

FOUR

JULY SEVENTH THROUGH JULY TWENTY THIRD

YEAH I'D LIKE TO PURCHASE A TICKET FROM SEATTLE TO SAN FRANCISCO  
UM FROM CHRISTMAS TO UM THE SUPER BOWL  
FOR FOUR PEOPLE SAN FRANCISCO

FOUR

CHRISTMAS TO NEW YEAR'S

JANUARY SECOND

YES I'D LIKE TO GET A TICKET FROM SPOKANE WASHINGTON TO PHOENIX  
ARIZONA LEAVING ON FEBRUARY TWELVETH RETURNING ON FEBRUARY  
SEVENTEENTH FOR EIGHT PEOPLE

EIGHT

YES I'D LIKE TO FLY FROM SEATTLE TO PHOENIX NEXT THURSDAY AN COME  
BACK ON THE FOLLOWING SUNDAY FOUR PEOPLE

HI I WANT TO BUY A AIRLINE TICKET

YEAH THERE'S FIVE PEOPLE THAT WANT TO FLY TO MIAMI FLORIDA IN TWO  
WEEKS FROM TODAY

UH SPOKANE

UH WE WANT TO STAY THERE FOR TEN DAYS

CAN I HAVE FOUR PEOPLE INSTEAD OF FIVE

YEAH I WANT TO FLY FROM RENO TO SPOKANE

IT'S GOIN' BE LIKE SIX PEOPLE

WANNA LEAVE UH CHRISTMAS COME BACK TO WEEKS LATER

I I NEED A TICKET FROM SPOKANE TO CHICAGO FOR NEXT SATURDAY  
RETURNING THE FOLLOWING MONDAY ONE PERSON

SATURDAY

YEAH I NEED TO UH BUY A TICKET TO SAN FRANCISCO

MY HOME TOWN

YEAH I NEED A FLIGHT TO BREWSTER WASHINGTON

I NEED A FLIGHT FROM SPOKANE TO SEATTLE

THERE WILL BE THIRTEEN OF US

WE NEED TO LEAVE TOMORROW AND RETURN ON THE TWENTY FIRST

DO YOU FLY TO BUTTE MONTANA

I'M LEAVING FROM BOSTON

JUST ME  
I WILL BE LEAVING AUGUST TWENTY FIRST  
IN ONE WEEK  
YEAH I NEED A TICKET  
I'M TRAVELING TO MINNEAPOLIS  
NEW YORK  
JUST MY WIFE  
JUST MY DAUGHTER  
HALF A DOZEN  
THREE  
UH EASTER WEEKEND  
THREE DAYS LATER  
OH I GUESS I NEED A TICKET  
OH I'M TRAVELING FROM SPOKANE TO SPOKANE  
THIRTY SIX  
THIS WEEKEND  
THIS WEEKEND  
A WEEK FROM THIS TUESDAY  
NINE DAYS LATER  
I NEED A FLIGHT FROM SPOKANE TO SEATTLE A WEEK FROM THIS TUESDAY  
FOR SIX DAYS WITH THREE PEOPLE ATTENDING  
THREE DAYS LATER  
I NEED A FLIGHT FROM BOSTON TO SAINT LOUIS  
NO I NEED TO FLY TO SPOKANE  
I NEED TO FLY FROM BOSTON TO SPOKANE  
I NEED TO FLY FROM NEW YORK TO SPOKANE  
I NEED TO FLY FROM BOSTON TO J F K  
ONE THOUSAND THIRTY SIX  
THERE ARE THIRTY SIX PEOPLE PLANNING TO FLY  
A WEEK FROM YESTERDAY  
NEVER  
I AM NOT RETURNING IT'S A ONE WAY FLIGHT  
YESTERDAY  
UM I'D LIKE TO GO FROM SPOKANE TO UH UH MIAMI  
THREE  
UH I'D LIKE TO LEAVE UH THIS THURSDAY AND COME BACK NEXT  
THURSDAY  
I'D LIKE TO FLY THIS THURSDAY TO NEXT FRIDAY UH I'D LIKE TO LEAVE  
FROM SPOKANE AND FLY TO OMAHA AND THERE'S GOING TO BE TWO OF US  
AH THIS THURSDAY TO NEXT FRIDAY  
AH FEBRUARY SIXTEENTH  
I'D LIKE TICKETS FOR TWO PEOPLE FROM UH PORTLAND OREGON TO BOSTON  
MASSACHUSETTS AND I'D LIKE TO LEAVE THIS FRIDAY AND COME BACK  
NEXT FRIDAY  
UM TOMORROW

I'D LIKE TO MAKE RESERVATIONS FOR TWO FROM SPOKANE WASHINGTON  
TO SAN DIEGO DEPARTING ON THE FIFTEENTH OF FEBRUARY RETURNING ON  
THE EIGHTEENTH

THE EIGHTEENTH OF FEBRUARY

HI I'D LIKE TO GO TO SAN DIEGO FROM SPOKANE AND LEAVING ON THE  
FOURTEENTH FRIDAY OF FEBRUARY AND RETURNING ON MONDAY THE  
NINETEENTH OF FEBRUARY

I'D LIKE TO FLY FROM MIAMI TO NEW YORK TO NEW YORK CITY  
JUST MYSELF

THIS WEDNESDAY RETURNING THE FOLLOWING THURSDAY

WANT TO FLY FROM MIAMI TO PHOENIX

FOUR PEOPLE LEAVING THIS THURSDAY

WE'RE LEAVING THIS THURSDAY RETURNING THE FOLLOWING FRIDAY

I WANT TO LEAVE FROM MIAMI TO PHOENIX

JUST MYSELF

NEXT FRIDAY

I WANT TO FLY FROM L A TO NEW YORK CITY

FOUR PEOPLE

A WEEK FROM YESTERDAY

THAT FRIDAY

I WANT TO FLY FROM L A TO NEW YORK CITY WITH FOUR PEOPLE

THIS FRIDAY

FOLLOWING DAY

THE FOLLOWING DAY

THE NEXT DAY

I WANNA FLY FROM L A TO SAN DIEGO WITH FOUR PEOPLE

NEXT FRIDAY

THE SAME DAY

THE SAME DAY

THAT DAY

THAT DAY

FRIDAY THE EIGHTH

WANT A FLIGHT FROM SEATTLE TO THE BIG APPLE WITH FOUR PEOPLE

A WEEK FROM THIS WEDNESDAY

A WEEK AFTER THAT

I WANT A FLIGHT FROM MIAMI TO PHOENIX

JUST MYSELF

THIS FRIDAY

THE FOLLOWING WEDNESDAY

I WANT TO BY MYSELF FROM SEATTLE TO NEW YORK CITY

I'LL BE BY MYSELF

JUST ME

A WEEK FROM YESTERDAY

THE FOLLOWING WEEK

A WEEK AFTER THAT

I NEED A TICKET FOR ONE PERSON FROM SEATTLE TO NEW YORK CITY  
LEAVING THIS THURSDAY RETURNING A WEEK AFTER THAT  
THIS THURSDAY AND RETURNING A WEEK AFTER THAT  
ACTUALLY WE'RE LEAVING THIS THURSDAY AND RETURNING A WEEK AFTER  
THAT  
I WOULD LIKE TO FLY FROM SPOKANE TO SEATTLE  
I'D LIKE TO FLY FROM SPOKANE TO SEATTLE  
ONE PERSON  
THE SIXTEENTH OF FEBRUARY  
NEVER  
FEBRUARY TWENTY EIGHTH  
HI I'D LIKE TO FLY FROM SEATTLE TO BOSTON LEAVING MARCH THIRD  
COMING BACK MARCH SIXTEENTH  
SEATTLE  
TWO ADULTS  
LIKE TO BOOK A FLIGHT  
I'D LIKE TO BOOK A FLIGHT  
LAS VEGAS  
PORTLAND OREGON  
THREE PEOPLE  
DECEMBER FIRST RETURNING DECEMBER FIFTEENTH  
I'D LIKE TO TRAVEL FROM BOSTON TO ORLANDO  
FIFTEEN  
FEBRUARY SECOND RETURNING BACK FEBRUARY TWENTY SECOND  
I'D LIKE TO BOOK A FLIGHT  
IT'LL BE FLYING FROM LOS ANGELES TO PHOENIX ON THE APRIL FIFTEENTH  
COMING BACK APRIL TWENTY FIFTH  
ONE  
MAY THIRD  
I'D LIKE TO FLY FROM SPOKANE TO LOS ANGELES ONE PERSON ON JUNE  
THIRD RETURNING JULY FIFTH  
JUNE THIRD RETURNING JULY FIFTH  
HI DO YOU HAVE A FLIGHT AVAILABLE FROM SEATTLE TO SPOKANE  
ONE PERSON WILL BE FLYING ON FEBRUARY THIRTEENTH RETURNING  
FEBRUARY THIRTIETH  
FEBRUARY TWELFTH RETURNING  
FEBRUARY THIRTEENTH  
HI DO YOU HAVE ANYTHING AVAILABLE FROM SEATTLE TO BOULDER ON  
THE FIFTH OF APRIL  
THIRTEEN PEOPLE  
JUNE THIRD  
I WOULD LIKE TO BOOK A FLIGHT FROM SPOKANE TO LAS VEGAS  
SPOKANE  
TWO  
FEBRUARY FOURTEENTH

THE TWENTY SECOND  
I WOULD LIKE TO BUY A TICKET FROM SPOKANE TO CHICAGO LEAVING  
FEBRUARY FOURTEENTH  
FIVE  
UH FEBRUARY TWENTY SECOND  
I WANT TO FLY TO LOS ANGELES  
SPOKANE  
THERE SHOULD BE FOUR PEOPLE LEAVING ON JANUARY EIGHTEENTH  
ONE WEEK LATER  
I'D LIKE TO GO TO DISNEYLAND  
SOMEWHERE AROUND DISNEY LAND HOW ABOUT LOS ANGELES  
THREE OF US WILL BE LEAVING FROM SEATTLE  
WE'D LIKE TO LEAVE ON MARCH EIGHTH AND RETURN ON MARCH  
FIFTEENTH  
I LIVE IN SPOKANE AND I WOULD LIKE TO GO TO SEATTLE FOR THE  
WEEKEND  
THREE ADULTS AND ONE CHILD  
WE'D LIKE TO LEAVE NEXT MONDAY  
TWO WEEKS LATER AND ACTUALLY THERE'S FOUR PEOPLE NOT THREE  
I'D LIKE TO BOOK SOME TRAVEL FOR FIVE PEOPLE TO GO TO DES MOINES  
IOWA  
DES MOINES IOWA  
WE'D LIKE TO LEAVE FROM SEATTLE  
WE WOULD LIKE TO LEAVE MARCH NINETEENTH AND BE THERE FOR TWO  
WEEKS  
WE'D LIKE TO RETURN TWO WEEKS LATER  
I HAVE A FAMILY OF SIX AND WE WOULD LIKE TO FLY FROM DES MOINES TO  
SPOKANE  
SIX OF US WOULD LIKE TO LEAVE ON MARCH TWENTIETH  
WE'D LIKE TO RETURN AFTER TEN DAYS  
YOU HAVE FLIGHTS FROM SEATTLE TO NEW YORK  
I THINK WILL HAVE FOUR NO FIVE PEOPLE TOTAL  
WE WOULD LIKE TO BE GONE TWO WEEKS COMING HOME ON MARCH  
THIRTIETH  
I'D LIKE TO FLY FROM SEATTLE TO PHOENIX  
FIVE PEOPLE  
AUGUST FIFTEENTH  
LIKE TO BOOK A TICKET  
LICK TO BOOK A TICKET FROM PORTLAND TO MISSOULA  
TWO PEOPLE  
FEBRUARY THIRD COMING BACK FEBRUARY TENTH  
I'D LIKE TO FLY TO ANCHORAGE ON FEBRUARY TENTH  
SPOKANE WASHINGTON  
FOUR  
FEBRUARY TWENTY EIGHTH



I'D LIKE TO FLY FROM DALLAS TO SPOKANE ON FEBRUARY TENTH  
RETURNING FEBRUARY FIFTEENTH WITH TWO PASSENGERS  
I'D LIKE TO FLY FROM SPOKANE TO BALTIMORE ON MARCH TENTH  
ONE PERSON  
MARCH TWELFTH  
CAN I FLY TO SAN FRANCISCO  
SAN FRANCISCO  
DALLAS  
TWO  
JUNE FIFTEENTH  
A WEEK LATER  
I'D LIKE TO GO TO ALBUQUERQUE NEW MEXICO ON APRIL FIRST  
ALBUQUERQUE NEW MEXICO  
PHOENIX ARIZONA  
PHOENIX  
DALLAS  
ONE PERSON  
APRIL FIRST  
THREE DAYS LATER  
I'D LIKE A TICKET LEAVING FROM SPOKANE  
LOS ANGELES  
SPOKANE  
FIVE PEOPLE  
JANUARY TWENTY EIGHTH  
FEBRUARY TWENTY THIRD  
I WOULD LIKE TO TRAVEL FROM SPOKANE TO BOSTON ON JUNE FIFTEEN  
RETURNING JULY TWENTY NINTH  
ONE PASSENGER  
I LIKE TO FLY TO BALTIMORE ON JUNE THIRTY FIRST  
SPOKANE  
TWO  
JULY FIRST  
CAN I FLY TO BALTIMORE FROM SPOKANE ON MARCH FIFTEENTH  
ONE PERSON  
MARCH FIFTEENTH  
I WOULD LIKE A TICKET TO LOS ANGELES CALIFORNIA FOR TWO PEOPLE  
SPOKANE WASHINGTON  
MARCH SIXTH  
MARCH TWELVETH  
I WOULD LIKE A TICKET FOR TWO PEOPLE FROM SEATTLE WASHINGTON TO  
FORT RUCKER ALABAMA  
ATLANTA  
JULY FOURTH  
JULY EIGHTEENTH

I WOULD LIKE TICKET FOR TWO PEOPLE LEAVING SEATTLE WASHINGTON  
ARRIVING ORLANDO FLORIDA MARCH SIXTH RETURNING ON MARCH  
EIGHTEENTH

SEATTLE WASHINGTON

MARCH EIGHTEENTH

I WOULD LIKE A TICKET FOR ONE PERSON TRAVELING FROM SEATTLE  
WASHINGTON TO SPOKANE WASHINGTON ON MARCH NINTH RETURNING  
LATE THAT NIGHT

MARCH TENTH

IN NEED A TICKET FOR FEBRUARY SIXTEENTH FOR ONE PERSON FROM  
SPOKANE WASHINGTON TO LOS ANGELES CALIFORNIA RETURNING ON  
FEBRUARY EIGHTEENTH

LOS ANGELES CALIFORNIA

SPOKANE WASHINGTON

I'D LIKE TO BOOK A FLIGHT

I'D LIKE TO UH LEAVE FROM SEATTLE AND FLY TO BOSTON ON FEBRUARY  
EIGHTH AND RETURN FEBRUARY TENTH

FIFTY

CHANGE TO FEBRUARY TWELVETH

I WANT TO BOOK A FLIGHT LEAVING FROM SPOKANE FLYING TO NEW YORK  
LEAVING DECEMBER TWENTY FIFTH RETURNING DECEMBER TWENTY FIFTH  
NO I'D LIKE TO LEAVE FROM SPOKANE AND FLY TO NEW YORK

ONE MILLION

ONE THOUSAND

SAME DAY

THURSDAY DECEMBER TWENTY FIFTH

I WANT TO FLY TO HAWAII

HAWAII

WILBUR

SPOKANE

TWO PEOPLE WILL BE LEAVING ON

MARCH THIRD AND RETURNING MARCH TENTH

UH LOOKING FOR A FLIGHT TO SEATTLE TODAY

SPOKANE

JUST ONE

SAME DAY

TODAY

UM DO YOU HAVE ANY FLIGHTS FROM SPOKANE GOING TO UH PORTLAND  
OREGON LEAVING TODAY AND RETURNING TOMORROW

ONE

TOMORROW

I NEED A TICKET TO SAN ANTONIO WHERE IN THE HELL IS SAN ANTONIO UH  
SAN ANTONIO TEXAS SAN ANTONIO TEXAS

I'M GONNA LEAVE FROM WHAT'S A GOOD CITY SAN JUAN NO I'M GOING  
LEAVE FROM NEW YORK

THERE WILL BE FIVE PEOPLE NO NO NO THERE'S GOING TO BE SIX PEOPLE  
TRAVELING WITH ME TODAY  
I'M GOING TO TRAVEL ON THE SEVENTEENTH AND RETURN ON THE TWENTY  
SECOND  
SIXTEENTH NO DANG IT  
I WANT TO FLY TO NEW YORK  
OH I'M FLYING TO NEW YORK CITY  
I'M LEAVING FROM SPOKANE WASHINGTON  
ONE  
I WOULD LIKE TO GET SOME TICKETS  
I'D LIKE TO GO TO BOSTON FROM SPOKANE NEXT WEEK  
I'D LIKE TO FLY TO BOSTON  
I'M GOING TO BOSTON FROM SPOKANE  
BOSTON  
SPOKANE ON FEBRUARY SEVENTEENTH  
THREE  
FEBRUARY SEVENTEENTH  
MARCH SECOND  
I'D LIKE A ROUND TRIP TICKET FROM SEATTLE TO SPOKANE  
TWO  
APRIL SECOND THROUGH MAY FOURTH  
I'D LIKE TO FLY FROM LOS ANGELES TO ANCHORAGE  
MY GRANDPARENTS AND MYSELF  
THREE  
I LIKE TO TRAVEL TO ANCHORAGE FROM LOS ANGELES WITH THREE PEOPLE  
LEAVING ON MARCH SECOND AND RETURNING ON MARCH SEVENTEENTH  
MARCH SECOND THROUGH MARCH SEVENTEENTH  
I WANT TO GO TO HOUSTON ON MARCH SECOND AND RETURN TO SPOKANE  
ON MARCH TWENTY FIRST  
NO I WANT TO GO TO HOUSTON FROM SPOKANE  
NO I WANT TO GO TO HOUSTON FROM SPOKANE  
FOUR  
MARCH FIRST THROUGH MARCH THIRD  
I WANT TO FLY TO AUSTIN FROM SEATTLE ON MARCH FIRST AND RETURN  
MARCH SIXTH  
TWO  
I NEED TO GO TO HONOLULU AND I WANT TO LEAVE APRIL SECOND AND  
RETURN APRIL TENTH  
SEATTLE  
MY HUSBAND AND I  
I NEED TO GO TO SAN DIEGO FROM SAN JOSE IN APRIL  
TWO  
UM EASTER  
A WEEK LATER

MY HUSBAND AND I WANT TO FLY TO MAUI FROM SEATTLE ON JUNE  
SEVENTH AND RETURN TWO WEEKS LATER

TWO

I WANT TO GO TO NEW YORK FROM BUFFALO ON DECEMBER SEVENTH AND  
RETURN A WEEK AFTER CHRISTMAS

FOUR

THREE OF US NEED TO FLY FROM SPOKANE TO SEATTLE LEAVING  
TOMORROW AND COMING BACK IN TEN DAYS

FOUR

THREE PEOPLE NEED TO FLY FROM LOS ANGELES TO SPOKANE ON  
FEBRUARY SEVENTEENTH AND RETURN ON FEBRUARY NINETEENTH

FOUR

I NEED TO FLY TO PORTLAND AND I NEED TO LEAVE NEXT WEEK

PORTLAND

PORTLAND OREGON

SPOKANE

THREE

I NEED TO FLY TO PORTLAND OREGON FROM SPOKANE WASHINGTON

PORTLAND OREGON

TOMORROW

A WEEK LATER

I WOULD LIKE TO FLY FROM PHOENIX TO SPOKANE

THERE WILL BE TWO OF US

WE WOULD LIKE TO LEAVE DECEMBER TWELVETH

WE'LL BE COMING BACK JANUARY FIFTH

HELLO I WOULD LIKE TO TRAVEL FROM SAN FRANCISCO TO PHILADELPHIA

THERE'S JUST ONE OF US JUST ME

I WOULD LIKE TO GO TOMORROW

I WOULD LIKE TO COME BACK UM THE FOLLOWING WEDNESDAY

HI I WANT FIVE TICKETS TO FLY FROM SAN ANTONIO TO DETROIT

SAN ANTONIO TEXAS

I WOULD LIKE TO LEAVE ON VALENTINE'S DAY

WE WILL BE COMING BACK MARCH TWENTY SECOND

HELLO I WOULD LIKE A ONE WAY TICKET FROM NEW YORK

I'M GOING TO GO TO CHICAGO

NEW YORK NEW YORK

THERE IS A COLLEGE TEAM SO THEY'LL BE TWENTY FIVE

I WILL BE LEAVING THIS FRIDAY WHICH IS UH FEBRUARY NOT REALLY SURE

I THINK THE EIGHTH

WE WILL BE RETURNING ON FEBRUARY THE TWENTY NINTH WHICH IS ALSO

A FRIDAY

HI I NEED TO BOOK A FLIGHT FROM DALLAS

I'M ACTUALLY GOING TO GO TO JACKSONVILLE FLORIDA

I AM LEAVING FROM SAN EXCUSE ME JACKSONVILLE FLORIDA

SORRY I MADE A MISTAKE I AM TRAVEL FROM SAN ANTONIO TEXAS TO JACKSONVILLE FLORIDA

TWO

I NEED A TICKET FROM EL PASO

I'M GOING TO BE LEAVING FROM MILWAUKEE

THERE GOING TO BE FIVE OF US

WE WOULD LIKE TO LEAVE APRIL THIRTEENTH

WE'LL BE COMING BACK APRIL EIGHTEENTH

HI I WOULD LIKE TO BOOK A FLIGHT FROM ALBUQUERQUE NEW MEXICO TO LONG BEACH CALIFORNIA

I'M GOING OT GO TO LONG BEACH

LONG BEACH CALIFORNIA

LONG BEACH CALIFORNIA

YES I WOULD LIKE TO GET A ONE WAY TICKET FROM OKLAHOMA CITY TO LAS VEGAS NEVADA

LAS VEGAS

OKLAHOMA

OKLAHOMA CITY

HELLO

OKLAHOMA OKLAHOMA

I'D LIKE TO TRAVEL FROM SPOKANE TO SEATTLE ON FEBRUARY UH

FIFTEENTH AND RETURN ON FEBRUARY TWENTIETH

THERE WILL BE TWO ADULTS AND TWO CHILDREN

I'D LIKE TO UM I'D LIKE TO TRAVEL FROM SPOKANE TO SEATTLE THERE ARE FOUR PEOPLE IN OUR PARTY WE'D LIKE TO DEPART ON APRIL FIRST AND RETURN ON APRIL SIXTH

UH WE'D LIKE TO LEAVE ON APRIL FIRST AND RETURN ON APRIL FIFTH

THERE ARE FOUR PEOPLE IN MY FAMILY WHO WOULD LIKE TO TRAVEL FROM SPOKANE TO HONOLULU ON APRIL FIRST AND RETURNING ON THE FOLLOWING SATURDAY

THERE WILL BE FOUR IN OUR PARTY

I'D LIKE TO TRAVEL FROM SPOKANE TO PHILADELPHIA ON FEBRUARY TWENTIETH AND RETURN ON FEBRUARY TWENTY FIFTH THERE WILL BE FOUR PEOPLE IN OUR PARTY

I LIKE TO YA TO UH CREATE A RESERVATION FOR ONE PERSON TO FLY FROM PHILADELPHIA TO MIAMI FLORIDA ON UH APRIL FIRST AND RETURN ON UH THE FOLLOWING SATURDAY

I'M SORRY WILL YOU REPEAT THE QUESTION

UM UH WE HAVE FIVE IN OUR PARTY WE WOULD LIKE TO DEPART FROM SEATTLE ON APRIL FIRST AND ARRIVE UH THE SAME DAY IN SAN FRANCISCO WE WOULD LIKE TO RETURN FROM SAN FRANCISCO TO SEATTLE ON APRIL FIFTH

ON APRIL FIRST I WOULD LIKE TO FLY FROM PHOENIX TO SEATTLE THERE WILL BE FIVE IN OUR PARTY

FIVE

THE FOLLOWING SATURDAY

I'D LIKE TO TRAVEL FROM L A X TO G E G ON APRIL FIRST AND RETURN ON APRIL FIFTH

TWO ADULTS TWO CHILDREN

I'D LIKE TO PLAN TRAVEL FROM UH HAWAII HONOLULU HAWAII UH TO SPOKANE ON APRIL FIFTEENTH AND RETURNING THE FOLLOWING SUNDAY

TWO ADULTS AND TWO CHILDREN

I'D LIKE TO FLY FROM MISSOULA TO MIAMI ON FEBRUARY FIFTEENTH AND RETURN THE FOLLOWING MONDAY

THREE ADULTS AND TWO CHILDREN

ON THE FOLLOWING MONDAY

I WOULD LIKE TO FLY TO HONOLULU PLEASE

PHOENIX

THERE WILL BE ONE CHILDREN AND FOUR ADULTS

WE WOULD LIKE TO LEAVE ON APRIL FIRST AND RETURN ON THE FOLLOWING MONDAY

I NEED TO BOOK SOME TRAVEL PLEASE

WASHINGTON D C

SEATTLE WASHINGTON

THERE WILL BE FIVE CHILDREN AND TWO ADULTS

MONDAY

THE FOLLOWING FRIDAY

I'D LIKE TO BOOK AIRLINE TRAVEL PLEASE

MOSCOW

CHARLOTTE NORTH CAROLINA

GREAT FALLS

GREAT FALLS MONTANA

BOISE IDAHO

THERE'LL BE MY WIFE AND I

TUESDAY

THE FOLLOWING MONDAY

UM I WANT TO BOOK A FLIGHT FROM SPOKANE TO MIAMI ON PRESIDENTS DAY AND RETURN ON FOURTH OF JULY

TWELVE

I'D LIKE TO BOOK RESERVATIONS

I WANNA TRAVEL TO MIAMI AND LEAVE ON VALENTINE'S DAY

SAN DIEGO

ZERO

MY GRAND FATHER

EASTER

I'D LIKE TO BOOK A FLIGHT UH LEAVING ON UH SAINT PATRICK'S DAY AND RETURNING ON THE FOURTH OF JULY FOR FOUR PEOPLE

OH HONOLULU

G E G

UM I WANNA TO BOOK A FLIGHT UM VALENTINE'S DAY AND WITH FIVE NO  
 TWO PEOPLE  
 UM SPOKANE AND UM UH RETURN LET'S UH HONOLULU  
 UH VALENTINES DAY  
 PRESIDENT'S DAY  
 UM I'D LIKE TO SCHEDULE A FLIGHT  
 I NEED TO FLY UH FLY SOMEWHERE  
 SEATTLE TO SPOKANE TOMORROW  
 UH TWO NO ONE  
 CHRISTMAS  
 I NEED T BOOK A FLIGHT  
 THE GRAND CANYON  
 ORLANDO  
 MYSELF AND MY TWO DAUGHTERS AND MY DAD  
 NOW  
 YESTERDAY  
 TUESDAY  
 SATURDAY  
 I WANT TO FLY FROM SEATTLE TO UM SALT CITY  
 SIX PEOPLE  
 UH NEVER MIND MAKE THAT FOUR PEOPLE  
 FEBRUARY EIGHTH  
 TWO WEEKS FROM TODAY  
 CAN I FLY FROM SPOKANE TO NEW YORK CITY ON JANUARY SECOND  
 RETURNING AUGUST SECOND  
 TWO PEOPLE  
 FROM SEATTLE TO SPOKANE JANUARY FOURTH RETURNING JANUARY  
 SIXTEENTH FOR FOUR PEOPLE  
 SEATTLE  
 I CHANGES MY MIND CAN I RETURN JANUARY TWENTY SECOND  
 I WANT TO BOOK A FLIGHT  
 I WANT TO TRAVEL TO HONOLULU  
 I WANT TO LEAVE FROM SEATTLE  
 TWENTY SEVEN PEOPLE  
 UM FEBRUARY SIXTEENTH  
 FEBRUARY THIRTY SECOND  
 FEBRUARY EIGHTEENTH  
 I WANT TO FLY TO UH ALBUQUERQUE  
 ALBUQUERQUE  
 UH SAN ANTONIO  
 UY TOKYO  
 SEATTLE  
 ME MY DOG AND TWO CATS  
 FEBRUARY TWENTIETH  
 FEBRUARY TWENTIETH

I'D LIKE TO GET A TICKET  
DENVER  
DENVER  
SPOKANE  
I WOULD LIKE TO FLY FROM SPOKANE TO SEATTLE TOMORROW RETURNING  
ON SUNDAY  
SPOKANE  
SIXTEEN  
I NEED TO FLY TO MILWAUKEE TOMORROW  
MILWAUKEE  
DALLAS  
MYSELF AND MY WIFE  
THIS SUNDAY  
THREE WEEKS FROM SUNDAY  
I NEED TO FLY FROM SPOKANE TO SEATTLE TOMORROW RETURNING ON  
NEXT TUESDAY TWO PEOPLE  
NEXT TUESDAY  
I NEED TO FLY TO ORLANDO  
SANTA MARIA  
MYSELF MY WIFE AND MY TWO CHILDREN  
NEXT FRIDAY  
THE FOLLOWING DAY  
I WOULD LIKE TO FLY TO LONG BEACH FROM LOS ANGELES  
LOS ANGELES  
LONG BEACH  
LONG BEACH  
THIRTY SIX  
MARCH TWENTY SECOND  
THE FOLLOWING SUNDAY  
I NEED TO BOOK A TICKET TO SEATTLE  
TAMPA  
MYSELF AND MY TWO CHILDREN  
TOMORROW  
THREE DAYS LATER  
I NEED TO FLY TO MIAMI  
I AM LEAVING FROM IDAHO FALLS  
MY WIFE AND MY SELF MY TWO CHILDREN AND MY GRANDMA  
TONIGHT  
SIX WEEKS FROM TODAY  
I WOULD LIKE TO FLY TO TAMPA TONIGHT  
TAMPA BAY  
LOMPOC  
SAN LOUIS OBISPO  
MYSELF AND MY WIFE  
UH TOMORROW AFTERNOON



SIX DAYS LATER

I'D LIKE TO TRAVEL FROM SEATTLE TO BOSTON LEAVING ON FEBRUARY  
TWENTY EIGHTH RETURNING ON FEBRUARY TWENTY NINTH  
SIXTEEN

I NEED TO FLY FROM ORLANDO TO NEW YORK TOMORROW RETURNING  
THREE DAYS LATER WITH TWO PEOPLE  
COLUMBUS

I LIKE TO FLY FROM SPOKANE TO L A FEBRUARY SEVENTEENTH RETURNING  
TWO DAYS LATER

SPOKANE

JUST MYSELF

I NEED A FLIGHT FROM SPOKANE TO SAN DIEGO FEBRUARY NINETEENTH  
RETURNING THE TWENTY SECOND

ME AND MY WIFE

YEAH I NEED A TICKET FOR ME AND MY WIFE FLYING FROM KALISPELL  
MONTANA TO SEATTLE FEBRUARY TWENTIETH RETURNING ON THE TWENTY  
THIRD

SEATTLE

KALISPELL

MISSOULA

FEBRUARY TWENTY SIXTH

I NEED TO FLY TO DALLAS ON MARCH THIRD RETURNING THE FIFTH  
DALLAS

SPOKANE

ME AND MY GRANDMA

I NEED A TICKET TO FLY FROM LOS ANGELES TO WASHING D C MARCH THIRD  
RETURNING THE SEVENTH

ME AND MY WIFE AND MY TWO KIDS

I NEED A FLIGHT FROM L A TO DALLAS FER THREE PEOPLE AND ONE CHILD  
ON MARCH THIRD RETURNING THREE DAYS LATER

THREE ADULTS AND TWO CHILDREN

MARCH FIFTH RETURNING THREE DAYS LATER

I NEED AN EVENING FLIGHT FROM SEATTLE TO NEWARK ON MARCH THIRD  
RETURNING THE FIFTH FOR MYSELF

I NEED A FLIGHT FROM SEATTLE FROM TO SAN ANTONIO APRIL FRIST  
RETURNING ON THE THIRD

I WANT A FLIGHT TO SAN ANTONIO FROM SEATTLE  
ONE

I WANT A FLIGHT TO SAN ANTONIO FROM SEATTLE ON APRIL FIRST  
RETURNING ON THE THIRD

THREE

YEAH I WANT TO FLY FROM HONG KONG OR TO HONG KONG I SHOULD SAY  
FROM SPOKANE MARCH FIFTEENTH AND I'LL BE COMING BACK ON MARCH  
TWENTY THIRD

HONG KONG

BOISE  
 NEGATIVE FIVE  
 THREE BILLION SEVEN HUNDRED MILLION ONE HUNDRED AND SIXTY TWO  
 THOUSAND AND FORTY SEVEN  
 JUST ME  
 WHAT CITIES CAN I FLY TO  
 TOKYO  
 ITALY  
 GRANTS PATH  
 SEATTLE  
 BOISE  
 SEVENTY THREE  
 SEPTEMBER THIRTIETH TWO THOUSAND AND NINE  
 JULY FIRST  
 I WANT TO FLY FROM SPOKANE TO SEATTLE ON JULY THIRTY FIRST  
 I WANT TO FLY FROM SPOKANE TO SEATTLE ON JULY THIRTY THIRD  
 SPOKANE  
 FIVE  
 FEBRUARY THIRTIETH  
 I WANT TO FLY FROM SPOKANE TO SEATTLE ON MARCH RETURN MARCH  
 EIGHTH AND I'M GOING TO BE TAKING ONE PERSON WITH ME  
 YES I NEED A FLIGHT FROM WASHTUH WASHTUCK WASHINGTON TO  
 INDIANAPOLIS  
 I AM TRAVELING TO CRAP I DON'T KNOW I'M TRAVELING TO BUTTE  
 I I'M LEAVING FROM BOSTON  
 THREE  
 WE WILL BE TRAVELING THE TWENTY THIRD OF MAY ITS A FRIDAY  
 THE TWENTY FOURTH OF APRIL  
 YES I NEED A FLIGHT FOR THIS FRIDAY  
 I'M TRAVELING TO OKLAHOMA  
 LITTLE ROCK  
 I'M TRAVELING TO LITTLE ROCK  
 I'D LIKE TO BOOK A FLIGHT FROM OKLAHOMA CITY TO TULSA  
 I'M PLANNING ON LEAVING FROM OKLAHOMA CITY  
 KANSAS CITY  
 JUST ME  
 I NEED TO LEAVE THE DAY AFTER TOMORROW  
 NO I NEED TO LEAVE THE TWELVETH  
 YES I NEED A FLIGHT FROM PORTLAND TO PORTLAND TO PORTLAND TO TO  
 GREAT FALLS  
 GREAT FALLS  
 I'M TRAVELING TO MISSOULA MONTANA  
 I AM LEAVING FROM LEWISTON IDAHO  
 MY WIFE AND THREE KIDS  
 ACTUALLY MYSELF MY WIFE AND THREE KIDS

THERE ARE FIVE PEOPLE TRAVELING  
I'M PLANNING TO TRAVEL UH THE FIRST SATURDAY OF FEBRUARY  
NO I'M LEAVING THE FIRST SATURDAY  
I NEED A FLIGHT FROM MISSOULA TO PULLMAN  
THERE WILL BE FRED FRANK AND GEORGE  
I AM PLANNING TO TRAVEL UH ON HALLOWEEN  
ACTUALLY I LIKE TO FLY TO SPOKANE  
ACTUALLY THERE'S THREE PEOPLE TRAVELING  
THE NEXT DAY  
HI I'D LIKE TO BOOK A FLIGHT FROM OAKLAND TO SPOKANE  
JUST ONE PERSON  
I'D LIKE TO LEAVE NEXT TUESDAY AND RETURN THE FOLLOWING FRIDAY  
I'D LIKE TO GET A TICKET FOR ONE PERSON FLYING FROM OAKLAND TO  
SPOKANE NEXT TUESDAY RETURNING NEXT FRIDAY  
I WANT TO FLY FROM SEATTLE TO NEW YORK  
I WANT TO FLY FROM NEW YORK TO SEATTLE  
ME AND MY WIFE  
NEXT FRIDAY  
THE FOLLOWING WEDNESDAY  
CAN I FLY ROUND TRIP BETWEEN SAINT LOUIS AND SAINT BERNARD  
I'D LIKE TO FLY TO SEATTLE  
SPOKANE  
FOUR PEOPLE LEAVING ON MEMORIAL DAY  
THE FOURTH OF JULY  
YEAH I'D LIKE TO FLY TO RENO NEVADA FROM SPOKANE  
UH I THINK RIGHT NOW'S GOIN' TO BE THREE  
AH I'D LIKE TO LEAVE A WEEK FROM FRIDAY  
NO I WANT TO LEAVE ON THE FIFTEENTH  
NO I WANT TO RETURN ON THE TWENTY SEVENTH  
I WANT TO FLY TO NEXT IT CORPORATE OFFICE I WANT TO FLY TO SPOKANE  
BATON ROUGE  
ONE  
NEXT MONDAY  
THE FOLLOWING FRIDAY  
YEAH I NEED TO FLY TO MINNEAPOLIS MINNESOTA  
DES MOINES  
DES MOINES  
DEZ MOINNZ  
DES MOINES IOWA CHICAGO  
JUST ME  
FOURTH OF JULY  
SUNDAY  
I'D LIKE TO MAKE RESERVATIONS FOR A FLIGHT FOR TWO FROM SPOKANE  
TO ONTARIO AIRPORT IN CALIFORNIA

I WOULD LIKE TO LEAVE FRIDAY THE UM FIFTEENTH OF FEBRUARY AND  
RETURN ON MONDAY THE EIGHTEENTH OF FEBRUARY  
UH I'D LIKE TO FLY UH FROM SPOKANE TO PORTLAND  
TWO  
UH NEXT SATURDAY  
SATURDAY AFTER THAT  
UM I'D LIKE TO HAVE TWO PEOPLE FLY FORM SPOKANE TO ATLANTA  
TOMORROW  
FRIDAY  
I'D LIKE TICKETS FOR FOUR PEOPLE FROM SPOKANE TO LOS ANGELES  
NEXT MONDAY  
FEBRUARY SIXTEENTH  
I'D LIKE TO FLY FROM SEATTLE TO ORLANDO LEAVING FEBRUARY SIXTH  
COMING BACK ON MARCH SECOND AND I'D LIKE THAT FOR FOUR  
PASSENGERS  
UH I'D LIKE TO FLY FROM SPOKANE  
UM HOW ABOUT BOSTON  
SPOKANE  
ONE  
UH FEBRUARY TWELVETH  
FEBRUARY THIRTIETH  
I WOULD LIKE TO FLY FROM SPOKANE TO SEATTLE TOMORROW RETURNING  
ON FRIDAY  
I'D LIKE TO GET A TICKET FOR SEATTLE TO NEW YORK  
ONE PERSON  
JUNE FIFTH RETURNING AUGUST FIFTH  
I'D LIKE TO FLY  
I'D LIKE TO GO TO LAS VEGAS  
I WILL BE LEAVING FROM LOS ANGELES ON JUNE THIRD COMING BACK JUNE  
SIXTH  
ONE  
JUNE SIXTH  
LIKE TO GET A TICKET FROM HOUSTON TO SEATTLE  
HOUSTON  
ONE PERSON TRAVELING  
APRIL FIRST COMING BACK APRIL THIRTIETH  
I'D LIKE TO GET A TICKET FROM ORLANDO TO LAS VEGAS  
ONE ONE PERSON  
MAY FIRST RETURNING MAY TENTH  
I WOULD LIKE A TICKET FROM LOS ANGELES TO UH NORTH DAKOTA  
BISMARK ON FEBRUARY TWENTY SEVENTH COMING BACK ON MARCH  
THIRD  
BISMARK NORTH DAKOTA  
DETROIT MICHIGAN  
TWO

FEBRUARY TWENTY SIXTH  
I WOULD LIKE A TICKET FOR TWO PEOPLE FLYING ON FEBRUARY  
FOURTEENTH RETURNING BACK ON MARCH TENTH GOING FROM SEATTLE  
WASHINGTON TO PHILADELPHIA PENNSYLVANIA  
FEBRUARY FOURTEENTH  
CAN I GET A FLIGHT TO PORT ANGELES  
LIKE TO GET A FLIGHT TO LOS ANGELES  
LOS ANGELES  
SEATTLE ON MARCH THIRTEENTH RETURNING JUNE FIFTH  
TWENTY THREE  
I'D LIKE TO FLY TO HOUSTON FROM SEATTLE ON JUNE THIRD RETURNING  
JUNE SIXTH FOR ONE PERSON  
ONE PERSON  
I'D LIKE TO FLY TO VANCOUVER FROM VICTORIA  
BOSTON  
CLEVELAND  
THIRTY FIVE  
NOVEMBER TWENTY FIRST RETURNING NOVEMBER TWENTY FOURTH  
I'M LEAVING SEATTLE TO BOSTON  
ONE  
MARCH FIRST THROUGH MARCH THIRTEENTH  
DO YOU HAVE ANY FLIGHTS AVAILABLE FROM LOS ANGELES TO SEATTLE  
ON JUNE FIRST RETURNING JUNE THIRD  
ONE PERSON  
I WANT TO FLY TO SEATTLE  
I'M LEAVING FROM SPOKANE  
THEY'LL BE FIVE PEOPLE TRAVELING WITH ME  
TONIGHT I WANT TO TRAVEL RIGHT NOW  
I WANT TO RETURN TO SPOKANE TONIGHT TOO  
I WILL BE RETURNING TO SPOKANE ON THE TWENTY NINTH OF FEBRUARY  
I'D LIKE TO BOOK TRAVEL TO HAWAII PLEASE  
SEATTLE  
THERE WILL BE FIVE IN OUR PARTY  
WOULD LIKE TO RETURN ON APRIL FIRST AND RETURN ON APRIL FIFTEENTH  
YEAH I'D LIKE A TICKET TO UH NEW YORK  
OH UH BOISE  
AH THREE OF US  
WELL I'D LIKE TO LEAVE TOMORROW AND COME BACK ON THE TWENTY  
FIRST  
I'D LIKE TO FLY FROM SEATTLE TO SPOKANE ON THE FOURTEENTH AND  
RETURN MAY THIRD  
THREE ADULTS AND FIVE CHILDREN  
I'D LIKE TO FLY FROM PORTLAND MAINE TO PORTLAND OREGON ON UH THE  
ELEVENTH AND RETURN ON THE FOLLOWING THURSDAY  
AH TWO

UM I'D LIKE TO FLY FROM ANCHORAGE TO MAINE  
PORTLAND  
OH A COUPLE I GUESS  
TOMORROW  
THE THIRTIETH  
I'D LIKE TO FLY FROM HONOLULU TO SEATTLE  
MY BROTHER AND SISTER AND I  
THE THIRTY FIRST OF MARCH  
THE EIGHTH  
I NEED A TICKET  
I NEED TO BOOK A FLIGHT  
I WANT TO GO SOMEPLACE SUNNY  
MIAMI  
UH LET'S SAY ANCHORAGE WITH MY MOM AND I  
SOON AS POSSIBLE  
THE NEXT AVAILABLE FLIGHT  
THE NEXT FLIGHT  
THIS EVENING  
TODAY  
AH THE FOLLOWING THURSDAY NIGHT  
I'D LIKE TO FLY FROM PHOENIX TO MIAMI THIS MONDAY  
JUST MY SPOUSE AND MYSELF  
I'D LIKE TO RETURN TO PHOENIX ON THE FOLLOWING SUNDAY  
I'D LIKE TO FLY TO UH LAS VEGAS  
SPOKANE  
TWO ADULTS AND ONE CHILD  
WE'RE GOING TO LEAVE MARCH THIRTIETH  
UH WE'RE ACTUALLY GOING TO LEAVE ON MARCH TWENTY SECOND AND  
COME BACK MARCH THIRTIETH  
I WANNA BOOK A FLIGHT TO AUSTIN FROM BOSTON ON UH JANUARY FIRST  
NO ON JANUARY FIRST  
NO I'M FLYING FROM BOSTON TO AUSTIN ON JANUARY FIRST  
NO I'M RETURNING JANUARY THIRD  
I NEED TO BOOK A FLIGHT TO SEATTLE FOR MY WIFE AND I NEXT FOR NEXT  
TUESDAY  
SPOKANE  
NEXT TUESDAY  
UM THE FOLLOWING TUESDAY  
I'M RETURNING ON FEBRUARY TWENTIETH  
I NEED TO BOOK A FLIGHT  
SPOKANE  
ORLANDO  
FIVE I I MEAN THREE  
EASTER SUNDAY  
APRIL FOOLS DAY

## Appendix D Reference Files

What follows are hand transcriptions of the eighteen audio recordings used in this study. The audio files were recorded by Next It engineers in 2007 in signed 16bit PCM format at 8 kHz from calls made over both the public switched network and cell phones (Next It, 2008). An email message was sent to other Next It engineers asking them to dial a specific phone number and, in response to a voice prompt, request information from a virtual travel agent yet to be built. They were told that they were simply to speak, as if they were interacting with an LVCSR. Thus, Female 1 would have uttered “List all flights departing from Houston and landing in Phoenix directly after she said, “I want the fare and flight schedule from Spokane to Boston.” The calls transcribed below were made by technically sophisticated, college-educated adults, speaking Standard American English, and divided evenly between men and women. The duration of each utterance, for alignment purposes, is shown within parentheses.

Female 1

REF: (6.282,8.743) I NEED A PLANE FROM SPOKANE TO SEATTLE  
 REF: (20.029,21.826) OCTOBER THIRTIETH  
 REF: (31.540,34.892) ROUNDTrip I WANT A ROUNDTrip FROM SPOKANE TO SEATTLE  
 REF: (45.919,48.485) I WANT TO FLY FROM SPOKANE TO NEW YORK  
 REF: (60.031,63.739) I WANT THE FARE AND FLIGHT SCHEDULE FROM SPOKANE TO BOSTON  
 REF: (74.845,79.401) LIST ALL FLIGHTS DEPARTING FROM HUSTON[sic] AND LANDING IN PHOENIX

Female 2

REF: (3.203,5.553) GIVE ME A FLIGHT BETWEEN SPOKANE AND SEATTLE  
 REF: (15.633,18.307) { UM / @ } OCTOBER SEVENTEENTH  
 REF: (26.827,29.606) OH I NEED A PLANE FROM SPOKANE TO SEATTLE  
 REF: (43.337,46.682) I WANT A ROUNDTrip FROM MINNEAPOLIS TO  
 REF: (58.050,61.762) I WANT TO BOOK A TRIP FROM MISSOULA TO PORTLAND  
 REF: (73.397,77.215) I NEED A TICKET FROM ALBUQUERQUE TO NEW YORK

REF: (87.370,94.098) YEAH RIGHT { UM / @ } I NEED A TICKET FROM SPOKANE SEPTEMBER THIRTIETH TO SEATTLE RETURNING OCTOBER THIRD

REF: (107.381,113.593) I WANT TO GET FROM ALBUQUERQUE TO NEW ORLEANS ON OCTOBER THIRD TWO THOUSAND SEVEN

Female 3

REF: (7.960,10.580) I WANT A ROUNDTRIP FROM SPOKANE TO SEATTLE

REF: (21.409,27.399) I WANT TO GET FROM SPOKANE TO SEATTLE ON OCTOBER THIRD TWO THOUSAND AND SEVEN

REF: (39.331,41.757) OCTOBER TENTH TWO THOUSAND AND SEVEN

Female 4

REF: (5.257,10.068) I WANT TO GET FROM SEATTLE TO SAN FRANCISCO ON OCTOBER TENTH TWO THOUSAND SEVEN

REF: (14.575,17.116) I WANT TO TRAVEL FROM SPOKANE TO SEATTLE

REF: (29.801,33.548) A LIST ALL FLIGHTS DEPARTING FROM SEATTLE AND LANDING IN PHOENIX

REF: (45.366,47.652) I WANT TO FLY FROM SPOKANE TO SEATTLE

REF: (59.091,62.103) I NEED A PLANE FROM SEATTLE TO SAN FRANCISCO

REF: (72.014,75.109) I WANT A ROUNDTRIP FROM SEATTLE TO SPOKANE

Female 5

REF: (0.376,4.278) LIST ALL FLIGHTS DEPARTING FROM SEATTLE AND LANDING IN PHOENIX

REF: (16.586,19.967) PLEASE GIVE ME A RESERVATION FROM SEATTLE TO SPOKANE

REF: (32.747,35.874) SHOW ME THE AIRFARE FROM SPOKANE TO SEATTLE

REF: (48.157,52.314) ROUNDTRIP FARE INFORMATION FROM SAN FRANCISCO TO PHOENIX

Female 6

REF: (6.924,8.832) I WANT TO TAKE OFF FROM SPOKANE

REF: (20.762,23.657) { UM / @ } I WANT TO FLY ROUNDTRIP FROM SPOKANE TO SEATTLE

REF: (34.878,36.638) { UH / @ } FEBRUARY NINETEENTH

REF: (45.618,47.946) GIVE ME A FLIGHT BETWEEN SPOKANE AND L A

REF: (60.494,64.205) { UM / @ } I'D LIKE TO MAKE A RESERVATION TO FLY FROM SEATTLE TO SPOKANE

REF: (72.175,75.102) SHOW ME THE AIRFARE FOR SPOKANE TO L A

REF: (79.348,82.289) I WANNA TRAVEL FROM SPOKANE TO BOISE

Female 7

REF: (4.848,7.858) I NEED A FLIGHT FROM SPOKANE TO SEATTLE

REF: (20.859,24.091) I NEED A ROUNDTRIP TICKET FROM SEATTLE TO SPOKANE

REF: (35.868,40.525) { UM / @ } I NEED TO GET OUT OF SPOKANE AND GO TO PHOENIX

REF: (52.714,62.135) CAN I PURCHASE A TICKET FOR PORTLAND TO SEATTLE { UH / @ } LEAVING OCTOBER TENTH RETURNING ON OCTOBER EIGHTEENTH

REF: (75.802,81.484) I'D LIKE TO LEAVE FROM SPOKANE BUT I DON'T KNOW WHERE I WANT TO GO TO

REF: (94.147,97.812) CAN YOU TELL ME THE AIRFARES TO GET FROM SPOKANE TO SEATTLE



Female 8

REF: (6.895,10.273) PLEASE GIVE ME A FLIGHT BETWEEN SPOKANE AND SEATTLE  
REF: (19.295,21.099) DECEMBER TWENTY FIFTH  
REF: (22.936,24.319) FROM SPOKANE TO SEATTLE

Female 9

REF: (3.7,6.5) LIKE TO FLY FROM PORTLAND TO SPOKANE  
REF: (17.7,21.0) I WOULD LIKE TO FLY FROM SPOKANE TO SEATTLE

Male 1

REF: (7.301,11.097) I WANNA TRAVEL FROM SPOKANE TO SEATTLE  
REF: (23.902,27.596) I WANNA GET FROM SPOKANE TO SEATTLE ON OCTOBER  
TWELFTH  
REF: (42.947,44.830) I WANNA TAKE OFF FROM SEATTLE  
REF: (58.574,62.872) CAN YOU SHOW ME THE AIRFARE FROM SAN FRANCISCO TO  
SEATTLE

Male 2

REF: (6.770,8.985) SCHEDULE FROM SPOKANE TO BOSTON  
REF: (19.745,22.924) I WANNA TRAVEL FROM SPOKANE TO SEATTLE  
REF: (26.326,29.727) I WANT THE FARE AND FLIGHT SCHEDULE FROM SPOKANE  
TO BOSTON  
REF: (40.813,42.874) OCTOBER TWENTY FIRST  
REF: (49.544,53.045) ROUNDTrip FARE INFORMATION FROM SAN FRANCISCO TO  
SEATTLE  
REF: (63.706,67.288) ROUNDTrip INFORMATION FROM SAN FRANCISCO TO  
SEATTLE  
REF: (77.568,80.770) LIST ALL FLIGHTS DEPARTING FROM SEATTLE AND  
LANDING IN PHOENIX

Male 3

REF: (6.536,8.917) I WANT TO FLY FROM SPOKANE TO SEATTLE  
REF: (19.822,22.633) SHOW ME THE AIRFARE FROM SPOKANE TO SEATTLE  
REF: (33.353,37.132) LIST ALL FLIGHTS DEPARTING FROM SEATTLE AND  
LANDING IN PHOENIX  
REF: (48.332,50.299) I WANT TO TAKE OFF FROM SPOKANE  
REF: (61.652,65.104) ROUNDTrip FARE INFORMATION FROM SAN FRANCISCO TO  
SEATTLE

Male 4

REF: (7.536,10.399) SHOW ME THE AIRFARE FROM SPOKANE TO SEATTLE  
REF: (21.712,23.850) I WANT TO FLY FROM SPOKANE TO SEATTLE  
REF: (34.741,40.208) I WANT TO GET FROM SPOKANE TO SEATTLE ON OCTOBER  
THE THIRD OF TWO THOUSAND SEVEN  
REF: (51.046,57.332) I NEED A TICKET FROM SPOKANE SEPTEMBER THIRTIETH  
TO SEATTLE RETURNING OCTOBER THIRD  
REF: (70.249,72.250) I NEED A PLANE FROM SPOKANE TO SEATTLE

Male 5

REF: (3.729,5.738) I WANT TO TAKE OFF FROM SPOKANE  
REF: (9.780,12.928) PLEASE GIVE ME A RESERVATION FROM SEATTLE TO  
SPOKANE

REF: (23.886,29.459) I NEED A TICKET FROM SPOKANE SEPTEMBER THIRTIETH TO SEATTLE RETURNING OCTOBER THIRD

REF: (53.887,57.427) LIST ALL FLIGHTS DEPARTING FROM SEATTLE AND LANDING IN PHOENIX

REF: (68.801,70.190) ON SEPTEMBER THIRTIETH

Male 6

REF: (4.325,6.253) TO TAKE OFF FROM SEATTLE

REF: (13.580,16.715) SHOW ME THE AIRFARE FROM SPOKANE TO SEATTLE

REF: (28.234,31.672) I WANT TO FLY FROM SEATTLE TO ATLANTA

REF: (42.578,45.491) I WANT A ROUNDTRIP FROM SPOKANE TO ATLANTA

REF: (60.164,63.530) I WANT THE FARE AND FLIGHT SCHEDULE FROM SPOKANE TO WALLA WALLA

REF: (74.605,78.620) I WANT THE FARE AND FLIGHT SCHEDULE FROM SPOKANE TO WALLA WALLA

Male 7

REF: (1.0,4.5) I WOULD LIKE TO FLY FROM SPOKANE TO SEATTLE

Male 8

REF: (2.7,5.1) I NEED A PLANE FROM SPOKANE TO SEATTLE

REF: (16.0,21.5) I NEED A TICKET FROM SPOKANE SEPTEMBER THIRTIETH TO SEATTLE RETURNING OCTOBER THIRD

REF: (33.6,36.5) I WANT TO TRAVEL FROM SPOKANE TO SEATTLE

REF: (48.0,52.5) ROUNDTRIP FARE INFORMATION FROM SAN FRANCISCO TO SEATTLE

REF: (59.5,63.0) I WANT TO FLY FROM SAN FRANCISCO TO SEATTLE

REF: (76.5,79.2) I WANT TO FLY FROM SPOKANE TO BOSTON

REF: (89.4,91.5) I WANT TO TAKE OFF FROM SPOKANE

Male 9

REF: (1.2,4.0) I WANT TO FLY FROM SPOKANE TO SEATTLE

REF: (5.3,8.6) I WOULD LIKE TO FLY FROM SEATTLE TO SAN FRANCISCO

### Appendix E Four Equivalence Classes Used in Experiment 3

BE	
Word Map	Syllable Map
Be	b_iy
goin' to be	g_ow ax_n b_iy
Going to be	g_ow ix_ng t_uw b_iy
I am	ay ae_m
I have	ay hh_ae_v
I'll be	ay_l b_iy
I'll have	ay_l hh_ae_v
I'm	ay_m
I'm going	ay_m g_ow ix_ng
I'll	ay_l
Is	ih_z
It's	ih_ts
It's goin' be like	ih_ts g_ow ax_n b_iy
it'll be	ih dx_ax_l b_iy
There are	dh_eh_r aa_r
There going to be	dh_eh_r g_ow ix_ng t_uw b_iy
There should be	dh_eh_r sh_uh_dd b_iy
There will be	dh_eh_r w_ih_l b_iy
There'll be	dh_eh_r ax_l b_iy
There's	dh_eh_r_z
there's	dh_eh_r_z
there's going to be	dh_eh_r_z g_ow ix_ng t_uw b_iy
they'll	dh_ey_l b_iy
We have	w_iy hh_ae_v
We will be	w_iy w_ih_l b_iy
We'll be	w_iy l b_iy
we'll	w_iy_l
Will be	w_ih_l b_iy
Will have	w_ih_l hh_ae_v

GO	
Word Map	Syllable Map
A flight	ax f_l ay_td
A ticket	ax t_ih k_ax_td
book airline travel	b_uh_kd eh_r l_ay_n t_r ae v_ax_l
book reservations	b_uh_kd r_eh s_axr v_ey sh_ax_n_z
Create a reservation	k_r_ey ey_td ax_r_eh z_axr v_ey sh_ax_n
Departing	d_ax p_aa_r dx_ix_ng
Fly	f_l_ay
Flying	f_l_ay ix_ng
Get	g_eh_td
I am leaving	ay ae_m l_iy v_ix_ng
I will be leaving	ay w ih l b iy l iy v ix ng
I'm flying	ay_m f_l_ay ix_ng
I'm leaving	ay_m l iy v ix ng
I'm traveling	ay_m t_r ae v_ax l_ix_ng
Is travelling	ih z t_r ae v_ax l_ix_ng
Leave	l_iy_v
Leaving	l_iy v_ix_ng
Make reservations	m_ey kd r_eh z_axr v_ey sh_ax_n_z
Return	r_ax t_er_n
schedule a flight	s_k_eh jh_uh_l ax f_l_ay_td
Ticket	t_ih k_ax_td
Ticket	t_ih k_ax_td
ticket leaving	t_ih k_ax_td l_iy v_ix_ng
Tickets	t_ih k_ax_ts
To book a flight	t_uw b_uh_kd ax f_l_ay_td
To book a ticket	b_uh kd ax t_ih k_ax_td
To book some travel	b_uh kd s_ah m t_r ae v_ax_l
To buy a ticket	t_uw b_ay
To buy an airline ticket	t_uw b_ay ax eh_r l_ay_n t_ih k_ax_td
To depart	t_uw d_ax p_aa_r td
To fly	t_uw f_l_ay
To get	t_uw g_eh_td
To get a flight	t_uw g_eh_td ax f_l_ay_td
To get a ticket	t_uw g_eh_td ax t_ih k_ax_td
To go	t_uw g_ow

To leave	t uw l iy v
To plan travel	t uw p l ae n t r ae v ax l
To purchase a ticket	t uw p er ch ax s ax t ih k ax td
To schedule a flight	t uw s k eh jh uh l ax f l ay td
To travel	t uw t r ae v ax l
Travelling	t r ae v ax l ix ng
We'll leave	w iy l l iy v
We're leaving	w iy r l iy v ix ng
Will be flying	w ih l b iy f l ay ix ng

RETURN	
Word Map	Syllable Map
come back	k ah m b ae k
coming back	k ah m ix ng b ae kd
I'll return	ay l r ax t er n
Return	r ax t er n
returning	r ax t er n ix ng
to come back	t uw k ah m b ae k
to return	t ux r ax t er n

WANT	
Word Map	Syllable Map
Buy	b ay
can i	k ae n ay
can I have	k ae n ay hh ae v
could I	g eh td
could I get	k uh dd ay g eh td
I like	ay l ay kd
I need	ay n iy dd
I need	ay n iy dd
I wanna	ay w aa n ax
I want	ay w aa n td
I would like	ay w uh dd l ay kd
I'd like	ay dd l ay kd
I'd like to have	ay dd l ay kd t uw hh ae v

I'm planning on	ay dd l ay kd t uw hh ae v
looking for	l uh k ix ng f ao r
Need	n iy dd
Wanna	w aa n ax
Want	w aa n td
We need	w iy n iy dd
We want	w iy w aa n td
We would like	w iy w uh dd l ay kd
We'd like	w iy dd l ay kd
We'll need	w iy l n iy dd
Would like	w uh dd l ay kd

## Appendix F Results, Experiments 2.1 and 2.2

The data summarized in Section 4 comes from the columns labeled *Sub*, *Del*, *Ins*, *Err*, and the rows labeled *Mean*, *S.D.*, and *Median*.

### F.1 Word Language Model: Unigrams

../Output/Wer/multiplewordsAllhyp.trn										
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err		
female1	9	49	46.9	22.4	30.6	2.0	55.1	66.7		
female2	30	77	53.2	10.4	36.4	27.3	74.0	56.7		
female3	7	29	41.4	20.7	37.9	24.1	82.8	71.4		
female4	7	59	49.2	33.9	16.9	5.1	55.9	100.0		
female5	6	35	60.0	37.1	2.9	8.6	48.6	66.7		
female6	14	58	36.2	34.5	29.3	8.6	72.4	57.1		
female7	11	74	40.5	17.6	41.9	23.0	82.4	90.9		
female8	6	16	68.8	18.8	12.5	12.5	43.8	66.7		
female9	6	16	93.8	0.0	6.3	12.5	18.8	50.0		
male1	5	34	50.0	32.4	17.6	0.0	50.0	80.0		
male2	11	51	54.9	41.2	3.9	3.9	49.0	63.6		
male3	7	41	53.7	24.4	22.0	7.3	53.7	71.4		
male4	12	53	30.2	13.2	56.6	28.3	98.1	91.7		
male5	11	42	45.2	26.2	28.6	11.9	66.7	45.5		
male6	9	53	37.7	43.4	18.9	0.0	62.3	66.7		
male7	3	9	77.8	11.1	11.1	11.1	33.3	33.3		
male8	18	61	72.1	19.7	8.2	4.9	32.8	38.9		
male9	2	18	72.2	16.7	11.1	0.0	27.8	100.0		
Sum/Avg	174	775	50.2	24.9	24.9	11.6	61.4	64.4		
Mean	9.7	43.1	54.7	23.5	21.8	10.6	56.0	67.6		
S.D.	6.4	20.0	16.6	11.7	14.8	9.3	21.0	19.5		
Median	8.0	45.5	51.6	21.6	18.3	8.6	54.4	66.7		

## F.2 Word Language Model: Bigrams

../Output/Wer/multiplewordsAllhyp.trn									
SPKR	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	53.1	26.5	20.4	2.0	49.0	66.7	
female2	30	77	57.1	10.4	32.5	32.5	75.3	56.7	
female3	7	29	48.3	17.2	34.5	24.1	75.9	71.4	
female4	7	59	69.5	22.0	8.5	5.1	35.6	100.0	
female5	6	35	57.1	37.1	5.7	5.7	48.6	66.7	
female6	14	58	51.7	25.9	22.4	10.3	58.6	57.1	
female7	11	74	54.1	10.8	35.1	28.4	74.3	90.9	
female8	6	16	87.5	6.3	6.3	6.3	18.8	33.3	
female9	6	16	93.8	0.0	6.3	18.8	25.0	66.7	
male1	5	34	55.9	29.4	14.7	2.9	47.1	80.0	
male2	11	51	60.8	35.3	3.9	7.8	47.1	54.5	
male3	7	41	65.9	22.0	12.2	7.3	41.5	57.1	
male4	12	53	47.2	7.5	45.3	30.2	83.0	100.0	
male5	11	42	54.8	23.8	21.4	11.9	57.1	54.5	
male6	9	53	43.4	41.5	15.1	0.0	56.6	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	86.9	11.5	1.6	6.6	19.7	22.2	
male9	2	18	88.9	5.6	5.6	0.0	11.1	50.0	
Sum/Avg	174	775	60.6	20.3	19.1	13.3	52.6	61.5	
Mean	9.7	43.1	65.3	18.5	16.2	11.7	46.4	62.7	
S.D.	6.4	20.0	17.9	12.7	13.3	10.5	22.7	21.4	
Median	8.0	45.5	57.1	19.6	13.5	7.6	47.8	61.9	



### F.3 Word Language Model: Trigrams

../Output/Wer/multiplewordsAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	53.1	28.6	18.4	2.0	49.0	66.7	
female2	30	77	57.1	10.4	32.5	31.2	74.0	56.7	
female3	7	29	51.7	13.8	34.5	27.6	75.9	71.4	
female4	7	59	71.2	22.0	6.8	5.1	33.9	85.7	
female5	6	35	60.0	40.0	0.0	8.6	48.6	66.7	
female6	14	58	51.7	27.6	20.7	10.3	58.6	57.1	
female7	11	74	50.0	14.9	35.1	28.4	78.4	90.9	
female8	6	16	87.5	6.3	6.3	12.5	25.0	33.3	
female9	6	16	100.0	0.0	0.0	18.8	18.8	50.0	
male1	5	34	58.8	29.4	11.8	2.9	44.1	80.0	
male2	11	51	56.9	37.3	5.9	7.8	51.0	54.5	
male3	7	41	65.9	24.4	9.8	7.3	41.5	57.1	
male4	12	53	45.3	9.4	45.3	32.1	86.8	91.7	
male5	11	42	50.0	23.8	26.2	11.9	61.9	45.5	
male6	9	53	47.2	37.7	15.1	0.0	52.8	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	86.9	11.5	1.6	6.6	19.7	22.2	
male9	2	18	88.9	5.6	5.6	0.0	11.1	50.0	
Sum/Avg	174	775	60.5	21.0	18.5	13.7	53.2	59.2	
Mean	9.7	43.1	65.7	19.0	15.3	12.5	46.8	60.0	
S.D.	6.4	20.0	18.6	12.8	14.1	10.6	23.5	19.7	
Median	8.0	45.5	58.0	18.4	10.8	9.5	48.8	57.1	

#### F.4 Word Language Model: Quadrigrams

../Output/Wer/multiplewordsAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	51.0	30.6	18.4	2.0	51.0	66.7	
female2	30	77	57.1	10.4	32.5	31.2	74.0	56.7	
female3	7	29	48.3	17.2	34.5	27.6	79.3	71.4	
female4	7	59	69.5	22.0	8.5	5.1	35.6	100.0	
female5	6	35	62.9	37.1	0.0	8.6	45.7	66.7	
female6	14	58	51.7	27.6	20.7	8.6	56.9	57.1	
female7	11	74	51.4	13.5	35.1	31.1	79.7	90.9	
female8	6	16	87.5	6.3	6.3	12.5	25.0	33.3	
female9	6	16	100.0	0.0	0.0	18.8	18.8	50.0	
male1	5	34	55.9	29.4	14.7	2.9	47.1	80.0	
male2	11	51	58.8	37.3	3.9	7.8	49.0	54.5	
male3	7	41	65.9	24.4	9.8	7.3	41.5	57.1	
male4	12	53	45.3	9.4	45.3	32.1	86.8	91.7	
male5	11	42	54.8	23.8	21.4	11.9	57.1	45.5	
male6	9	53	49.1	37.7	13.2	0.0	50.9	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	86.9	11.5	1.6	6.6	19.7	22.2	
male9	2	18	88.9	5.6	5.6	0.0	11.1	50.0	
Sum/Avg	174	775	60.8	21.0	18.2	13.8	53.0	59.8	
Mean	9.7	43.1	65.8	19.1	15.1	12.5	46.7	60.8	
S.D.	6.4	20.0	18.5	12.6	13.9	10.9	23.6	21.0	
Median	8.0	45.5	58.0	19.6	11.5	8.6	48.0	57.1	

### F.5 Syllable Language Model, Word Reference File: Unigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	36.7	59.2	4.1	89.8	153.1	77.8	
female2	30	77	31.2	51.9	16.9	115.6	184.4	56.7	
female3	7	29	24.1	51.7	24.1	96.6	172.4	71.4	
female4	7	59	40.7	54.2	5.1	74.6	133.9	100.0	
female5	6	35	37.1	62.9	0.0	162.9	225.7	66.7	
female6	14	58	20.7	69.0	10.3	72.4	151.7	57.1	
female7	11	74	23.0	59.5	17.6	98.6	175.7	90.9	
female8	6	16	18.8	81.3	0.0	118.8	200.0	66.7	
female9	6	16	75.0	18.8	6.3	43.8	68.8	66.7	
male1	5	34	26.5	70.6	2.9	102.9	176.5	80.0	
male2	11	51	39.2	60.8	0.0	121.6	182.4	63.6	
male3	7	41	36.6	58.5	4.9	90.2	153.7	71.4	
male4	12	53	24.5	20.8	54.7	83.0	158.5	91.7	
male5	11	42	26.2	69.0	4.8	109.5	183.3	54.5	
male6	9	53	30.2	60.4	9.4	67.9	137.7	66.7	
male7	3	9	66.7	22.2	11.1	44.4	77.8	33.3	
male8	18	61	50.8	45.9	3.3	77.0	126.2	38.9	
male9	2	18	61.1	33.3	5.6	77.8	116.7	100.0	
Sum/Avg	174	775	33.8	54.8	11.4	93.9	160.1	66.1	
Mean	9.7	43.1	37.2	52.8	10.1	91.5	154.4	69.7	
S.D.	6.4	20.0	16.3	18.1	12.9	28.7	39.9	18.5	
Median	8.0	45.5	33.9	58.9	5.3	90.0	156.1	66.7	

**F.6 Syllable Language Model, Word Reference File: Bigrams**

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	49.0	38.8	12.2	16.3	67.3	77.8	
female2	30	77	57.1	16.9	26.0	41.6	84.4	56.7	
female3	7	29	51.7	13.8	34.5	27.6	75.9	71.4	
female4	7	59	67.8	30.5	1.7	27.1	59.3	100.0	
female5	6	35	48.6	51.4	0.0	97.1	148.6	66.7	
female6	14	58	44.8	46.6	8.6	31.0	86.2	57.1	
female7	11	74	48.6	20.3	31.1	50.0	101.4	90.9	
female8	6	16	75.0	25.0	0.0	6.3	31.3	50.0	
female9	6	16	93.8	0.0	6.3	12.5	18.8	50.0	
male1	5	34	50.0	38.2	11.8	26.5	76.5	80.0	
male2	11	51	62.7	33.3	3.9	74.5	111.8	54.5	
male3	7	41	58.5	39.0	2.4	24.4	65.9	57.1	
male4	12	53	49.1	15.1	35.8	62.3	113.2	100.0	
male5	11	42	50.0	38.1	11.9	26.2	76.2	54.5	
male6	9	53	43.4	50.9	5.7	37.7	94.3	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	83.6	14.8	1.6	18.0	34.4	27.8	
male9	2	18	83.3	11.1	5.6	0.0	16.7	50.0	
Sum/Avg	174	775	57.7	29.2	13.2	37.3	79.6	62.6	
Mean	9.7	43.1	62.1	26.9	11.1	32.8	70.7	63.6	
S.D.	6.4	20.0	17.7	16.2	12.2	24.9	37.4	20.3	
Median	8.0	45.5	54.4	27.8	6.0	26.8	76.0	57.1	

### F.7 Syllable Language Model, Word Reference File: Trigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	51.0	32.7	16.3	24.5	73.5	77.8	
female2	30	77	55.8	18.2	26.0	33.8	77.9	56.7	
female3	7	29	58.6	6.9	34.5	34.5	75.9	71.4	
female4	7	59	64.4	33.9	1.7	22.0	57.6	100.0	
female5	6	35	51.4	48.6	0.0	97.1	145.7	66.7	
female6	14	58	50.0	37.9	12.1	27.6	77.6	57.1	
female7	11	74	45.9	23.0	31.1	41.9	95.9	90.9	
female8	6	16	68.8	25.0	6.3	12.5	43.8	50.0	
female9	6	16	100.0	0.0	0.0	18.8	18.8	50.0	
male1	5	34	52.9	38.2	8.8	26.5	73.5	80.0	
male2	11	51	60.8	33.3	5.9	74.5	113.7	54.5	
male3	7	41	58.5	39.0	2.4	41.5	82.9	57.1	
male4	12	53	49.1	17.0	34.0	50.9	101.9	91.7	
male5	11	42	57.1	31.0	11.9	19.0	61.9	45.5	
male6	9	53	43.4	50.9	5.7	28.3	84.9	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	88.5	11.5	0.0	13.1	24.6	22.2	
male9	2	18	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	775	59.1	27.6	13.3	34.8	75.7	60.3	
Mean	9.7	43.1	64.2	24.8	10.9	32.1	67.8	59.5	
S.D.	6.4	20.0	19.3	16.3	12.3	23.5	37.4	25.1	
Median	8.0	45.5	57.8	28.0	6.1	27.0	74.7	57.1	

### F.8 Syllable Language Model, Word Reference File: Quadrigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	49	51.0	32.7	16.3	24.5	73.5	77.8	
female2	30	77	57.1	16.9	26.0	32.5	75.3	56.7	
female3	7	29	58.6	6.9	34.5	34.5	75.9	71.4	
female4	7	59	64.4	33.9	1.7	22.0	57.6	100.0	
female5	6	35	48.6	51.4	0.0	97.1	148.6	66.7	
female6	14	58	50.0	37.9	12.1	29.3	79.3	57.1	
female7	11	74	43.2	25.7	31.1	41.9	98.6	90.9	
female8	6	16	81.3	12.5	6.3	31.3	50.0	33.3	
female9	6	16	100.0	0.0	0.0	18.8	18.8	50.0	
male1	5	34	55.9	38.2	5.9	26.5	70.6	80.0	
male2	11	51	58.8	35.3	5.9	66.7	107.8	54.5	
male3	7	41	58.5	39.0	2.4	41.5	82.9	57.1	
male4	12	53	49.1	17.0	34.0	52.8	103.8	91.7	
male5	11	42	59.5	28.6	11.9	26.2	66.7	45.5	
male6	9	53	47.2	45.3	7.5	17.0	69.8	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	61	88.5	11.5	0.0	13.1	24.6	22.2	
male9	2	18	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	775	59.5	27.2	13.3	34.5	75.0	59.8	
Mean	9.7	43.1	65.1	24.0	10.9	32.6	67.5	58.6	
S.D.	6.4	20.0	19.6	16.4	12.3	22.4	37.1	25.8	
Median	8.0	45.5	58.6	27.1	6.1	27.9	72.0	57.1	

### F.9 Syllable Language Model, Syllable Reference File: Unigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	70	38.6	37.1	24.3	1.4	62.9	77.8	
female2	30	121	35.5	20.7	43.8	24.8	89.3	56.7	
female3	7	44	38.6	25.0	36.4	20.5	81.8	71.4	
female4	7	88	48.9	30.7	20.5	6.8	58.0	100.0	
female5	6	58	43.1	44.8	12.1	1.7	58.6	66.7	
female6	14	81	27.2	40.7	32.1	6.2	79.0	57.1	
female7	11	102	30.4	27.5	42.2	22.5	92.2	90.9	
female8	6	26	26.9	42.3	30.8	3.8	76.9	66.7	
female9	6	21	76.2	14.3	9.5	9.5	33.3	66.7	
male1	5	53	34.0	45.3	20.8	0.0	66.0	80.0	
male2	11	87	48.3	39.1	12.6	0.0	51.7	63.6	
male3	7	63	39.7	36.5	23.8	1.6	61.9	71.4	
male4	12	82	22.0	11.0	67.1	26.8	104.9	91.7	
male5	11	71	28.2	39.4	32.4	15.5	87.3	54.5	
male6	9	75	38.7	32.0	29.3	0.0	61.3	66.7	
male7	3	12	66.7	16.7	16.7	8.3	41.7	33.3	
male8	18	95	60.0	30.5	9.5	0.0	40.0	38.9	
male9	2	25	72.0	20.0	8.0	0.0	28.0	100.0	
Sum/Avg	174	1174	39.7	31.3	29.0	9.6	69.9	66.1	
Mean	9.7	65.2	43.0	30.8	26.2	8.3	65.3	69.7	
S.D.	6.4	30.3	16.1	10.8	15.1	9.5	21.5	18.5	
Median	8.0	70.5	38.7	31.3	24.0	5.0	62.4	66.7	

**F.10 Syllable Language Model, Syllable Reference File: Bigrams**

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	70	57.1	22.9	20.0	5.7	48.6	77.8	
female2	30	121	61.2	5.0	33.9	32.2	71.1	56.7	
female3	7	44	63.6	4.5	31.8	22.7	59.1	71.4	
female4	7	88	71.6	19.3	9.1	5.7	34.1	100.0	
female5	6	58	62.1	31.0	6.9	0.0	37.9	66.7	
female6	14	81	53.1	24.7	22.2	7.4	54.3	57.1	
female7	11	102	54.9	10.8	34.3	24.5	69.6	81.8	
female8	6	26	80.8	15.4	3.8	7.7	26.9	50.0	
female9	6	21	95.2	0.0	4.8	9.5	14.3	50.0	
male1	5	53	62.3	24.5	13.2	1.9	39.6	80.0	
male2	11	87	74.7	18.4	6.9	2.3	27.6	45.5	
male3	7	63	69.8	15.9	14.3	0.0	30.2	57.1	
male4	12	82	42.7	7.3	50.0	30.5	87.8	100.0	
male5	11	71	57.7	16.9	25.4	15.5	57.7	54.5	
male6	9	75	52.0	28.0	20.0	0.0	48.0	66.7	
male7	3	12	100.0	0.0	0.0	8.3	8.3	33.3	
male8	18	95	89.5	8.4	2.1	1.1	11.6	27.8	
male9	2	25	88.0	8.0	4.0	0.0	12.0	50.0	
Sum/Avg	174	1174	64.5	15.5	20.0	11.4	46.9	61.5	
Mean	9.7	65.2	68.7	14.5	16.8	9.7	41.0	62.6	
S.D.	6.4	30.3	16.3	9.5	13.9	10.8	22.8	20.0	
Median	8.0	70.5	63.0	15.6	13.7	6.6	38.8	57.1	



### F.11 Syllable Language Model, Syllable Reference File: Trigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	70	61.4	18.6	20.0	4.3	42.9	77.8	
female2	30	121	62.0	5.0	33.1	31.4	69.4	56.7	
female3	7	44	70.5	2.3	27.3	20.5	50.0	57.1	
female4	7	88	72.7	17.0	10.2	5.7	33.0	100.0	
female5	6	58	67.2	29.3	3.4	0.0	32.8	66.7	
female6	14	81	56.8	27.2	16.0	7.4	50.6	50.0	
female7	11	102	52.9	12.7	34.3	27.5	74.5	81.8	
female8	6	26	76.9	15.4	7.7	11.5	34.6	50.0	
female9	6	21	100.0	0.0	0.0	14.3	14.3	50.0	
male1	5	53	62.3	20.8	17.0	3.8	41.5	80.0	
male2	11	87	71.3	20.7	8.0	2.3	31.0	45.5	
male3	7	63	68.3	20.6	11.1	0.0	31.7	57.1	
male4	12	82	43.9	7.3	48.8	28.0	84.1	91.7	
male5	11	71	63.4	12.7	23.9	15.5	52.1	45.5	
male6	9	75	50.7	29.3	20.0	0.0	49.3	66.7	
male7	3	12	100.0	0.0	0.0	8.3	8.3	33.3	
male8	18	95	92.6	5.3	2.1	1.1	8.4	22.2	
male9	2	25	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	1174	66.0	14.9	19.1	11.5	45.5	58.0	
Mean	9.7	65.2	70.7	13.6	15.7	10.1	39.4	57.3	
S.D.	6.4	30.3	17.2	10.1	13.8	10.5	23.0	24.5	
Median	8.0	70.5	67.7	14.1	13.6	6.5	38.1	56.9	

### F.12 Syllable Language Model, Syllable Reference File: Quadrigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	70	61.4	18.6	20.0	4.3	42.9	77.8	
female2	30	121	62.8	4.1	33.1	29.8	66.9	56.7	
female3	7	44	70.5	2.3	27.3	27.3	56.8	57.1	
female4	7	88	72.7	17.0	10.2	5.7	33.0	100.0	
female5	6	58	65.5	31.0	3.4	0.0	34.5	66.7	
female6	14	81	56.8	27.2	16.0	7.4	50.6	50.0	
female7	11	102	51.0	13.7	35.3	27.5	76.5	81.8	
female8	6	26	84.6	7.7	7.7	7.7	23.1	33.3	
female9	6	21	100.0	0.0	0.0	14.3	14.3	50.0	
male1	5	53	64.2	20.8	15.1	3.8	39.6	80.0	
male2	11	87	69.0	23.0	8.0	2.3	33.3	45.5	
male3	7	63	68.3	20.6	11.1	0.0	31.7	57.1	
male4	12	82	42.7	8.5	48.8	29.3	86.6	91.7	
male5	11	71	64.8	12.7	22.5	15.5	50.7	45.5	
male6	9	75	54.7	26.7	18.7	0.0	45.3	66.7	
male7	3	12	100.0	0.0	0.0	8.3	8.3	33.3	
male8	18	95	92.6	5.3	2.1	1.1	8.4	22.2	
male9	2	25	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	1174	66.2	14.9	18.9	11.6	45.4	57.5	
Mean	9.7	65.2	71.2	13.3	15.5	10.2	39.0	56.4	
S.D.	6.4	30.3	17.4	10.2	13.8	11.0	23.6	25.1	
Median	8.0	70.5	66.9	13.2	13.1	6.5	37.1	56.9	

## Appendix G Output of Sclite for Experiment 3.1

### G.1 Unigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	41.3	38.1	20.6	4.8	63.5	77.8	
female2	30	107	36.4	23.4	40.2	27.1	90.7	56.7	
female3	7	41	41.5	26.8	31.7	19.5	78.0	71.4	
female4	7	77	48.1	35.1	16.9	13.0	64.9	100.0	
female5	6	56	42.9	44.6	12.5	5.4	62.5	66.7	
female6	14	73	28.8	43.8	27.4	9.6	80.8	57.1	
female7	11	85	32.9	31.8	35.3	29.4	96.5	90.9	
female8	6	25	24.0	48.0	28.0	4.0	80.0	66.7	
female9	6	17	82.4	11.8	5.9	17.6	35.3	66.7	
male1	5	47	40.4	44.7	14.9	4.3	63.8	80.0	
male2	11	82	48.8	40.2	11.0	1.2	52.4	63.6	
male3	7	58	39.7	41.4	19.0	1.7	62.1	71.4	
male4	12	72	25.0	12.5	62.5	30.6	105.6	91.7	
male5	11	63	28.6	42.9	28.6	20.6	92.1	54.5	
male6	9	70	40.0	32.9	27.1	2.9	62.9	66.7	
male7	3	9	66.7	22.2	11.1	11.1	44.4	33.3	
male8	18	81	60.5	30.9	8.6	8.6	48.1	38.9	
male9	2	20	80.0	20.0	0.0	5.0	25.0	100.0	
Sum/Avg	174	1046	41.0	33.7	25.2	13.3	72.3	66.1	
Mean	9.7	58.1	44.9	32.8	22.3	12.0	67.1	69.7	
S.D.	6.4	26.8	17.2	11.3	14.8	9.8	21.6	18.5	
Median	8.0	63.0	40.8	34.0	19.8	9.1	63.7	66.7	

## G.2 Bigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	58.7	22.2	19.0	6.3	47.6	77.8	
female2	30	107	60.7	6.5	32.7	30.8	70.1	56.7	
female3	7	41	65.9	4.9	29.3	22.0	56.1	71.4	
female4	7	77	74.0	19.5	6.5	7.8	33.8	85.7	
female5	6	56	62.5	30.4	7.1	3.6	41.1	66.7	
female6	14	73	53.4	27.4	19.2	8.2	54.8	57.1	
female7	11	85	56.5	10.6	32.9	22.4	65.9	81.8	
female8	6	25	80.0	16.0	4.0	8.0	28.0	50.0	
female9	6	17	100.0	0.0	0.0	11.8	11.8	33.3	
male1	5	47	70.2	21.3	8.5	4.3	34.0	80.0	
male2	11	82	73.2	19.5	7.3	3.7	30.5	45.5	
male3	7	58	67.2	17.2	15.5	3.4	36.2	71.4	
male4	12	72	38.9	8.3	52.8	31.9	93.1	100.0	
male5	11	63	57.1	19.0	23.8	15.9	58.7	54.5	
male6	9	70	52.9	27.1	20.0	0.0	47.1	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	87.7	9.9	2.5	1.2	13.6	27.8	
male9	2	20	95.0	5.0	0.0	5.0	10.0	50.0	
Sum/Avg	174	1046	64.7	16.3	19.0	12.0	47.3	60.9	
Mean	9.7	58.1	69.7	14.7	15.6	11.0	41.3	61.7	
S.D.	6.4	26.8	17.2	9.4	14.5	9.8	22.8	19.8	
Median	8.0	63.0	66.5	16.6	12.0	7.9	38.6	61.9	

### G.3 Trigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	60.3	20.6	19.0	7.9	47.6	77.8	
female2	30	107	62.6	5.6	31.8	29.9	67.3	56.7	
female3	7	41	73.2	2.4	24.4	19.5	46.3	57.1	
female4	7	77	76.6	15.6	7.8	9.1	32.5	85.7	
female5	6	56	66.1	30.4	3.6	0.0	33.9	66.7	
female6	14	73	54.8	30.1	15.1	8.2	53.4	50.0	
female7	11	85	56.5	12.9	30.6	27.1	70.6	81.8	
female8	6	25	76.0	16.0	8.0	12.0	36.0	50.0	
female9	6	17	94.1	5.9	0.0	11.8	17.6	50.0	
male1	5	47	63.8	25.5	10.6	4.3	40.4	80.0	
male2	11	82	68.3	23.2	8.5	3.7	35.4	45.5	
male3	7	58	65.5	22.4	12.1	3.4	37.9	71.4	
male4	12	72	41.7	8.3	50.0	29.2	87.5	91.7	
male5	11	63	61.9	12.7	25.4	15.9	54.0	45.5	
male6	9	70	51.4	28.6	20.0	0.0	48.6	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	91.4	6.2	2.5	1.2	9.9	22.2	
male9	2	20	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	1046	65.6	16.3	18.2	12.0	46.5	58.0	
Mean	9.7	58.1	70.2	14.8	15.0	10.8	40.6	57.3	
S.D.	6.4	26.8	16.8	10.4	13.5	9.9	22.3	23.6	
Median	8.0	63.0	65.8	14.3	11.4	8.7	39.2	56.9	

## G.4 Quadrigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	60.3	20.6	19.0	7.9	47.6	77.8	
female2	30	107	63.6	4.7	31.8	28.0	64.5	56.7	
female3	7	41	73.2	2.4	24.4	26.8	53.7	57.1	
female4	7	77	76.6	15.6	7.8	9.1	32.5	85.7	
female5	6	56	64.3	32.1	3.6	0.0	35.7	66.7	
female6	14	73	54.8	30.1	15.1	8.2	53.4	50.0	
female7	11	85	54.1	14.1	31.8	27.1	72.9	81.8	
female8	6	25	84.0	8.0	8.0	8.0	24.0	33.3	
female9	6	17	94.1	5.9	0.0	11.8	17.6	50.0	
male1	5	47	66.0	25.5	8.5	4.3	38.3	80.0	
male2	11	82	67.1	24.4	8.5	3.7	36.6	45.5	
male3	7	58	65.5	22.4	12.1	3.4	37.9	71.4	
male4	12	72	40.3	8.3	51.4	31.9	91.7	91.7	
male5	11	63	63.5	12.7	23.8	15.9	52.4	45.5	
male6	9	70	54.3	25.7	20.0	0.0	45.7	55.6	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	91.4	6.2	2.5	1.2	9.9	22.2	
male9	2	20	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	1046	65.8	16.1	18.2	12.2	46.5	56.9	
Mean	9.7	58.1	70.7	14.4	14.9	11.0	40.3	55.8	
S.D.	6.4	26.8	17.1	10.5	13.8	10.6	23.2	24.0	
Median	8.0	63.0	65.7	13.4	10.3	8.1	38.1	56.1	

## Appendix H Output of Sclite for Experiment 3.2

### H.1 Unigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	39.7	38.1	22.2	4.8	65.1	77.8	
female2	30	107	33.6	24.3	42.1	27.1	93.5	56.7	
female3	7	41	43.9	24.4	31.7	22.0	78.0	71.4	
female4	7	77	48.1	36.4	15.6	10.4	62.3	100.0	
female5	6	56	42.9	46.4	10.7	5.4	62.5	66.7	
female6	14	73	28.8	46.6	24.7	8.2	79.5	57.1	
female7	11	85	32.9	30.6	36.5	30.6	97.6	90.9	
female8	6	25	28.0	48.0	24.0	4.0	76.0	66.7	
female9	6	17	82.4	11.8	5.9	17.6	35.3	66.7	
male1	5	47	42.6	40.4	17.0	2.1	59.6	80.0	
male2	11	82	48.8	40.2	11.0	1.2	52.4	63.6	
male3	7	58	37.9	41.4	20.7	3.4	65.5	71.4	
male4	12	72	29.2	11.1	59.7	31.9	102.8	91.7	
male5	11	63	30.2	39.7	30.2	19.0	88.9	54.5	
male6	9	70	42.9	32.9	24.3	4.3	61.4	66.7	
male7	3	9	66.7	22.2	11.1	11.1	44.4	33.3	
male8	18	81	55.6	34.6	9.9	8.6	53.1	38.9	
male9	2	20	80.0	20.0	0.0	5.0	25.0	100.0	
Sum/Avg	174	1046	41.0	33.8	25.1	13.3	72.3	66.1	
Mean	9.7	58.1	45.2	32.7	22.1	12.0	66.8	69.7	
S.D.	6.4	26.8	16.5	11.4	14.4	10.1	21.1	18.5	
Median	8.0	63.0	42.7	35.5	21.5	8.4	63.8	66.7	

## H.2 Bigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrd	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	54.0	25.4	20.6	4.8	50.8	77.8	
female2	30	107	60.7	9.3	29.9	29.9	69.2	56.7	
female3	7	41	68.3	4.9	26.8	19.5	51.2	71.4	
female4	7	77	74.0	19.5	6.5	7.8	33.8	85.7	
female5	6	56	62.5	28.6	8.9	3.6	41.1	66.7	
female6	14	73	49.3	32.9	17.8	9.6	60.3	64.3	
female7	11	85	55.3	14.1	30.6	22.4	67.1	81.8	
female8	6	25	80.0	16.0	4.0	12.0	32.0	50.0	
female9	6	17	100.0	0.0	0.0	17.6	17.6	50.0	
male1	5	47	70.2	21.3	8.5	2.1	31.9	80.0	
male2	11	82	70.7	20.7	8.5	3.7	32.9	54.5	
male3	7	58	69.0	17.2	13.8	3.4	34.5	57.1	
male4	12	72	38.9	6.9	54.2	33.3	94.4	100.0	
male5	11	63	57.1	20.6	22.2	14.3	57.1	54.5	
male6	9	70	52.9	27.1	20.0	0.0	47.1	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	90.1	8.6	1.2	1.2	11.1	27.8	
male9	2	20	95.0	5.0	0.0	0.0	5.0	50.0	
Sum/Avg	174	1046	64.2	17.3	18.5	11.9	47.6	62.1	
Mean	9.7	58.1	69.3	15.5	15.2	10.9	41.6	62.7	
S.D.	6.4	26.8	17.8	9.9	14.2	10.1	23.2	18.3	
Median	8.0	63.0	68.6	16.6	11.4	8.7	37.8	60.7	



### H.3 Trigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	58.7	22.2	19.0	7.9	49.2	88.9	
female2	30	107	63.6	6.5	29.9	29.0	65.4	56.7	
female3	7	41	73.2	4.9	22.0	19.5	46.3	57.1	
female4	7	77	77.9	14.3	7.8	10.4	32.5	85.7	
female5	6	56	66.1	30.4	3.6	0.0	33.9	66.7	
female6	14	73	52.1	31.5	16.4	9.6	57.5	57.1	
female7	11	85	57.6	11.8	30.6	30.6	72.9	81.8	
female8	6	25	76.0	16.0	8.0	8.0	32.0	50.0	
female9	6	17	94.1	5.9	0.0	11.8	17.6	50.0	
male1	5	47	63.8	25.5	10.6	2.1	38.3	80.0	
male2	11	82	68.3	23.2	8.5	4.9	36.6	54.5	
male3	7	58	65.5	22.4	12.1	1.7	36.2	57.1	
male4	12	72	40.3	8.3	51.4	31.9	91.7	91.7	
male5	11	63	63.5	12.7	23.8	15.9	52.4	45.5	
male6	9	70	51.4	25.7	22.9	0.0	48.6	66.7	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	91.4	7.4	1.2	1.2	9.9	22.2	
male9	2	20	100.0	0.0	0.0	5.0	5.0	50.0	
Sum/Avg	174	1046	65.6	16.3	18.1	12.5	46.9	59.8	
Mean	9.7	58.1	70.2	14.9	14.9	11.1	41.0	60.8	
S.D.	6.4	26.8	17.0	10.1	13.6	10.4	22.6	19.0	
Median	8.0	63.0	65.8	13.5	11.4	8.8	37.4	57.1	

## H.4 Quadrigrams

../Output/Wer/multiplesyllablesAllhyp.trn									
SPKR	# Snt	# Wrds	Corr	Sub	Del	Ins	Err	S.Err	
female1	9	63	60.3	20.6	19.0	7.9	47.6	77.8	
female2	30	107	63.6	4.7	31.8	29.0	65.4	56.7	
female3	7	41	75.6	2.4	22.0	24.4	48.8	57.1	
female4	7	77	76.6	14.3	9.1	13.0	36.4	85.7	
female5	6	56	67.9	28.6	3.6	0.0	32.1	66.7	
female6	14	73	53.4	28.8	17.8	8.2	54.8	50.0	
female7	11	85	55.3	14.1	30.6	28.2	72.9	81.8	
female8	6	25	84.0	8.0	8.0	16.0	32.0	50.0	
female9	6	17	94.1	5.9	0.0	5.9	11.8	33.3	
male1	5	47	66.0	25.5	8.5	4.3	38.3	80.0	
male2	11	82	67.1	25.6	7.3	3.7	36.6	45.5	
male3	7	58	65.5	22.4	12.1	1.7	36.2	57.1	
male4	12	72	40.3	6.9	52.8	31.9	91.7	91.7	
male5	11	63	63.5	12.7	23.8	15.9	52.4	54.5	
male6	9	70	54.3	24.3	21.4	0.0	45.7	55.6	
male7	3	9	100.0	0.0	0.0	11.1	11.1	33.3	
male8	18	81	88.9	8.6	2.5	2.5	13.6	27.8	
male9	2	20	100.0	0.0	0.0	0.0	0.0	0.0	
Sum/Avg	174	1046	65.9	15.8	18.4	12.7	46.8	57.5	
Mean	9.7	58.1	70.9	14.1	15.0	11.3	40.4	55.8	
S.D.	6.4	26.8	16.9	10.0	13.9	10.7	23.1	23.2	
Median	8.0	63.0	66.5	13.4	10.6	8.1	37.4	56.1	

## **Appendix I    Software Written for the Study**

- I.1. ChooseBest.py: Chooses the best candidate output for a given collection of utterances.
- I.2. CollectVotes.py: Transforms the output of ROVER runs into a single hypothesis file.
- I. 3.CreateNbestFileClass.py: renames the output of the decoder and copies it to the nBest directory when the nbest option is on.
- I.4. Driver.py: Controls the operation of the basic software from transcription, to syllabification, to language model generation, to recognition, to scoring.
- I.5. LangModGen.py: Controls language model generation software.
- I.6. JuMakeConceptsClass.py: Transforms syllables in the decoder's nBest output to concepts/equivalence classes.
- I.7.MakeConceptHypFiles.py: Transforms syllables in the decoder's output to concepts/equivalence classes.
- I.8. MakeConceptRefFiles.py: Transforms syllables in the reference files to concepts/equivalence classes.
- I.9. Prplx.py: Computes the perplexity of the training transcript.
- I.10. Recognize.py: Controls the recognizer.
- I.11. RoverClass.py: Run's NIST ROVER software.
- I.12. ScLite.py-1: Controls scoring software in the basic experiment.
- I.13. ScLite.py-2: Controls scoring software when using concepts/equivalence classes.
- I.14. ScLiteTst.py: Controls ScLite.py.
- I.15. Syllabify.py: Controls NIST syllabification software that syllabifies the transcript.
- I.16. SyllabifyRefFiles: Controls NIST syllabification software that syllabifies the reference files.
- I.17. SyllablesToWords: Software that can transform syllabified output to words. Based on a chart parser written at Next IT. Works in conjunction with ChooseBest.py.
- I.18. Transcript.py: Controls transcription software.

## I.19. Utilities.py: Various one-off utilities used in the experiments

### I.1. Chooses the best output from a given collection of utterances. Used in Experiment 2.1

```

import string
import commands
import sys

class ChooseBest:
    def __init__(self,arguments):
        print '*****'
        print '*****'
        print 'ChooseBest.py'
        print '*****'
        print '*****'
        self.tempFiles = []
        self.args = []
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.transcript = self.args[0] #transcript used to build language model
        self.decoderWords = self.args[1] #output of Fan's parser
        self.finalScores = self.args[2] #Best cand from Fan's parser
        self.finalScoresTimes = self.args[3] #As above, but with times
        self.Score = self.args[4] #directory where scoring data is kept

        'temp files used in scoring. Store in list for deletion'

        self.ngramCounts = str(self.Score) + 'ngramCounts.dat'
        #print self.ngramCounts

        #self.tempFiles.append(self.ngramCounts)
        self.arpaFormat = str(self.Score) + 'arpaFormat'
        #self.tempFiles.append(self.arpaFormat)

    def cleanUp(self):
        #print 'deleting temporary files'
        for item in self.tempFiles:
            cmd = 'rm ' + item
            results = commands.getstatusoutput(cmd)

    def runGenerators(self):
        noErrors = True

        #save nextLM. replace it with one that uses NGram = 3
        cmd1 = 'cp ' + '../langMod/nextLM.py' + ' ../langMod/nextLMsave.py'
        results = commands.getstatusoutput(cmd1)
        if results[0] != 0:
            print "Error Saving NextLM"
            sys.exit(1)

        cmd2 = 'cp ' + '../langMod/nextLM3.py' + ' ../langMod/nextLM.py'
        results = commands.getstatusoutput(cmd2)
        if results[0] != 0:
            print "Error Saving NextLM"
            noErrors = False
            sys.exit(1)

        cmd3 = 'python ../langMod/genCounts3.py ' + self.transcript + ' > ' + self.ngramCounts

```

```

results = commands.getstatusoutput(cmd3)
if results[0] != 0:
    print results
    print "Error in Ngram"
    sys.exit(1)

#generate word language model
cmd4 = 'python ../../langMod/genModKNarpa3.py ' + '-G ' + self.ngramCounts + ' > ' + self.arpaFormat
results = commands.getstatusoutput(cmd4)
if results[0] != 0:
    print "Error in language model generation"
    noErrors = False
    sys.exit(1)

print "Language model for scoring has been generated"

#Returns a list of file names. Each file in the list contains
#candidate sentences generated by fan's parser for the same utterance

def genCandidateFiles(self):
    #Get output of Fan's parser ready for perplexity testing
    dirtyList = []
    dirtyFile = open(self.decoderWords, 'r')
    for item in dirtyFile:
        dirtyList.append(item)
    dirtyFile.close()

    listOfLists = []
    curTime = "(**.*.,**.*.)"

    for item in dirtyList:
        newlItem = str.split(item, ',')
        newlItem[0] = newlItem[0] + ' '
        newlItem[1] = str.lstrip(newlItem[1])
        newlItem[1] = str.rstrip(newlItem[1])
        littleList = []
        littleList.append(newlItem[0])
        littleList.append(newlItem[1])
        listOfLists.append(littleList)

    fileNum = 0
    fileList = []
    timeList = [] #contains times for each utterance--to be reinserted after scoring
    for lists in listOfLists:
        if curTime != lists[0]:
            curTime = lists[0]
            timeList.append(curTime)
            if fileNum > 0:
                tmpFile.close()
            fileNum = fileNum + 1
            tmpFileStr = str(self.Score) + 'f' + str(fileNum)
            self.tmpFiles.append(tmpFileStr) #store for deletion
            tmpFile = open(tmpFileStr, 'w')
            fileList.append(tmpFileStr)
            tmpFile.write(lists[1])
            tmpFile.write('\n')
    tmpFile.close()

    #print "List of Candidate files has been generated"
    return fileList, timeList

```

```

def chooseBestMain(self, fileList, timeList):

    #print "Choosing the best candidates..."
    finalScoreFile = open(self.finalScores, 'w')
    #generate the logprobs for each set of candidates
    for fileName in fileList:
        tmpFileStr = str(self.Score) + 'LogProb'
        self.tmpFiles.append(tmpFileStr) #store for deletion. This file contains the collection of candidates for a single
        utterance and their log probs
        cmd = 'python ../../langMod/ppl.py -V ' + fileName + ' ' + self.arpaFormat + ' > ' + tmpFileStr
        results = commands.getstatusoutput(cmd)
        if results[0] != 0:
            print results
            print "Error in perplexity generation"
        self.chooseBest(tmpFileStr, finalScoreFile)
    finalScoreFile.close()
    self.reinsertTimes(timeList)

    #restore nextLM with the the one using the proper NGram size
    cmd = 'cp ' + '../../langMod/nextLMSave.py' + ' ../../langMod/nextLM.py'
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error resetting NextLM"
        sys.exit(1)

    print "Reset NextLM"
    print cmd
    #print "Final Output " + self.finalScores
    #print "Final Output with Timing Data " + self.finalScoresTimes
    return self.finalScoresTimes

def reinsertTimes(self, timeList):
    finalScoreFile = open(self.finalScores, 'r')
    finalScoreFileTimes = open(self.finalScoresTimes, 'w')
    scoreList = []
    i = 0
    for item in finalScoreFile:
        #line = 'HYP: ' + str(timeList[i]) + ' ' + item
        line = str(timeList[i]) + ' ' + item
        finalScoreFileTimes.write(line)
        i = i + 1
    finalScoreFileTimes.close()

def chooseBest(self, tmpFileStr, finalScoreFile):
    scoredList = self.cleanLogProbs(tmpFileStr)

    #Linear search through scores looking for best candidate. Write to a file

    curBestScore = -500.00
    curBestString = ""
    for stuff in scoredList:
        if float(stuff[1]) > curBestScore:
            curBestScore = float(stuff[1])
            curBestString = stuff[0]

    finalScoreFile.write(curBestString);
    finalScoreFile.write("\n")

def cleanLogProbs(self, tmpFileStr):
    logFile = open(tmpFileStr, 'r')
    candList = []

    """remove extraneous stuff generated by ppl.py"""
    for line in logFile:
        found = False;

```

```

pos = string.find(line, 'Input:')
if pos >= 0:
    line = str.replace(line, 'Input: ', '')
    found = True
pos = string.find(line, 'logprob')
if pos >= 0:
    line = str.replace(line, 'logprob= ', '')
    found = True;
if found:
    line = str.rstrip(line)
    line = str.lstrip(line)
    candList.append(line)

"""Create a list of lists, where the inner list is a candidate and its log prob"""
scoredList = []
i = 0
for item in candList:
    if i % 2 == 0:
        littleList = []
        littleList.append(item)
    if i % 2 == 1:
        numList = str.split(item, ", ")
        littleList.append(numList[0])
    scoredList.append(littleList)
    i = i + 1

return scoredList

```

## I.2. Collect Votes. Transforms the output of ROVER runs into a single hypothesis file. Used in Experiment 3.2

```

import commands
import sys

nbestDir = '../Output/Decoder/nbest/'
decoderDir = '../Output/Decoder/'
decoderSave = '../Output/Decoder/DecoderSave/ngram'
class CollectVotesClass:
    def __init__(self,ngramIn,spkrIn):
        print ''
        print "collecting votes"
        self.ngram = ngramIn
        self.spkr = spkrIn
        numUtts = getNumUtts()
        uttList = makeUttList(numUtts)
        stringList = getStrings(uttList)
        stringList = cleanStringList(stringList)
        stringList = insertTimes(self, stringList)
        fileName = nbestDir + self.spkr + 'BestConceptTimes.out'
        fout = open(fileName, 'w')
        for item in stringList:
            fout.write(item + '\n')
        fout.close()
        target = decoderDir + self.spkr + 'decoderBestWordsTimes.out'
        cmd = 'cp ' + fileName + ' ' + target
        print cmd
        results = commands.getstatusoutput(cmd)
        if (results[0] != 0):
            print "error copying rover output to proper directory"
            sys.exit(1)

        print 'Wrote ' + target
        deleteFiles(nbestDir)

def getStrings(uttList):
    stringList = []
    for file in uttList:
        fin = open(nbestDir + file, 'r')
        str1 = fin.read()
        stringList.append(str1)
    return stringList

def makeUttList(numUtts):
    uttList = []
    for i in range(numUtts):
        uttList.append('vote' + str(i) + '.rvr')
    return uttList

def cleanStringList(stringList):
    filler = '7654 A 11.340 0.200 '
    conf = '-1.000000'
    eol = '\n'
    newStringList = []
    for string in stringList:
        string = string.replace(filler, '')
        string = string.replace(conf, '')
        string = string.replace(eol, '')
        newStringList.append(string)
    return newStringList

def insertTimes(self,stringList):
    fileName = decoderSave + self.ngram + '/' + self.spkr + 'decoderBestWordsTimes.out'

```



```

fin = open(fileName, 'r')
i = 0
newStringList = []
for line in fin:
    lineLst = line.split(' ')
    newLine = lineLst[0] + ' '
    newStringList.append(newLine + ' ' + stringList[i])
    i = i + 1
return newStringList

def deleteFiles(nbestDir):
    cmd = 'rm ' + nbestDir + '*.ctm'
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "No ctm files to remove"

    cmd = 'rm ' + nbestDir + '*.cncpt'
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "No .cncpt files to remove"

    cmd = 'rm ' + nbestDir + '*.rvr'
    print cmd
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "No .rvr files to remove"

def getNumUtts():
    voteOut = nbestDir + 'voteOut.out'
    cmd = 'ls ' + nbestDir + '*.rvr > ' + voteOut
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "error counting vote files"
        sys.exit(1)

    fin = open(voteOut, 'r')
    numUtts = 0
    for item in fin:
        numUtts = numUtts + 1
    fin.close()

    cmd = 'rm ' + voteOut
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "error removing temp vote file"
        sys.exit(1)
    return numUtts

```

### **I.3. Renames the output of the decoder and copies it to the NBest directory when the NBest option on. Used in Experiment 3.2**

```
import sys
import commands

class CreateNbestFileClass:
    def __init__(self, spkr):
        print 'Voting for ' + spkr
        nbestDir = '../Output/Decoder/nbest'
        num = howMany(spkr, nbestDir)
        fout = open(nbestDir + '/Nbest.in', 'w')
        for i in range (num):
            fileName = nbestDir + '/' + spkr + '.raw_' + str(i) + '.Nbest'
            print fileName
            fout.write(fileName + '\n')
        fout.close()

def howMany(spkr, nbestDir):
    cmd = 'ls ' + nbestDir + '/' + spkr + '*.Nbest > tmp'
    print cmd
    results = commands.getstatusoutput(cmd)
    if (results[0] != 0):
        print "error determining number of Nbest files"
        sys.exit(1)
    fin = open('tmp', 'r')
    num = 0
    for file in fin:
        num = num + 1
    fin.close()
    return num
```

#### **I.4. Controls the operation of most of the software from transcription, to syllabification, to language model generation, to recognition, and scoring. Used in Experiments 2 and 3**

```

from Transcribe import *
from Syllabify import *
from LangModGen import *
from Recognize import *
from SyllablesToWords import *
from ChooseBest import *
from ScLite import *
from os.path import *
import commands
import sys

def recognizeLoop(syllablesOrWords, audioFileName, scliteArgs, recognizeArgs, syllToWordsArgs, chooseBestArgs, refFile):
    #get names of audio files and corresponding align files. align file names to be returned to sclite loop
    """
    print "*****"
    print "Arguments to recognizeScoreLoop"
    print "syllables or words " + str(syllablesOrWords)
    print "audio file " + str(audioFileName)
    print "scliteArgs " + str(scliteArgs)
    print "recognizeArgs " + str(recognizeArgs)
    print "syllToWordsArgs " + str(syllToWordsArgs)
    print "chooseBestArgs " + str(chooseBestArgs)
    print "refFile " + str(refFile)
    print "End Arguments"
    """
    audioFiles = []
    alignFiles = []
    audio = open(audioFileName, 'r')
    for fname in audio:
        fname = str.rstrip(fname)
        audioName = fname + '.raw'
        audioFiles.append(audioName)
        alignName = fname + '.align'
        alignFiles.append(alignName)

    for fileName in audioFiles:
        print "Recognizing " + fileName
        """if word lang model, best contains the cleaned up output
        of the recognizer. If syll lang model, further processing might be
        necessary.
        """

        best = recognizeStep(syllablesOrWords, recognizeArgs, fileName)

        if (syllablesOrWords == "words" and refFile == "syllables"):
            print "Error: reference file is syllables while using word language model"
            sys.exit(1)

        if (syllablesOrWords == "syllables" and refFile == "words"): #translate syllable output to words
            syllToWordsStep(syllToWordsArgs)
            best = chooseBestStep(chooseBestArgs)#choose the best words from fan's parser

        if (syllablesOrWords == "syllables" and refFile == "syllables"):
            #convert to upper case and remove underscore separating phones
            bestFileLower = open(best, 'r')
            bestFileUpper = open('tmp', 'w')
            for line in bestFileLower:
                line = string.upper(line)
                line = line.replace('_', '')
                bestFileUpper.write(line)
            bestFileLower.close()
            bestFileUpper.close()
            cmd = 'cp ' + 'tmp' + ' ' + best

```

```

cmd1 = 'rm ' + 'tmp'
results = commands.getstatusoutput(cmd)
results = commands.getstatusoutput(cmd1)

#necessary so that files that don't pass through fan's parser have the same output as from the parser
#put a fix here 12/18. I confess that I don't remember what this does

lst = best.split('Syll')
if len(lst) > 1:
    newBest = str(lst[0] + 'Words' + str(lst[1]))
    cmd = 'cp ' + best + ' ' + newBest
    results = commands.getstatusoutput(cmd)
    if (results[0] != 0):
        print "error constructing decoder file output when parser is not used"
        sys.exit(1)
    best = newBest

"""the idea here is to rename the output of the decoder so that
it corresponds to the raw file used as input"""
pair = os.path.split(best)
out = str.rstrip(fileName)
out = str.split(out, '.')
out1 = str.lstrip(pair[1])
output = pair[0] + '/' + out[0] + out1
cmd = 'mv ' + best + ' ' + output

print "Here is the decoder output to be sent on to sclite: " + str(output)
results = commands.getstatusoutput(cmd)
if (results[0] != 0):
    print "error constructing decoder file output name"
    sys.exit(1)
print "Here are the alignment files: " + str(alignFiles)
return alignFiles

def scliteStep(scliteArgs, syllablesOrWords, alignFiles):
    for alignmentFile in alignFiles:
        fileName = str.split(alignmentFile, '.')
        print "Running sclite on recognizer's guess for: " + str(fileName[0])
        sclite = ScLite(scliteArgs, syllablesOrWords, alignmentFile)
        sclite.runRefHypAlign()
        sclite.runConcat()
        scoringData = sclite.runScLite()
    return scoringData

def chooseBestStep(chooseBestArgs):
    print "Choosing the best of the Fan-translated syllable hypotheses"
    #generate a word language model
    so = ChooseBest(chooseBestArgs)
    so.runGenerators()
    #divide parser output into files where each file
    #contains the candidate utterances for a given time
    fileList, timeList = so.genCandidateFiles()
    #Compute log likelihood for each candidate
    best = so.chooseBestMain(fileList, timeList)
    so.cleanUp()
    print ""
    print ""
    return best

def syllToWordsStep(syllToWordsArgs):
    print "Syllables To Words Step"
    sw = SyllablesToWords(syllToWordsArgs)
    sw.callParse()

```

```

print ""
print ""

def recognizeStep(syllablesOrWords, recognizeArgs, audioFile):
    print "Recognition Step"
    rc = Recognize(syllablesOrWords, recognizeArgs, audioFile)
    rc.runDecoder()
    return rc.cleanUp() #cleaned up results of recognition
    print ""
    print ""

def langModelStep(syllablesOrWords, langModArgs):
    print "Language Model Generation Step"
    lm = LangModGen(langModArgs)
    if (syllablesOrWords == "syllables"):
        lm.createLexiconSyll()
    else:
        lm.createLexiconWord()
    lm.genNGramCounts()
    lm.genLangModel()
    lm.genBinaryLangModel()
    print ""
    print ""

def syllabifyStep(utteranceFile, complDir, syllabifyArgs):
    print "Syllabification Step"
    syll = Syllabify(utteranceFile, complDir, syllabifyArgs)
    syll.syllabify()
    syll.writeDict()
    #syll.readDict()
    syll.produceSyllabifiedTranscript()
    #syll.cleanUp()
    print ""
    print ""

def transcribeStep(utteranceFile, outputDir):
    print "Transcription Step"
    trans = Transcribe(utteranceFile, outputDir)
    trans.transcribe()
    print ""
    print ""

def readArgs():
    #print "Reading global argument file"
    argFile = open("../Arguments/globalArgs.args")
    argList = []
    for arg in argFile:
        arg = string.rstrip(arg, '\n')
        argList.append(arg)
    argFile.close()
    print ""
    print ""
    return argList

def controlLoop(syllablesOrWords, refFile):

    argList = readArgs()
    outputDir = argList[0]
    utteranceFile = argList[1]
    langModArgs = argList[2]
    recognizeArgs = argList[3]
    syllToWordsArgs = argList[4]
    chooseBestArgs = argList[5]
    syllableLangModelFile = argList[6]
    wordLangModelFile = argList[7]
    syllableConfigurationFile = argList[8]
    wordConfigurationFile = argList[9]
    scliteArgs = argList[10]

```

```

audioFileNames = argList[11]
syllabifyArgs = argList[12]

#argument cd ..specifies whether we are generating
#a syllable or word language model. If the argument is "words," the transcribe
#and syllabify steps are not necessary, nor is it necessary to run createLexicon from
#within LangModGen

print "*****LANGUAGE MODEL CONSTRUCTION*****"
print "What size N-Grams: 1, 2, 3, 4"
size = raw_input("Enter one: ")
genCounts = "genCounts"
genModKNarpa = "genModKNarpa"
nextLM = "nextLM"

genCounts = genCounts + size + '.py'
genModKNarpa = genModKNarpa + size + '.py'
nextLM = nextLM + size + '.py'

cmd1 = 'cp ' + '../langMod/' + genCounts + ' ../langMod/genCounts.py'
cmd2 = 'cp ' + '../langMod/' + genModKNarpa + ' ../langMod/genModKNarpa.py'
cmd3 = 'cp ' + '../langMod/' + nextLM + ' ../langMod/nextLM.py'

print "NGram size set to " + size
print cmd1
print cmd2
print cmd3

results = commands.getstatusoutput(cmd1)
if results[0] != 0:
    print "Error copying genCounts"
    sys.exit(1)

results = commands.getstatusoutput(cmd2)
if results[0] != 0:
    print "Error copying genModKNarpa"
    sys.exit(1)

results = commands.getstatusoutput(cmd3)
if results[0] != 0:
    print "Error copying nextLM"
    sys.exit(1)

complDir = str(outputDir + 'Syllabify')
transcribeStep(utteranceFile, complDir)
if (syllablesOrWords == "syllables"): #gen syllable language model
    syllabifyStep(utteranceFile, complDir, syllabifyArgs)
    cmd = 'cp ' + syllableLangModelFile + ' ' + langModArgs #language model arguments to use when doing syllable
recognition
    cmd1 = 'cp ' + syllableConfigurationFile + ' ' + recognizeArgs #recognizer arguments to use when doing syllable
recognition
else:
    cmd = 'cp ' + wordLangModelFile + ' ' + langModArgs #language model arguments to use when doing word recognition
    cmd1 = 'cp ' + wordConfigurationFile + ' ' + recognizeArgs #recognizer arguments to use when doing word recognition

results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print "Error creating correct language model argument file. Ending execution"
    sys.exit()
results = commands.getstatusoutput(cmd1)
if results[0] != 0:
    print "Error creating correct recognize argument file. Ending execution"
langModelStep(syllablesOrWords, langModArgs)
print "*****END LANGUAGE MODEL CONSTRUCTION*****"
print " "
print " "

```

```

print "Run the recognizer?"
more = raw_input("Enter yes/no:")
if more == 'yes':
    print "*****BEGIN EXECUTING RECOGNITION CLASSES*****!!"
    alignFiles = recognizeLoop(syllablesOrWords, audioFileNames, scliteArgs, recognizeArgs, syllToWordsArgs,
chooseBestArgs, refFile )
    print "*****END EXECUTION OF RECOGNITION CLASSES*****!!"
    print " "
    print " "
    print " "
    print "Do SCLITE scoring?"
    more = raw_input("Enter yes/no:")
    if more == 'yes':
        print "*****BEGIN SCLITE
SCORING*****!!"
        scoringData = scliteStep(scliteArgs,syllablesOrWords,alignFiles)
        scoringDataFinal = scoringData + size
        cmd = 'mv ' + scoringData + ' ' + scoringDataFinal
        results = commands.getstatusoutput(cmd)
        if results[0] != 0:
            print "Error renaming scoring data"
            sys.exit(1)
        return scoringDataFinal
    else:
        return "nothing"
    print "*****END SCLITE SCORING*****!!"
    print " "
else:
    return "nothing"

def main():
    print "*****!"
    print "*****!"
    print 'Driver.py'
    print "*****!"
    print "*****!"

    """
    names of audio files to be scored are stored in:
    ~/pdepalma/multiUttDecode/audioFiles.txt
    names of all possible audio files are in:
    ~/pdepalma/multiUttDecode/audioFiles.full
    """

    print "Erase all previous output: all"
    print "Erase everything but sclite results: part"
    print "Erase nothing: none"
    erase = raw_input("Enter one: ")
    if erase == "all":
        cmd = "./initScript.sc"
    else:
        if erase == "part":
            cmd = "./initScriptSmall.sc"

    if (erase == "all" or erase == "part"):
        results = commands.getstatusoutput(cmd)
    else:
        if erase == "none":
            print "nothing erased"
        else:
            print "Erase Error"
            sys.exit(1)

    #set up proper reference files for scoring
    print "Reference for Scoring: syllables, words?"
    refFile = raw_input("Enter one: ")

```

```

if refFile == "words":
    cmd = "cp ../../multiUttDecode/refs/refsWord/*.* ../../multiUttDecode/refs/"
else:
    cmd = "cp ../../multiUttDecode/refs/refsSyll/*.* ../../multiUttDecode/refs/"
results = commands.getstatusoutput(cmd)
if (results[0] != 0):
    print "error copying proper reference file"
    sys.exit(1)

print "Language Model: syllables, words?"
syllablesOrWords = raw_input("Enter one: ")
if (syllablesOrWords != 'syllables' and syllablesOrWords != 'words' and syllablesOrWords != 'both'):
    print "Which language model error, syllables, words, or both?"
    sys.exit(1)

if syllablesOrWords == 'syllables':
    scoringDataFinal = controlLoop('syllables', refFile)
    print "Syllable Language Model Data Generated"
    print " "

if syllablesOrWords == 'words':
    scoringDataFinal = controlLoop('words', refFile)
    print "Word Language Model Data Generated"
    print " "

if scoringDataFinal != 'nothing':
    dirList = scoringDataFinal.split('final/')
    dir = dirList[0] + 'final/'

    #name scoring file so that it corresponds to type of
    # file being scored:
    #word ref file, word lang: .ww
    #word ref file, syll lang mod: .ws
    #syl ref file, syll lang mod: ss

    ext = ""
    if (refFile == 'words' and syllablesOrWords == 'words'):
        scoringData = scoringDataFinal.split("words")
        ext = 'ww'
    if (refFile == 'words' and syllablesOrWords == 'syllables'):
        scoringData = scoringDataFinal.split("syllables")
        ext = 'ws'
    if (refFile == 'syllables' and syllablesOrWords == 'syllables'):
        scoringData = scoringDataFinal.split("syllables")
        ext = 'ss'

    scoringDataFinal1 = 'score' + scoringData[1] + '.' + ext
    cmd = 'mv ' + str(scoringDataFinal) + ' ' + dir + str(scoringDataFinal1)
    print cmd
    results = commands.getstatusoutput(cmd)
    if (results[0] != 0):
        print "error giving final name to scoring file"
        sys.exit(1)

    print "SCLITE Scored Data found in: " + dir + str(scoringDataFinal1)

main()

```



## I.5. Controls language model generation software. Used in experiments 2 and 3

```

import string
import commands
import sys

#if a word language model, use --> langModWords.args
#if a syllable language model, use --> langModSylls.args
#Files used in syllable lang mod generation in this order :
#0:name of syllabified transcript
#1:ngram Counts (output of genCounts)
#2:arpa formatted language model (output of genModKNarpa)
#3:name of binary version of arpa formatted language model (output of lm_encode)
#4:name of syllable dictionary created through the class Syllabify
#5:name of lexicon.
#6:path to sonic so that lm_encode can be used

class LangModGen:
    def __init__(self, arguments):
        print '*****'
        print
        print 'LangModGen.py'
        print '*****'
        print
        self.args = []
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.syllables = ""

    def createLexiconSyll(self):
        syllableDict = self.args[4]
        lexicon = self.args[5]
        syllDict = open(syllableDict, 'r')
        lex = open(lexicon, 'w')

        for line in syllDict:
            line = string.rstrip(line)
            line = string.split(line, ':')
            strng = line[1]
            line = string.split(strng)
            phStr = ""
            for syll in line:
                phStr = phStr + syll
                phone = string.split(syll, '_')
                for item in phone:
                    phStr = phStr + ' ' + item
                phStr = string.upper(phStr)
                lex.write(phStr)
                phStr = ""
            lex.write("\n")
        syllDict.close()
        lex.close()

        #now, remove duplicates
        cmd = 'sort -n < ' + lexicon + ' | uniq > tmp'
        cmd1 = 'cp tmp ' + lexicon
        cmd2 = 'rm tmp'
        results1 = commands.getstatusoutput(cmd)
        results2 = commands.getstatusoutput(cmd1)
        results3 = commands.getstatusoutput(cmd2)

        results = results1[0] + results2[0] + results3[0]

```

```

if results > 0:
    print "error removing duplicates from lexicon"
    sys.exit(1)
self.appendOOV(lexicon)

def createLexiconWord(self):
    wordDict = self.args[4]
    lexicon = self.args[5]
    wrdDict = open(wordDict,'r' )
    lex = open(lexicon,'w')

    for line in wrdDict:
        line = string.rstrip(line)
        line = string.split(line,':')
        string0 = string.upper(line[0])
        string1 = string.upper(line[1])
        newStr = string0 + ' ' + string1
        lex.write(newStr)
        lex.write("\n")
    wrdDict.close()
    lex.close()

    #sort
    cmd = 'sort -n < ' + lexicon + ' > tmp'
    cmd1 = 'cp tmp ' + lexicon
    cmd2 = 'rm tmp'
    results1 = commands.getstatusoutput(cmd)
    results2 = commands.getstatusoutput(cmd1)
    results3 = commands.getstatusoutput(cmd2)

    results = results1[0] + results2[0] + results3[0]
    if results > 0:
        print "error"
    self.appendOOV(lexicon)

def appendOOV(self,lexicon):
    #append out of vocabulary stuff
    lex = open(lexicon,'a')
    lex.write('<UNK> re\n')
    lex.write('<SIL> SIL\n')
    lex.write('<BR> br\n')
    lex.write('<GA> ga\n')
    lex.write('<LS> ls\n')
    lex.write('<LG> lg\n')
    lex.write('<REG> reg\n')
    lex.close()
    print "lexicon created: " + lexicon

def genNGramCounts(self):
    transcript = self.args[0]
    ngramCounts = self.args[1]
    #generate N-gram counts
    cmd = 'python ../../langMod/genCounts.py ' + transcript + ' > ' + ngramCounts
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in N-Gram"
        noErrors = False
        return noErrors
    #print "N-Gram counts generated"

def genLangModel(self):
    ngramCounts = self.args[1]
    arpaFormat = self.args[2]

    #generate language model

```

```

cmd = 'python ../../langMod/genModKNarpa.py ' + '-G ' + ngramCounts + ' > ' + arpaFormat
results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print "Error in language model generation"
    print results[0]
    print results[1]
    noErrors = False
    return noErrors
#print "language model generated: " + arpaFormat

def genBinaryLangModel(self):
    arpaFormat = self.args[2]
    arpaBinary = self.args[3]
    pathToSonic = self.args[6]

    langModEncode = pathToSonic + 'lm_encode'
    cmd = langModEncode + ' ' + arpaFormat + ' ' + arpaBinary
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print results
        print "Error in binary language model generation"
        sys.exit(1)
    #print 'binary language model generated: ' + arpaBinary

```

## I.6. Creates the Concept hypothesis files from the N best output of the recognizer. Used in Experiment 3.2

```

import commands
import sys
class MakeConceptsClass:
    def __init__(self):
        conceptList, conceptDict = makeConceptDS()
        makeConceptHypFile(conceptList, conceptDict)

def makeConceptDS():
    """Returns a list of all syllable strings that map to any concept, from
    longest to shortest. Returns a dictionary, concepts are indexed by their
    corresponding syllable strings"""
    conceptList = []
    conceptDict = {}
    dir = './Concepts/'
    conceptNamesIn = open(dir + 'conceptNamesIn.txt', 'r')
    for conceptName in conceptNamesIn:
        conceptName = conceptName.rstrip()
        inFile = conceptName + '.txt'
        fileIn = open(dir + inFile, 'r')
        for concept in fileIn:
            concept = concept.rstrip()
            concept = concept.replace('_', '.')
            concept = concept.upper()
            conceptList.append(concept)
            conceptDict[concept] = conceptName
        fileIn.close()
    conceptList = sorted(conceptList, key=len)
    revList = []
    for i in range(len(conceptList) - 1, -1, -1):
        revList.append(conceptList[i])

    conceptNamesIn.close()
    return revList, conceptDict

def makeConceptHypFile(conceptList, conceptDict):
    """Substitutes concepts for corresponding syllable strings in the
    hypothesis files"""
    dir = './Concepts/'
    decoderDir = './Output/Decoder'
    nbestDir = './Output/Decoder/nbest'

    cmd = 'rm ' + nbestDir + '/*.ctm'
    print cmd
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "No ctm files to remove"

    cmd = 'rm ' + nbestDir + '/*.cncpt'
    print cmd
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "No cncpt files to remove"

    files = open(nbestDir + '/Nbest.in', 'r')
    fileList = []
    for item in files:
        item = str.rstrip(item)
        fileList.append(item)
    files.close()

    for fileIn in fileList:
        fileOut = fileIn + '.cIn'

```

```

    cleanFiles(fileIn,fileOut)

cleanFileList = []
for item in fileList:
    item = item + '.cln'
    cleanFileList.append(item)

conceptFileList = []
for item in fileList:
    itemList = item.split('Nbest')
    conceptFileList.append(itemList[0] + 'cncpt')

print 'creating concept files'
for i in range(len(cleanFileList)):
    hypFileIn = open(cleanFileList[i], 'r')
    conceptFileOut = open(conceptFileList[i], 'w')
    print conceptFileList[i]
    readWriteFile(hypFileIn, conceptFileOut, conceptList, conceptDict)
    hypFileIn.close()
    conceptFileOut.close()

#delete temporary files
cmd = 'rm ' + nbestDir + '/* .cln'
print cmd
results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print "No clean files to remove"

def cleanFiles(fileIn, fileOut):
    """eliminates underscores, timing data at end, changes to upper case"""
    dirty = open(fileIn, 'r')
    clean = open(fileOut, 'w')

    for line in dirty:
        line = line.replace('_', '')
        line = line.upper()
        lineList = line.split(' ')
        line = lineList[0]
        line = str.rstrip(line)
        #if there is nothing on the line, rover fails
        if len(line) == 0:
            line = line + '<SIL><UNK>'
        clean.write(line + '\n')
    dirty.close()
    clean.close()

def readWriteFile(refFileIn, conceptFileOut, conceptList, conceptDict):
    for line in refFileIn:
        for syllString in conceptList:
            line = line.rstrip()
            line = ' ' + line + ' '
            concept = ' ' + conceptDict[syllString] + ' '
            syllString = ' ' + syllString + ' '
            line = line.replace(syllString, concept)
            line = line.lstrip()
            line = line.rstrip()
            conceptFileOut.write(line + '\n')

def getFileList(nbestDir, decoderDir):

    cmd = 'ls ' + nbestDir + ' > ' + decoderDir + '/fileList'
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error creating nbest file list"
        sys.exit(1)

    hypFiles = open(decoderDir + '/fileList')

```

```
hypFileList = []  
for item in hypFiles:  
    item = item.rstrip()  
    hypFileList.append(nbestDir + '/' + item)  
hypFiles.close()  
return hypFileList
```

## I.7. Transforms syllables in the decoder's output to concepts. Used in Experiment 3.2

```

import commands
import sys
def makeConceptDS():
    """Returns a list of all syllable strings that map to any concept, from
    longest to shortest. Returns a dictionary, concepts are indexed by their
    corresponding syllable strings"""
    conceptList = []
    conceptDict = {}
    dir = './Concepts/'
    conceptNamesIn = open(dir + 'conceptNamesIn.txt', 'r')
    for conceptName in conceptNamesIn:
        conceptName = conceptName.rstrip()
        inFile = conceptName + '.txt'
        fileIn = open(dir + inFile, 'r')
        for concept in fileIn:
            concept = concept.rstrip()
            concept = concept.replace('_', '')
            concept = concept.upper()
            conceptList.append(concept)
            conceptDict[concept] = conceptName
        fileIn.close()
    conceptList = sorted(conceptList, key=len)
    revList = []
    for i in range(len(conceptList) - 1, -1, -1):
        revList.append(conceptList[i])

    conceptNamesIn.close()
    return revList, conceptDict

def makeConceptHypFile(conceptList, conceptDict):
    """Substitutes concepts for corresponding syllable strings in the
    hypothesis files"""
    dir = './Concepts/'
    refsDirSyll = '../multiUttDecode/refs/refsSyll/'
    refsDirCncpt = '../multiUttDecode/refs/refsConceptNBest/'
    refsDir = '../multiUttDecode/refs/'

    fileLst = getFileLst(refsDir)

    for file in fileLst:
        makeConcepts(file, refsDirSyll, refsDirCncpt, conceptList, conceptDict)

def getFileLst(refsDir):
    fin = open(refsDir + 'refs.list', 'r')
    fileLst = []
    for item in fin:
        item = item.rstrip()
        fileLst.append(item)
    fin.close()
    return fileLst

def makeConcepts(file, refsDirSyll, refsDirCncpt, conceptList, conceptDict):
    syllFile = open(refsDirSyll + file, 'r')
    cncptFile = open(refsDirCncpt + file, 'w')
    for line in syllFile:
        for syllString in conceptList:
            concept = '' + conceptDict[syllString] + ''
            syllString = '' + syllString + ''
            pos = line.find(syllString)
            if (pos >= 0):

```

```
        line = line.replace(syllString,concept)
        cncptFile.write(line)
    print refsDirCncpt+file
    syllFile.close()
    cncptFile.close()

def main():
    conceptList, conceptDict = makeConceptDS()

    makeConceptHypFile(conceptList, conceptDict)

main()
```



## I.8. Creates the Concept reference files from the already syllabified reference files. Used in Experiment 3

```
def makeConceptDS():
    """Returns a list of all syllable strings that map to any concept, from
    longest to shortest. Returns a dictionary, concepts are indexed by their
    corresponding syllable strings"""
    conceptList = []
    conceptDict = {}
    dir = '../Concepts/'
    conceptNamesIn = open(dir + 'conceptNamesIn.txt', 'r')
    for conceptName in conceptNamesIn:
        conceptName = conceptName.rstrip()
        inFile = conceptName + '.txt'
        fileIn = open(dir + inFile, 'r')
        for concept in fileIn:
            concept = concept.rstrip()
            concept = concept.replace('_', '')
            concept = concept.upper()
            conceptList.append(concept)
            conceptDict[concept] = conceptName
        fileIn.close()
    conceptList = sorted(conceptList, key=len)
    revList = []
    for i in range(len(conceptList) - 1, -1, -1):
        revList.append(conceptList[i])

    conceptNamesIn.close()
    return revList, conceptDict
```

```
def makeConceptRefFile(conceptList, conceptDict):
    """Substitutes concepts for corresponding syllable strings in the
    reference files"""
    dir = '../Concepts/'
    refsDir = '../multiUttDecode/refs/refsSyll/'
    conceptsDir = '../multiUttDecode/refs/refsConceptNBest/'
    refFiles = open(dir + 'refFilesIn.txt')
    refFileList = []
    conceptFileList = []
    for item in refFiles:
        item = item.rstrip()
        refFileList.append(refsDir + item)
        conceptFileList.append(conceptsDir + item)
    print conceptFileList
    refFiles.close()

    for i in range(len(refFileList)):
        refFileIn = open(refFileList[i], 'r')
        conceptFileOut = open(conceptFileList[i], 'w')
        readWriteFile(refFileIn, conceptFileOut, conceptList, conceptDict)
        refFileIn.close()
        conceptFileOut.close()
        print conceptFileList[i] + ' written'
```

```
def readWriteFile(refFileIn, conceptFileOut, conceptList, conceptDict):
```

```
    for line in refFileIn:
        for syllString in conceptList:
            concept = '' + conceptDict[syllString] + ''
            syllString = '' + syllString + ''
            pos = line.find(syllString)
            if (pos >= 0):
                line = line.replace(syllString, concept)
        conceptFileOut.write(line)
```

```
def main():  
    conceptList, conceptDict = makeConceptDS()  
    makeConceptRefFile(conceptList, conceptDict)  
  
main()
```

## I.9. Computes the perplexity of the training transcript. Used in Experiment 1

```

import string
import random
import math
import commands
import sys
import re
from numpy import *

"""This script computes the perplexity of either the word or syllable transcript multiple times, then
computes the median, mean, and stdev over all runs. Each run uses a different test and training set,
where the training set is the original transcript minus the part used in the test set. The fraction used
in the test set is a parameter
"""

class Prplx:
    def __init__(self):
        self.args = []
        #Arguments used in perplexity testing in this order:
        #name of transcript file (either syllabified or not)
        #name of training file
        #name of test file
        #fraction of transcript used in testing
        #ngram Counts (output of genCounts)
        #arpa formatted language model (output of genModKNarpa)
        #perplexity data (output of ppl)
        #number of times to compute the perplexity. Each run results
        # in a line the perplexity output file
        #name of file holding perplexity statistics
        #name of binary version of arpa formatted language model
        #name of file that contains syllables alone along with their counts
    def getArgs(self, arguments):
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.cleanUpFiles()

    def genPerplexData(self):
        perplexity = self.args[6]
        perplexityComplete = self.args[9] #perplexity file to be deleted if both loops don't run to completion
        numTimes = int(self.args[7])
        finished = False

        #print "Computing perplexity " + str(numTimes) + " times"
        while (not finished):
            for i in range(0, numTimes, 1):
                print "Run " + str(i)
                self.divideSyllableTrans()
                finished = self.runGenerators()
                if (not finished):
                    cmd = 'rm ' + perplexityComplete
                    results = commands.getstatusoutput(cmd)
                    print "Errors. Starting over"
                    break
            self.computeResults()

    def cleanUpFiles(self):
        cmd = 'rm ' + './Output/Perplexity/*.dat'
        results = commands.getstatusoutput(cmd)

    def computeResults(self):
        perplexityComplete = self.args[9]
        perplexFile = open(perplexityComplete, 'r')
        perplexList = []
        for line in perplexFile:

```

```

    perplexDataLine = re.findall(r'ppl = [0-9]+\.[0-9]+',line)
    perplexData = re.findall(r'[0-9]+\.[0-9]+',str(perplexDataLine))
    perplexList.append(float(perplexData[0]))
perplexFile.close()
#use NumPy to compute perplexity statistics
numpyArray = array(perplexList)
mn = mean(numpyArray)
mn = around(mn,2) #round to two decimals
md = median(numpyArray)
md = around(md,2)
st = std(numpyArray)
st = around(st, 2)

#write stats to a file
perplexStats = self.args[8]
stats = open(perplexStats,'w')
testFract = int(self.args[3])
trainFract = 100 - testFract
numTimes = self.args[7]
stats.write("Number of Runs: " + numTimes + "\n")
stats.write("Fraction of transcript used in training: " + str(trainFract) + "\n")
stats.write("Fraction of transcript used in testing: " + str(testFract) + "\n")
stats.write("Mean = " + str(mn) + "\n")
stats.write("Median = " + str(md) + "\n")
stats.write("Standard Deviation = " + str(st) + "\n")
stats.close()
print "perplexity statistics generated: " + perplexStats

#Pre: getArgs has been run
def divideSyllableTrans(self):

    transcript = self.args[0] #transcript file
    trainData = self.args[1] #training file
    testData = self.args[2] #testing file
    testFractIn = self.args[3] #fraction of transcript used for testing

    sylTr = open(transcript, 'r')
    sylTrain = open(trainData, 'w')
    sylTest = open(testData, 'w')

    sylTrList = []
    for utterance in sylTr: #read utterances into a list
        sylTrList.append(utterance)

    testFract = int(testFractIn) #fraction of the transcript to use in test

    if testFract == 100: #use 100% in both training and testing. This produces the best possible results
        self.cheat(sylTrain, sylTest, sylTrList)
    else:
        self.dontCheat(sylTrain, sylTest,sylTrList, testFract) #use testFract in testing and 100 - testFract in training

    sylTr.close()
    sylTrain.close()
    sylTest.close()

def cheat(self,sylTrain,sylTest,sylTrList):
    #put the entire transcript in both the training and test files
    for i in range(0, len(sylTrList), 1):
        sylTest.write(sylTrList[i])
        sylTrain.write(sylTrList[i])

def dontCheat(self,sylTrain,sylTest,sylTrList, testFract):
    testLines = int(math.floor(.01 * testFract * len(sylTrList))) #number of lines in test
    testTop = random.randint(0, len(sylTrList) - testLines) #generate a random number at least testLines from the end of the
list-->becomes top of test data
    testBot = testTop + testLines #bottom of test data
    train = 0

```

```

test = 0
for i in range(0, len(sylTrList), 1):

    if (i >= testTop) and (i <= testBot):
        sylTest.write(sylTrList[i])
        test = test + 1
    else:
        sylTrain.write(sylTrList[i])
        train = train + 1

def runGenerators(self):
    noErrors = True
    transcript = self.args[0]
    trainData = self.args[1]    #training file
    testData = self.args[2]    #testing file
    ngramCounts = self.args[4]
    arpaFormat = self.args[5]
    perplexity = self.args[6]
    perplexityComplete = self.args[9]

    #generate N-gram counts
    cmd = 'python ../../langMod/genCounts.py ' + trainData + ' > ' + ngramCounts
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in Ngram"
        noErrors = False
        return noErrors

    #generate language model
    cmd = 'python ../../langMod/genModKNarpa.py ' + '-G ' + ngramCounts + ' > ' + arpaFormat
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in language model generation"
        noErrors = False
        return noErrors

    #generate perplexity file
    cmd = 'python ../../langMod/ppl.py ' + testData + ' ' + arpaFormat + ' > ' + perplexity
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in perplexity generation"
        noErrors = False
        return noErrors
    cmd = 'cat ' + perplexity + ' >> ' + perplexityComplete
    results = commands.getstatusoutput(cmd)
    return noErrors

#count tokens
def typeCnts(self):

    ngramsIn = self.args[4]
    typeCounts = self.args[10]
    ngramsIn = open(ngramsIn, 'r')
    typesOut = open(typeCounts, 'w')
    numbers = '[0-9]+'
    typeList = []
    for line in ngramsIn:
        lineList = string.split(line)
        #singletons have digits after the first space
        if (re.search(numbers, lineList[1])):
            typeList.append(lineList[0])

    typeList.sort()
    for item in typeList:
        typesOut.write(item + '\n')

    typesOut.write("Total Types: " + str(len(typeList)))

```

```

ngramsIn.close()
typesOut.close()

''' print "Token counts generated: " + typeCounts
'''
sort is on item in position 0 within item in position 1.
Transform the list, position 1 item is prepended to the tuple
Python naturally sorts on the 1st item within the 0th
So, sort the list
Now reconstruct the list, eliminated the prepended item
'''
def schwartzianSort(self, typeList):
    for n in range(len(typeList)):
        typeList[n] = (typeList[n][1], typeList[n])
    typeList.sort(reverse = True)
    for n in range(len(typeList)):
        typeList[n] = typeList[n][1]

```

## I.10. Controls the recognizer. Used in Experiments 2 and 3

```

import string
import commands
import sys
import re
#from SyllablesToWords import *
import os

#Files used in recognition:
#0:name of decoder
#1:name of configuration file for the decoder relative to the decoder
#2:name of sound file to be recognized relative to the decoder
#3:name of the file that contains the output of the recognizer
#4:name of the file that contains only the syllables hypothesized in #3
#5:dictionary of words and their syllabification
#6:name of the file that contains Fan's guesses at words, given syllables in #5

class Recognize:
    def __init__(self, syllablesOrWords, arguments, audioFile):
        print '*****'
        print '*****'
        print 'Recognize.py'
        print '*****'
        print '*****'
        self.args = []
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.syllables = ""
        self.syllablesOrWords = syllablesOrWords
        self.audioFile = audioFile

    def runDecoder(self):
        print "Running recognizer. Please wait."
        decoder = self.args[0]
        configFile = self.args[1]
        audioDirectory = self.args[2]

        cmd = decoder + ' ' + configFile + ' ' + audioDirectory + '/' + self.audioFile

        #necessary because constructing cmd seems to have put some garbage in the string
        cmd = str.lstrip(cmd)
        cmd = str.rstrip(cmd)
        results = commands.getstatusoutput(cmd)

        print cmd
        if results[0] != 0:
            print results[0]
            print results[1]
            print "Error running decoder"
            sys.exit(1)

        preSyllables = open(self.args[3], 'w')
        for item in results[1]:
            preSyllables.write(item)
        preSyllables.close()

    #Removes extraneous recognizer stuff
    #Saves timing data if the recognizer is working on syllables because
    #this is necessary to judge the best candidate from the syll->word parser
    #Saves HYP: + timing data if words because the next step reinserts scoring step
    #reinserts HYP: for syllables
    def cleanUp(self):

```

```

tokens = 'HYP:'
decoderDirty = open(self.args[3], 'r')
decoderClean = open(self.args[4], 'w')

for line in decoderDirty:
    if (re.search(tokens,line)):
        line = str.lstrip(line)
        line = str.rstrip(line)
        decoderClean.write(line)
        decoderClean.write('\n')
decoderDirty.close();
decoderClean.close();

cmd = 'cp ' + self.args[4] + ' ' + self.args[5]
results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print results[1]
    print "Error running decoder"
    sys.exit(1)
return self.args[5] #results using word lang model

```



## I.11. Controls ROVER. Used in Experiment 3.2

```

import commands
import sys
nbestDir = '../Output/Decoder/nbest/'

class RunRoverClass:
    def __init__(self):
        fin = open(nbestDir + 'Nbest.in', 'r')
        fileLst = []
        for file in fin:
            file = file.rstrip()
            fileLst.append(file)
        hypNum = 0
        for i in range (len(fileLst)):
            numNbest = getNbest(fileLst[i])
            hypStr = makeString(hypNum,numNbest)
            runRover(hypStr)
            hypNum = hypNum + 1

def makeString(hypNum,maxNbest):

    hypString = './' + nbestDir + 'rover' + ' -m meth1 -s '
    for i in range(maxNbest):
        hypString = hypString + '-h ' + nbestDir + 'out' + str(hypNum) + '_' + str(i) + '.ctm ctm '
    hypString = hypString + '-o ' + nbestDir + 'vote' + str(hypNum) + '.rvr'
    return hypString

def runRover(hypStr):
    print 'running rover using ' + hypStr
    print ''
    results = commands.getstatusoutput(hypStr)
    if (results[0] != 0):
        print results[0]
        print results[1]
        print "error running rover"
        sys.exit(1)

def getNbest(file):
    print file
    fin = open(file, 'r')
    lineNum = 0
    for item in fin:
        lineNum = lineNum + 1
    #necessary for the case when there is a single line. rover won't work
    if lineNum > 1:
        return lineNum
    else:
        return lineNum + 1

```

## I.12. Controls scoring software for Experiment 2

```

import string
import sys
import commands

class CommandError(Exception):
    def __init__(self, results):
        print "Here's the Error "
        print " " + results[1]

class ScLite:
    def __init__(self, arguments, syllablesOrWords, alignmentFile):
        print '*****'
        print '*****'
        print 'ScLite.py'
        print '*****'
        print '*****'
        self.args = []
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.decoderDir = self.args[0] #directory for decoder output
        self.refDir = self.args[1] #reference file directory
        self.outputDirectory = self.args[2] #Wer directory
        self.alignOut = self.args[3] #output of alignment scripts
        self.alignScript = self.args[4] #alignment script
        self.scliteScript = self.args[5] #sclite script
        self.scliteOut = self.args[6] #output of sclite
        self.singleOrMultiple = self.args[7] #single or multiple audio files
        self.syllablesOrWords = syllablesOrWords #working on syllables or words-->input comes from the early lines of
        DoEverythingTst
        self.alignmentFile = alignmentFile
        twoParts = alignmentFile.split('.')
        self.baseName = twoParts[0] #e.g., bryan1
        self.decoderOut = self.decoderDir + self.baseName + 'decoder' + 'BestWordsTimes.out' #actual name of fan's parser
        output file

        def runRefHypAlign(self):

            cmd = self.alignScript + ' -o ' + self.outputDirectory + ' ' + self.refDir + '/' + self.alignmentFile + ' ' + self.decoderOut + ' ' +
            self.baseName
            print cmd
            try:
                self.runCommands(cmd)
            except CommandError:
                print "Error running genScLiteHypRef"

        def runConcat(self):

            #are we doing a single or multiple audio files?
            if self.singleOrMultiple == "multiple":
                cmd = 'cat ' + self.outputDirectory + '/*_hyp.trn > ' + self.outputDirectory + '/' + self.singleOrMultiple +
                self.syllablesOrWords + 'Allhyp.trn '
                cmd1 = 'cat ' + self.outputDirectory + '/*_ref.trn > ' + self.outputDirectory + '/' + self.singleOrMultiple +
                self.syllablesOrWords + 'Allref.trn '
            else:
                cmd = 'cp ' + self.outputDirectory + '/' + self.baseName + '_hyp.trn ' + self.outputDirectory + '/' + self.baseName +
                self.syllablesOrWords + 'Allhyp.trn '
                cmd1 = 'cp ' + self.outputDirectory + '/' + self.baseName + '_ref.trn ' + self.outputDirectory + '/' + self.baseName +
                self.syllablesOrWords + 'Allref.trn '

            try:
                self.runCommands(cmd)
            except CommandError:

```

```

        print "Error concatenating aligned hypothesis files"
    try:
        self.runCommands(cmd1)
    except CommandError:
        print "Error concatenating aligned reference files"

def runScLite(self):
    if self.singleOrMultiple == "multiple":
        self.baseName = self.singleOrMultiple
        cmd = self.scliteScript + ' -i swb -h ' + self.outputDirectory + '/' + self.baseName + self.syllablesOrWords + 'Allhyp.trn -r '
        cmd = cmd + self.outputDirectory + '/' + self.baseName + self.syllablesOrWords + 'Allref.trn '
        outputFile = self.baseName + self.syllablesOrWords
        cmd = cmd + '> ' + self.outputDirectory + '/' + outputFile
    try:
        self.runCommands(cmd)
    except CommandError:
        print "Error running sclite"
        cmd = 'cp ' + self.outputDirectory + '/' + outputFile + ' ' + self.outputDirectory + '/final/'

    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error copying final sclite files"
        sys.exit(1)

    return self.outputDirectory + '/final/' + outputFile

def runCommands(self,cmd):
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        raise CommandError(results)

```

### I.13. Controls scoring software for Experiment 3

```

import string
import sys
import commands

class CommandError(Exception):
    def __init__(self,results):
        print "Here's the Error "
        print " " + results[1]

class ScLite:
    def __init__(self,arguments,syllablesOrWords,alignmentFile):
        print '*****'
        print '*****'
        print 'ScLite.py'
        print '*****'
        print '*****'
        self.args = []
        argFile = open(arguments, 'r')
        for item in argFile:
            item = string.rstrip(item)
            self.args.append(item)
        argFile.close()
        self.decoderDir = self.args[0] #directory for decoder output
        self.refDir = self.args[1] #reference file directory
        self.outputDirectory = self.args[2] #Wer directory
        self.alignOut = self.args[3] #output of alignment scripts
        self.alignScript = self.args[4] #alignment script
        self.scliteScript = self.args[5] #sclite script
        self.scliteOut = self.args[6] #output of sclite
        self.singleOrMultiple = self.args[7] #single or multiple audio files
        self.syllablesOrWords = syllablesOrWords #working on syllables or words-->input comes from the early lines of
        DoEverythingTst
        self.alignmentFile = alignmentFile
        twoParts = alignmentFile.split('.')
        self.baseName = twoParts[0] #e.g., bryan1
        self.decoderOut = self.decoderDir + self.baseName + 'decoder' + 'BestWordsTimes.out' #actual name of fan's parser
        output file

        def runRefHypAlign(self):

            cmd = self.alignScript + ' -o ' + self.outputDirectory + ' ' + self.refDir + '/' + self.alignmentFile + ' ' + self.decoderOut + ' ' +
            self.baseName
            print cmd
            try:
                self.runCommands(cmd)
            except CommandError:
                print "Error running genScLiteHypRef"

        def runConcat(self):

            #are we doing a single or multiple audio files?
            if self.singleOrMultiple == "multiple":
                cmd = 'cat ' + self.outputDirectory + '/*_hyp.trn > ' + self.outputDirectory + '/' + self.singleOrMultiple +
                self.syllablesOrWords + 'Allhyp.trn '
                cmd1 = 'cat ' + self.outputDirectory + '/*_ref.trn > ' + self.outputDirectory + '/' + self.singleOrMultiple +
                self.syllablesOrWords + 'Allref.trn '
            else:
                cmd = 'cp ' + self.outputDirectory + '/' + self.baseName + '_hyp.trn ' + self.outputDirectory + '/' + self.baseName +
                self.syllablesOrWords + 'Allhyp.trn '
                cmd1 = 'cp ' + self.outputDirectory + '/' + self.baseName + '_ref.trn ' + self.outputDirectory + '/' + self.baseName +
                self.syllablesOrWords + 'Allref.trn '

            try:
                self.runCommands(cmd)
            except CommandError:

```

```

        print "Error concatenating aligned hypothesis files"
    try:
        self.runCommands(cmd1)
    except CommandError:
        print "Error concatenating aligned reference files"

def runScLite(self):
    if self.singleOrMultiple == "multiple":
        self.baseName = self.singleOrMultiple
        cmd = self.scliteScript + ' -i swb -h ' + self.outputDirectory + '/' + self.baseName + self.syllablesOrWords + 'Allhyp.trn -r '
        cmd = cmd + self.outputDirectory + '/' + self.baseName + self.syllablesOrWords + 'Allref.trn '
        outputFile = self.baseName + self.syllablesOrWords
        cmd = cmd + '> ' + self.outputDirectory + '/' + outputFile
    try:
        self.runCommands(cmd)
    except CommandError:
        print "Error running sclite"
        cmd = 'cp ' + self.outputDirectory + '/' + outputFile + ' ' + self.outputDirectory + '/final/'

    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error copying final sclite files"
        sys.exit(1)

    return self.outputDirectory + '/final/' + outputFile

def runCommands(self,cmd):
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        raise CommandError(results)

```

### I.14. Controls program that controls scoring software for Experiment 3

```

from ScLite import *

def scLite():

    #location of the files, scoring type
    audioFileName = '../multiUttDecode/raw/audioFiles.txt'
    scliteArgs = '../Arguments/sclite.args'
    syllablesOrWords = 'syllables'
    size = '4' #current ngram size
    finalDest = '../Output/Wer/final/finalSave/OneBest/multiplesyllables' + size
    alignFiles = []
    audio = open(audioFileName, 'r')
    for fname in audio:
        fname = str.rstrip(fname)
        alignName = fname + '.align'
        alignFiles.append(alignName)
    scoringData = scliteStep(scliteArgs,syllablesOrWords,alignFiles)
    scoringDataFinal = scoringData + size
    cmd = 'cp ' + scoringData + ' ' + scoringDataFinal
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error renaming scoring data"
        sys.exit(1)
    cmd = 'cp ' + scoringDataFinal + ' ' + finalDest
    print cmd
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error copying final scoring data"
        sys.exit(1)

def scliteStep(scliteArgs,syllablesOrWords,alignFiles):
    for alignmentFile in alignFiles:
        fileName = str.split(alignmentFile, '.')
        print " "
        print "Running sclite on recognizer's guess for: " + str(fileName[0])
        sclite = ScLite(scliteArgs,syllablesOrWords,alignmentFile)
        sclite.runRefHypAlign()
        sclite.runConcat()
        scoringData = sclite.runScLite()
    return scoringData

def main():
    scLite()
main()

```

## I.15. Controls syllabification software to syllabify the transcript. Used in Experiment 2 and 3

```

import string
import commands
import re
import sys
from sets import Set

#Contains the methods necessary to syllabify a transcript
#Creates some useful intermediate files along the way
class Syllabify:
    """Contains methods necessary to syllabify a transcript"""

    def __init__(self, transcriptFile, repositoryIn, syllabifyArgs):
        """
        Constructor function. Instance of Syllabify is created. Names of files used in this class
        set to variable names.
        arg1: name of transcript file
        arg2: directory where input and output data is stored
        arg3: argument file
        """
        print '*****'
        print '*****'
        print 'syllabify.py'
        print '*****'
        print '*****'
        argList = self.readArgs(syllabifyArgs)

        #names of files used in this class
        self.repository = repositoryIn + '/' #directory where data is stored relative to current directory
        self.syllableDictionary = {} #syllable dictionary created in makeSyllableDict
        self.transcript = transcriptFile

        self.repositoryS = argList[0] #directory where data is stored relative to the where the syllabifier script is stored
        self.syllabifier = argList[1] #NIST syllabifier
        self.NISTtransDict = argList[2] #words followed by their lower case NIST transcriptions, one to a line
        self.SONICtransDict = argList[3] #words followed by their lower case SONIC transcriptions, one to a line
        self.NISTtrans = argList[4] #nist transcribed words, one to a line. input to syllabifier
        self.syllWords = argList[5] #output of NIST syllabifier
        self.syllTrans = argList[6] #syllabified transcript
        self.phonFile = argList[7] #file containing the NIST phonetic symbol set
        self.syllableDictFile = self.repository + argList[8] #stored version of the syllable dictionary created in makeSyllableDict
        self.words = self.repository + argList[9]

        infile = open(self.repository + self.NISTtransDict, 'r') #output of class Transcribe
        outfile = open(self.repository + self.NISTtrans, 'w') #NIST transcribed words
        for line in infile:
            line = line.replace("\n", "") #eliminate eol character
            dictList = string.split(line, ":")
            str = ""
            for elt in dictList[1]:
                str = str + elt
            str = str + '\n'
            outfile.write(str)
        infile.close
        outfile.close

        print 'file of NIST word transcriptions created: ' + self.repository + self.NISTtrans

    def readArgs(self, syllabifyArgs):
        #print "Reading syllabification argument file"
        argFile = open(syllabifyArgs)
        argList = []

```

```

for arg in argFile:
    arg = string.rstrip(arg, '\n')
    argList.append(arg)
argFile.close()
#print ""
#print ""
return argList

def syllabify(self):
    """
    syllabifies each word in the initial transcript, writing all to a file
    """

    cmd = "." + self.syllabifier + " " + self.repositoryS + self.NISTtrans + " " + self.repositoryS + self.syllWords + " " +
self.phonFile

    print "Executing NIST syllabifier"
    print cmd
    print " "
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error "
        print results[0]
        print results[1]

    allSylls = self.cleanSyllabifiedTranscript()

    #go back to the SONIC phone set
    allSonicSylls = self.returnToSonic(allSylls)

    self.syllableDictionary = self.makeSyllableDict(allSonicSylls)

    #delete temporary files
    #cmd = 'rm ' + self.repository + self.syllWords
    #commands.getoutput(cmd)

def cleanSyllabifiedTranscript(self):
    """
    returns list of syllabified words, where each syllabified word is a list of the syllables of that word
    """
    infile = open(self.repository + self.syllWords, 'r')
    str = ""
    allSylls = []
    for line in infile:
        syllableLine = re.findall(r'Basic pron is /# \[.+\]', line) #find the line containing the most basic syllabification output by
NIST
        if len(syllableLine) > 0:
            str = syllableLine[0]
            syllables = re.findall(r'\[.+\]', str) #extract the syllabification from the line
            str = syllables[0]
            str = re.sub("\\d", "", str) #eliminate stress markings
            str = str.replace(' ', '') #eliminate resulting double space
            str = str.replace(' ', '_') #insert underscore between phones
            str = str.replace('_[_]', ' ') #insert space between syllables
            str = str.replace('[', '') #eliminate left bracket
            str = str.replace(']', '') #eliminate right bracket
            syllList = string.split(str, " ") #split the string into list of syllables
            allSylls.append(syllList)
    infile.close()
    return allSylls

def returnToSonic(self, allSylls):

```



```

#file of words followed by their lc SONIC transcriptions
infile = open(self.repository + self.SONICtransDict)

#transform the file into a list of words, where the phones in each word are separated by underscores
SONICList = []
for line in infile:
    line = string.rstrip(line, '\n')
    lineList = string.split(line, ':')
    word = string.replace(lineList[1], ' ', '_')
    SONICList.append(word)

#allSylls is a list of words, where each list is a list of its syllables
#transform allSylls into a list of words, where the phones in the words are separated by underscores
#and the syllables are separated by asterisks
NISTList = []
for word in allSylls:
    newWord = ""
    for syll in word:
        newWord = newWord + syll + '*'
    newWord = string.rstrip(newWord, '*')
    NISTList.append(newWord)

i = 0
newAllSylls = []
for i in range(0, len(SONICList), 1):
    #mark syllables in the SONIC transcription of words with asterisks
    syllWordSONIC = self.NISTtoSONIC(SONICList[i], NISTList[i])
    #split the words into a list of syllables
    syllList = string.split(syllWordSONIC, '*')
    #add each word, represented as a list of syllables, to a new list of lists, where each list is a list of syllables in a word
    newAllSylls.append(syllList)

return newAllSylls

def NISTtoSONIC(self, SONICStr, NISTStr):

    #Store positions of syllable markers
    pos = -1
    syllBreaks = set()
    newNIST = ""
    for i in range(0, len(NISTStr), 1):
        if NISTStr[i] == '_':
            pos = pos + 1
        if NISTStr[i] == '*':
            pos = pos + 1
            syllBreaks.add(pos)

    #Place syllable markers in the SONIC string
    SONICList = string.split(SONICStr, '_')

    SONICStrOut = ""
    for i in range(0, len(SONICList), 1):
        if i in syllBreaks:
            SONICStrOut = SONICStrOut + SONICList[i] + '*'
        else:
            SONICStrOut = SONICStrOut + SONICList[i] + '_'

    #the SONIC transcription of a word but with syllables marked by asterisks
    SONICStrOut = string.rstrip(SONICStrOut, '_')

    return SONICStrOut

def makeSyllableDict(self, allSylls):

```

```

"""
Creates a dictionary, indexed by word, of syllable lists.
<word> [[syll1], [syll2], ... , [syln]] where = "

syllk are syllables of the word, encoded as underscore
delimited phones.
The following technique works because there are exactly as many words in self.SONICtransDict as there are
syllabifications in allSylls and they're in the same order.
"""

infile = open(self.repository + self.SONICtransDict, 'r')
syllableDict = {}
i = 0
for line in infile:
    lineList = string.split(line,':')
    syllableDict[lineList[0]] = allSylls[i] #add word followed by syllable list to the dictionary
    i = i + 1
infile.close()
#print "Syllable dictionary created"
return syllableDict

def writeDict(self):
    """
    Saves the syllable dictionary created in makeSyllableDict.
    Format is identical, except the word is separated from the syllable
    lists by a colon.
    """

    outfile = open(self.syllableDictFile, 'w') #:dictionary file
    str1 = ""
    for key in self.syllableDictionary:
        tempList = self.syllableDictionary[key]
        str1 = str1 + key + ':'
        for elt in tempList:
            str1 = str1 + elt + ' '
        str1 = string.rstrip(str1)
        outfile.write(str1)
        outfile.write("\n")
        str1 = ""
    outfile.close
    str = "Syllable dictionary saved to: " + self.syllableDictFile
    print str

def readDict(self):
    """
    Restores syllable dictionary from a file.
    File was created in writeDict
    """

    infile = open(self.syllableDictFile, 'r') #:file created in method, writeDict
    syllableDict = {} #:syllable dictionary as created in method, makeSyllableDict
    for line in infile:
        line = line.replace("\n","") #eliminate eol character
        dictList = string.split(line,":")
        str1 = ""
        for elt in dictList[1]:
            str1 = str1 + elt
        syllList = string.split(str1, " ")

        syllableDict[dictList[0]] = syllList
    infile.close

    self.syllableDictionary = syllableDict
    str1 = "Syllable dictionary restored from file: " + self.syllableDictFile
    #print str1

```

```

def produceSyllabifiedTranscript(self):
    """
    Writes a syllabification of the transcript used as input to the constructor.
    If the transcript has this format
    <word1> <word2> ... <wordn>, the syllabification is <syll1a> ... <syll1n> <syll2a> ... <syll2n> ...
    where each syllij is the jth syllable of the ith word
    """

    infile = open(self.transcript, 'r')
    outfile = open(self.repository + self.syllTrans, 'w')

    tempFile = []
    for line in infile:
        #line = string.rstrip(line, '\r\n')
        listLine = string.split(line)

        strng = ""
        tempRec = []
        for strng in listLine:
            strng = string.lower(strng)
            syllables = self.syllableDictionary[strng]
            for item in syllables:
                tempRec.append(item)
                tempRec.append(' ') #See comment below for issues
            tempFile.append(tempRec)
        self.removeTrailingSpace(tempFile, outfile)

    infile.close()
    outfile.close()

    print "Syllabified transcript created: " + self.repository + self.syllTrans

def removeTrailingSpace(self, tempFileIn, outFileIn):
    """syll.in
    Removes the trailing space which would get counted as a syllable.
    Spaces are inserted after each syllable in produceSyllables. This is a problem for
    monosyllabic words and for the last syllable in a line. The result is an extra space at the
    end of each line. It could be removed as the line is being built, but this
    is complicated since the number of syllables in a line is not predictable
    from the number of words. Partially constructing output in a list,
    reading the list, and removing the space item by item seemed simple (and
    not that much more compute intensive since the alternative method requires
    reading each item twice, once to determine the number of syllables and once to
    construct the output line.
    """

    tempFile = tempFileIn #:list from produceSyllables
    outFile = outFileIn #:syllabified transcript

    for rec in tempFile:
        strng = ""
        for fld in rec:
            strng = strng + fld
        strng = string.rstrip(strng)
        strng = strng + '\n'
        outFile.write(strng)

def cleanUp(self):
    cmd = 'rm ' + self.repository + self.NISTtransDict
    cmd1 = 'rm ' + self.repository + self.SONICtransDict
    cmd2 = 'rm ' + self.repository + self.NISTtrans
    commands.getoutput(cmd)

```

```
commands.getoutput(cmd1)
commands.getoutput(cmd2)
#print "Temporary files deleted"
```

## I.16. Controls syllabification software to syllabify reference files. Used in Experiment 2 and 3

```

import string
import sys
from Transcribe import *
from Syllabify import *
class SyllabifyRefFiles:
    """Contains methods necessary to syllabify reference files"""
    def __init__(self, alignFile, noRefs, ref):
        self.outputDir = "../Output/Syllabify"
        self.syllabifyArgs = "../Arguments/syllabify.args"
        self.syllabifiedRefFile = "syllTrans.out"
        fileIn = open(alignFile, 'r')
        refOut = open(ref, 'w')
        fileOut = open(noRefs, 'w')
        lineList = []
        for line in fileIn:
            lineList = string.split(line, ' ')
            lineStr = str.lstrip(lineList[1])
            refStr = lineList[0] + ' '
            refOut.write(refStr)
            refOut.write("\n")
            fileOut.write(lineStr)
        fileOut.close()
        fileIn.close()
        refOut.close()

    def transcribeStep(self, noRefs):
        print "Transcription Step"
        trans = Transcribe(noRefs, self.outputDir)
        trans.transcribe()
        print ""
        print ""

    def syllabifyStep(self, noRefs):
        print "Syllabification Step"
        syll = Syllabify(noRefs, self.outputDir, self.syllabifyArgs)
        syll.syllabify()
        syll.writeDict()
        syll.produceSyllabifiedTranscript()
        syll.cleanUp()
        print ""
        print ""

    def makeAlignFile(self, refs, syllAlignFile):
        refsIn = open(refs, 'r')
        syllables = str(self.outputDir + '/' + self.syllabifiedRefFile)
        syllsIn = open(syllables, 'r')
        syllsOut = open(syllAlignFile, 'w')
        sylList = []
        refList = []
        for line in syllsIn:
            line = str.rstrip(line)
            sylList.append(line)
        for line in refsIn:
            line = str.rstrip(line)
            refList.append(line)
        for ctr in range(0, len(refList), 1):
            syllsOut.write(refList[ctr])
            syllsOut.write(' ')
            #change to upper case, needed for sclite
            upperStr = string.upper(sylList[ctr])
            #remove underscore separating phones in a syllable, needed for sclite
            upperStr = upperStr.replace('_', ' ')
            syllsOut.write(upperStr)
            syllsOut.write("\n")

```

```
syllsIn.close()
syllsOut.close()
refsIn.close()
print "Alignment File Written: " + syllAlignFile

def cleanUp(self, refs, noRefs):
    cmd = 'rm ' + refs
    cmd1 = 'rm ' + noRefs
    commands.getoutput(cmd)
    commands.getoutput(cmd1)
    print "Temporary files deleted"
```



```

        utterance = string.rstrip(utterance)
        utterance = string.upper(utterance) #transform to UC. Necessary for log prob scoring
        words.write(utterance)
        words.write("\n")
    syllables.close()
    words.close()
    #print "Syllables from decoder transformed to words: " + self.args[2]

def parse (self, strInput):
    #print strInput
    m_lstResult = []
    m_lstInput = strInput.upper().split()
    self._initChart(m_lstInput)
    self.chart = []
    m_lstTmpAgenda = self.agenda
    self.agenda = []
    for m_tplItem in m_lstTmpAgenda:
        if m_tplItem[0]==0:
            self.chart.append(m_tplItem)
        else:
            self.agenda.append(m_tplItem)
    while (1):
        #print self.chart
        if len(self.chart)==0: break
        m_tpCurrentItem = self.chart.pop(0)
        if m_tpCurrentItem[1]==len(m_lstInput):
            m_lstResult.append(m_tpCurrentItem[2])
            continue
        for m_tplItem in self.agenda:
            if m_tpCurrentItem[1]==m_tplItem[0]:
                m_lstWord = []
                for m_strWord in m_tpCurrentItem[2]:
                    m_lstWord.append(m_strWord)
                for m_strWord in m_tplItem[2]:
                    m_lstWord.append(m_strWord)
                if not self._prune(m_lstWord):
                    self.chart.append((m_tpCurrentItem[0],m_tplItem[1],m_lstWord))
    return self._sortOutput(m_lstResult)

def _prune(self, lstWord):
    #print 'prune unreasonable outputs'
    m_strSent = '|' + '|' .join(lstWord) + '|'
    for m_strSub in self.lstSavedWords:
        m_strSub = '|' + m_strSub + '|'
        if m_strSent.find(m_strSub)!=-1:
            return 1
    return 0
"""
old prune
def _prune(self, lstWord):
    m_strSent = '|' .join(lstWord)
    for m_strSub in self.lstSavedWords:
        if m_strSent.find(m_strSub)!=-1:
            return 1
    return 0
"""

def _sortOutput(self, lstResult):
    """ sort the output based on how many syllables are contained"""
    def _genKey4Sort(lstSent):
        m_iCountSyl = 0
        for m_strItem in lstSent:
            if m_strItem.isupper():
                m_iCountSyl +=1
        return(m_iCountSyl)
    lstResult.sort(key=_genKey4Sort)
    return lstResult

```



```

def _findWord(self, lstWordSyllable):
    m_strIndex = ''.join(lstWordSyllable)
    return self.dictSylWord.get(m_strIndex, "")

def _initChart(self, lstSentSyllable):
    self.agenda = []
    self.lstSavedWords = []
    for i in range(len(lstSentSyllable)):
        self.agenda.append((i,i+1,[lstSentSyllable[i]]))
    for i in range(1, len(lstSentSyllable)+1):
        for j in range(0, len(lstSentSyllable)-i+1):
            m_lstWord = self._findWord(lstSentSyllable[j:i+j])
            if m_lstWord!="":
                self.lstSavedWords.append(''.join(lstSentSyllable[j:i+j]))
                for m_strWord in m_lstWord:
                    self.agenda.append((j,i+j,[m_strWord]))

if __name__ == '__main__':
    """
    Usage: sylWord.py dictFile
    """
    m_objSylWord = CSylWord(sys.argv[1])
    for line in sys.stdin:
        results = m_objSylWord.parse(line)
        for r in results:
            print ''.join(r)

```

## I.18. Controls transcription software. Used in Experiments 2 and 3

```

import string
import commands
import sys
import re

#Contains the methods necessary to transcribe a transcript using both the SONIC and NIST phone sets
class Transcribe:
    def __init__(self, transcriptFile, repositoryIn):
        """
        arg1: original transcript
        arg2: directory where output data is stored
        Constructor function. Instance of Transcribe is created. Each word in transcriptFile is saved to a file,
        one word per line. Use of a dictionary eliminates duplicate words. The result
        is a file where each word in the transcript appears only once.
        """
        print '*****'
        print '*****'
        print 'transcribe.py'
        print '*****'
        print '*****'

        #Names of files used in this class
        self.repository = repositoryIn          #directory where the files are stored
        self.wordFile = repositoryIn + '/words.out'      #words in transcript, one to a line
        self.wordSoundFile = repositoryIn + '/wordsSounds.out' #words in transcript with phonetic transcriptions, one to a line
        self.SONICtrans = repositoryIn + '/SONICtransDict.out' #words followed by their SONIC transcriptions, one to a line
        self.NISTtrans = repositoryIn + '/NISTtransDict.out' #words followed by their lower case NIST transcriptions, one to a
line
        infile = open(transcriptFile, 'r')
        outfile = open(self.wordFile, 'w')
        dict = {}
        for line in infile:
            listLine = string.split(line)
            for i in range(0, len(listLine), 1):
                dict[listLine[i]] = listLine[i]

        str = ""
        for key in dict:
            str = str + key
            str = str + ""
            str = ""
            outfile.write(key)
            outfile.write("\n")
        print "Word file written: " + self.wordFile

        infile.close();
        outfile.close();

    def transcribe(self):
        """
        invoked from: makeSylsTst
        Calls all methods necessary to transcribe.
        """
        #uses SONIC' sspell to transcribe the input
        self.makeSoundFile()

        #creates files with words followed by transcriptions, one file does SONIC, the other NIST. See the constructor for file
names
        self.makeTransDict()

        #delete files that are no longer necessary
        cmd = 'rm ' + self.wordFile
        cmd1 = 'rm ' + self.wordSoundFile
        #commands.getoutput(cmd1)
        #commands.getoutput(cmd)

```

```

#print "Temporary files deleted"

def makeSoundFile(self):
    """
    invoked from: transcribe
    Creates a file of phonetic transcriptions of the words found in the language transcript.
    Transcriptions are gotten from a single call to SONIC's sspell with the file created in the constructor as input
    The output is one word/transcription pair per line: <word> <transcription>
    Some words have multiple transcriptions, so will appear on more than one line.
    -lts option tells spell to do a letter to sound transcription if the word is not in its vocabulary
    wordFileIn = wordFile from the constructor
    wordSoundFile = word/transcription file
    Usage: <instanceVariable>.makeSoundFile(<fileOfWords>, <wordTranscriptionFile>)
    """

    cmd = 'sspell -lts -fl ' + self.wordFile + ' > ' + 'temp' #temp is the output of sspell

    commands.getoutput(cmd)
    #remove parentheses from words with multiple transcriptions: e.g., from(2) --> from
    infile = open('temp', 'r')
    outfile = open(self.wordSoundFile, 'w')
    for line in infile:
        cleanList = re.findall(r'[^\d\']', line)
        str = ""
        for item in cleanList: #transform list to string
            str = str + item
        outfile.write(str)
    infile.close()
    outfile.close()
    #self.cleanSoundFile() #creates a clean version of wordSoundFile

    print "Word transcription file written: " + self.wordSoundFile

    cmd = 'rm temp'
    commands.getoutput(cmd)

    """
    no longer used because -lts option is available
    Formerly used to extract words from sspell errors that required
    letter to sound transcription
    invoked from: makeSoundFile
    Transcribes words not found in the sspell dictionary
    def cleanSoundFile(self):
        infile = open(self.wordSoundFile, 'r')
        outfile = open('cleanWordSoundFile', 'w')

        for line in infile:
            pos = string.find(line, "ERROR: no pronunciation") #sspell error--found in position 0
            if pos != 0:
                outfile.write(line)
            else:
                line = self.doLetterToSound(line)
                outfile.write(line)
        infile.close()
        outfile.close()
        cmd = 'cp ' + 'cleanWordSoundFile' + self.wordSoundFile #replace the wordSoundFile with
        commands.getoutput(cmd)
        cmd = 'rm cleanWordSoundFile'
        commands.getoutput(cmd)

    def doLetterToSound(self, line):
        cleanLine = re.findall(r'[A-Z]+\s', line)
        cleanLine = re.findall(r'[A-Z]+\s', cleanLine[0])
        str = ""
        for item in cleanLine: #transform list to string

```

```

    str = str + item

cmd = 'spell -lts ' + str + ' > ' + 'NewTemp' #run spell in letter to sound mode on the word that generated the error
commands.getoutput(cmd)

#read the word from the output file generated by spell and return it to the calling function
infile = open('NewTemp', 'r')
for onlyLine in infile:
    line = onlyLine
infile.close()
cmd = 'rm ' + 'NewTemp' #remove temporary file
commands.getoutput(cmd)
return line
'''

def makeTransDict(self):
    '''
    invoked from: transcribe
    Creates a dictionary indexed by word, of SONIC transcriptions
    Transforms these to the NIST set
    Calls functions to write the two dictionaries to files
    '''
    #Special Notes
    #Each word is assumed to have a single transcription. This is accomplished by reading the output of makeSoundFile into
    a list of tuples,
    #where the tuples are word/transcription pairs treated as strings.
    #The list is then reversed (so that the lowest scored transcriptions from spell precede the highest).
    #The sublists are then entered into a dictionary. Since when duplicates arise, dictionary overwrites what was previously
    stored with the duplicate,
    #this has the effect of keeping only the highest scored transcription in the dictionary
    #Finally, everything is transformed to lower case as is demanded by the NIST syllabifier

    infile = open(self.wordSoundFile, 'r')
    tempList = []
    '''
    better code is below
    str1 = ""
    str2 = ""
    for line in infile:
        #extract word & transcription, transform them both into strings, and add them to tempList
        wordType1 = "[A-Z]+"
        wordType2 = "[A-Z]+\\"[A-Z]" #account for contractions like "I'd" or "I've" or "goin'"
        word = re.findall(wordType1,line)
        if len(word) == 0:
            word = re.findall(wordType2,line)
        for item in word:
            str1 = str1 + item
            str1 = string.lower(str1) #transform to lower case for NIST syllabifier
            str1 = string.rstrip(str1) #remove trailing space captured in the regex search above

            transcription = re.findall(r' [A-Z]+', line) #extract transcription
            for item in transcription:
                str2 = str2 + item
                str2 = string.lower(str2) #transform to lower case for NIST syllabifier
            t = str1,str2 #make a tuple
            tempList.append(t) #put tuple in a list

    str1 = ""
    str2 = ""
    infile.close()
    '''
    for line in infile:
        word = ""
        transc = ""
        line = str(line)
        lineList = line.split()

```

```

word = lineList[0]
word = string.lower(word)
str1 = ""
for ctr in range(1, len(lineList), 1):
    str1 = str1 + ' ' + lineList[ctr]
transc = str1.lstrip()
transc = string.lower(transc)
lineTuple = word,transc
tempList.append(lineTuple)

tempList.reverse() #reverse list so order of creation or pronunciations is backwards, making the best choice last

SONICdict = {} #put items in a dictionary, overwriting less good pronunciation choices with better choices
for item in tempList:
    SONICdict[item[0]] = item[1]

#create files containing NIST and SONIC transcriptions
self.transFormToNist(SONICdict)

def transFormToNist(self, SONICdict):
    """
    Transforms the transcriptions stored in the dictionary from those found in SONIC to those found in NIST
    Writes both transcriptions to files.
    """
    sonicFile = open(self.SONICtrans, 'w')
    nistFile = open(self.NISTtrans, 'w')
    lookup = {"ae":"ah", "ng":"nx", "pd":"p", "bd":"b", "td":"t", "dd":"d",
              "kd":"k", "gd":"g"}

    str1 = ""
    str2 = ""
    for key in SONICdict:
        listLine = string.split(SONICdict[key])
        for item in listLine:
            if item in lookup:
                str1 = str1 + " " + lookup[item]
            else:
                str1 = str1 + " " + item

    str1 = string.lstrip(str1)
    str2 = string.lstrip(SONICdict[key])

    sonicStr = key + ':' + str2
    nistStr = key + ':' + str1
    sonicFile.write(sonicStr + '\n')
    nistFile.write(nistStr + '\n')
    str1 = ""
    str2 = ""

    sonicFile.close()
    nistFile.close()
    print ("SONIC and NIST transcriptions files written: " + self.SONICtrans + ", " + self.NISTtrans)

```

## I.19. Various one-off utilities used in the experiments

```

import string
import random
import math
import commands
import sys
import re
from numpy import *
from pylab import *
from Transcribe import *

#contains various one-off utilities
class Utilities:
    def __init__(self):
        print "Instance of utilities created"

    def syllabifyAlignFiles(self, inp,out,ref):
        fileIn = open(inp,'r')
        refOut = open(ref,'w')
        fileOut = open(out,'w')
        lineList = []
        for line in fileIn:
            lineList = string.split(line,'')
            lineStr = str.lstrip(lineList[1])
            refStr = lineList[0] + ' '
            refOut.write(refStr)
            refOut.write('\n')
            fileOut.write(lineStr)
        fileOut.close()
        fileIn.close()
        refOut.close()

    #pastes file 2 to file 1
    def pasteFiles(self, f1, f2, f3):
        file1 = open(f1, 'r')
        file2 = open(f2, 'r')
        file3 = open(f3, 'w')

        for line in file1:
            line = str.rstrip(line)
            file3.write(line)
            file3.write('\n')
        file1.close()
        for line in file2:
            line = str.rstrip(line)
            file3.write(line)
            file3.write('\n')
        file2.close()
        file3.close()

    """
    determines if the collection of words in myWordsIn is a
    subset of the words in lexWordsIn
    Written specifically to make sure that the words I'm using
    in the transcript appears in the file that maps words to their
    phonetic transcriptions
    """

    def checkLexicon(self, myWordsIn, lexWordsIn):
        sortedFileIn = self.sortMyWordsIn(myWordsIn)
        lexDict = self.createLexDict(lexWordsIn)
        outOfLexTmp = self.findOutOfLexWords(sortedFileIn, lexDict)

    """
    creates two files in syllData/lex:
    SONICtransDict.out
    NISTtransDict.out

```

```

Only the first is used here. Both are ultimately deleted
The first has to be cleaned up so that it is in the same format
as lexWordsIn, to which it will be ultimately appended
'''
trans = Transcribe(outOfLexTmp, 'syllData/lex')
trans.transcribe()
print 'transcribe complete'

self.cleanUpTranscript(lexWordsIn)

'''
cmd = 'rm ' + sortedFileIn
results = commands.getstatusoutput(cmd)
cmd = 'rm ' + outOfLexTmp
results = commands.getstatusoutput(cmd)
cmd = 'rm ' + lex/NISTtransDict.out
results = commands.getstatusoutput(cmd)
cmd = 'rm ' + lex/SONICtransDict.out
results = commands.getstatusoutput(cmd)
'''

def cleanUpTranscript(self, lexWordsIn):
    cleanTranscript = open('syllData/lex/SONICtransDict.out')
    cleanList = []
    for line in cleanTranscript:
        line = string.rstrip(line)
        newLine = string.split(line, ':')
        cleanLine = str.upper(newLine[0]) + ' ' + str.upper(newLine[1])
        cleanList.append(cleanLine)

    finalLex = open(lexWordsIn, 'a')
    for item in cleanList:
        finalLex.write(item)
        finalLex.write("\n")
    finalLex.close()
    print 'Lexicon Written: ' + lexWordsIn
    #to here-->transfrom the result from lower to upper case

def findOutOfLexWords(self, sortedFileIn, lexDict):
    sortedFile = open(sortedFileIn, 'r')
    outOfLexTmp = 'syllData/outOfLexTmp'
    outOfLexFile = open(outOfLexTmp, 'w')
    for word in sortedFile:
        word = string.rstrip(word)
        if not word in lexDict:
            outOfLexFile.write(word)
            outOfLexFile.write("\n")
    sortedFile.close()
    outOfLexFile.close()
    return outOfLexTmp

def createLexDict(self, lexWordsIn):
    lexDict = {}

    lexWords = open(lexWordsIn, 'r')
    for line in lexWords:
        line = string.split(line, ' ')
        #some words appear more than once: an(2). This removes parens
        #duplicatates are removed in dictionary creation
        cleanList = re.findall(r'^[^\d)]', line[0])

        str = ""
        for item in cleanList: #transform list to string
            str = str + item
        lexDict[str] = str
    lexWords.close()
    return lexDict

```

```

'''
Creates a sorted file from transcript, one word to a line
Output file is t3. Ultimately will be deleted
'''
def sortMyWordsIn(self, myWordsIn):
    t1 = "syllData/t1"
    t2 = "syllData/t2"
    t3 = "syllData/t3"
    myWordsRaw = open(myWordsIn, 'r')
    myWordsCooked = open(t1, 'w')
    for line in myWordsRaw:
        line = string.split(line, ' ')
        for word in line:
            word = string.rstrip(word)
            myWordsCooked.write(word)
            myWordsCooked.write('\n')
    myWordsRaw.close();
    myWordsCooked.close()

    cmd = 'sort -n < ' + t1 + ' | uniq > ' + t2
    results = commands.getstatusoutput(cmd)

    cmd = 'grep [A-Z] ' + t2 + ' > ' + t3
    results = commands.getstatusoutput(cmd)

    cmd = 'rm ' + t1
    results = commands.getstatusoutput(cmd)
    cmd = 'rm ' + t2
    results = commands.getstatusoutput(cmd)
    return t3

#syllableDict: map of words and component syllables
#phoneDict: map of syllables and component phones
def createPhoneDict(self, syllableDict, phoneDict):
    syllDict = open(syllableDict, 'r')
    phDict = open(phoneDict, 'w')
    for line in syllDict:
        line = string.rstrip(line)
        line = string.split(line, ':')
        strng = line[1]
        line = string.split(strng)
        phStr = ""
        for syll in line:
            phStr = phStr + syll
            phone = string.split(syll, '_')
            for item in phone:
                phStr = phStr + ' ' + item
            phDict.write(phStr)
            phStr = ""
            phDict.write("\n")
    syllDict.close()
    phDict.close()

    #now, remove duplicates
    cmd = 'sort -n < ' + phoneDict + ' | uniq > tmp'
    cmd1 = 'cp tmp ' + phoneDict
    cmd2 = 'rm tmp'
    results1 = commands.getstatusoutput(cmd)
    results2 = commands.getstatusoutput(cmd1)
    results3 = commands.getstatusoutput(cmd2)

    results = results1[0] + results2[0] + results3[0]
    if results > 0:
        print "error removing duplicates from phone dictionary"

```



```

    sys.exit(1)
    print "Phone dictionary created"

#input is the syllabified transcript or the original transcript
#Creates a file containing syllables and word tokens along with their numbers. This is necessary to generate a histogram
#itemIndicator is set to Syllables or Words, whichever is to be counted
def countItems(self, counts, input, itemIndicator):
    #put each item on its own line
    #sort them internally
    #remove duplicates
    #put them in reverse order and write to a temporary file
    #remove the underscore character that links phones in a syllable
    #write output file
    #remove temporary file
    cmd = 'tr " " "\n" < ' + input + ' | sort | uniq -c | sort -rn > ' + counts
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in word counting"
        sys.exit(1)

    if itemIndicator == "Syllables":
        cmd = 'tr "_ " " < ' + counts + ' > tmp'
        results = commands.getstatusoutput(cmd)
        if results[0] != 0:
            print "Error in first part of syllable counting "
            sys.exit(1)
        cmd = 'cp tmp ' + counts
        results = commands.getstatusoutput(cmd)
        if results[0] != 0:
            print "Error in second part of syllable counting "
            sys.exit(1)
        cmd = 'rm tmp'
        results = commands.getstatusoutput(cmd)
        if results[0] != 0:
            print "Temp file not deleted"
            sys.exit(1)

#using the file syllableCounts generated in findSingles, makes a histogram
def histogram(self, countFile, bins):
    counts = open(countFile, 'r')
    stuff = []
    for rec in counts:
        num = re.findall(r'[0-9]+', rec)
        stuff.append(int(num[0]))
    counts.close()
    histArray = array(stuff)
    hist(histArray, bins)
    xlabel("Bin Sizes")
    ylabel("Syllables per Bin")
    title("Syllable or Word Histogram")
    show()

def cleanTranscript(self, fileIn, fileOut):
    input = open(fileIn, 'r')
    output = open(fileOut, 'w')
    for line in input:
        line = str(line)
        line = line.replace('&', ' and ')
        line = line.replace('-', ' ')
        line = line.replace('1', ' one ')
        line = line.replace('2', ' two ')
        line = line.replace('3', ' three ')
        line = line.replace('4', ' four ')
        line = line.replace('5', ' five ')
        line = line.replace('6', ' six ')
        line = line.replace('7', ' seven ')

```

```

        line = line.replace('8', ' eight ')
        line = line.replace('9', ' nine ')
        line = line.replace('0', ' oh ')
        line = line.replace('20/20', 'twenty twenty')
        line = line.upper()
        output.write(line)
'''
fileIn is output of countItems, above. fileOut divides the
number of occurrences of each type count and divides by the tokenCount.
the resulting file contains the probabilities, along with the token counts
'''
def probabilities(self, fileIn, fileOut, corpus):
    #Get the number of tokens in the corpus
    wcFile = 'tmp'
    cmd = 'wc ' + corpus + ' > ' + wcFile
    wcFile = 'tmp'
    results = commands.getstatusoutput(cmd)
    if results[0] != 0:
        print "Error in token count"
        sys.exit(1)
    input = open(wcFile, 'r')
    for line in input:
        line = line.lstrip()
        lineList = line.split()
        tokenCount = lineList[1]
        print "token count: " + str(tokenCount)
    input.close()

    input = open(fileIn, 'r')
    output = open(fileOut, 'w')

    probList = []
    for line in input:
        line = line.lstrip()
        lineList = line.split()
        lineListZeroStr = str(lineList[0])
        prob = float(lineListZeroStr)/float(tokenCount)
        lineList.append(str(prob))
        probList.append(prob)
        str1 = ""
        for item in lineList:
            str1 = str1 + str(item)
            str1 = str1 + ' '
        str1 = str1 + '\n'
        output.write(str1)

    #use NumPy to compute probability statistics
    numpyArray = array(probList)
    '''
    mn = mean(numpyArray)
    #mn = around(mn,2) #round to two decimals
    md = median(numpyArray)
    #md = around(md,2)
    st = std(numpyArray)
    #st = around(st, 2)
    '''
    entropy = 0.0
    for item in numpyArray:
        entropy = entropy + (-1 * item * log2(item))
    entropyRate = entropy / len(numpyArray)
    #output.write("Mean Probability: " + str(mn) + "\n")
    #output.write("Median Probability: " + str(md) + "\n")
    #output.write("Standard Deviation: " + str(st) + "\n")
    output.write("Total Entropy: " + str(entropy) + "\n")
    output.write("Entropy Rate: " + str(entropyRate) + "\n")

```

```

output.close()
input.close()

cmd = 'rm ' + wcFile
results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print "Token count file not deleted"
    sys.exit(1)

cmd = 'rm ' + fileIn
results = commands.getstatusoutput(cmd)
if results[0] != 0:
    print "Type count file not deleted"
    sys.exit(1)

"removes lexical stress markers"
def noStress(self, fileIn, fileOut):
    input = open(fileIn, 'r')
    output = open(fileOut, 'w')
    s = frozenset(['0', '1', '2', '3', '\'])
    for line in input:
        newLine = ""
        for ch in line:
            if ch not in s:
                newLine = newLine + ch
        lineLst = string.split(newLine)
        newLine = ""
        for item in lineLst:
            newLine = newLine + item + ' '
        newLine = newLine.lstrip()
        newLine = newLine.rstrip()
        output.write(newLine)
        output.write('\n')

```

## Appendix J Word-Transcription Lexicon

A AX  
 ABBREVIATED AX B R IY V IY EY DX AX DD  
 ABBREVIATION AX B R IY V IY EY SH AX N  
 ABBREVIATIONS AX B R IY V IY EY SH AX N Z  
 ABLE EY B AX L  
 ABOUT AX B AW TD  
 ABOVE AX B AH V  
 ACCEPT AE K S EH PD TD  
 ACCOMMODATE AX K AA M AX D EY TD  
 ACCOMMODATIONS AX K AA M AX D EY SH AX N Z  
 ACCOMPANYING AX K AH M P AX N IY IX NG  
 ACROSS AX K R AO S  
 ACTUAL AE K CH AX W AX L  
 ACTUALLY AE K CH AX W AX L IY  
 ADD AE DD  
 ADDED AE DX AX DD  
 ADDITIONAL AX D IH SH AX N AX L  
 ADULT AX D AH L TD  
 ADULTS AX D AH L TS  
 ADVANCE AX D V AE N S  
 ADVANTAGE AE D V AE N T IX JH  
 ADVERTISES AE D V AXR T AY Z AX Z  
 ADVISE AE D V AY Z  
 AFTER AE F T AXR  
 AFTERNOON AE F T AXR N UW N  
 AFTERNOONS AE F T AXR N UW N Z  
 AFTERWARDS AE F T AXR W AXR D Z  
 AGAIN AX G EH N  
 AGENCIES EY JH AX N S IY Z  
 AH AA  
 AIRBUS EH R B AX S  
 AIRCRAFT EH R K R AE F TD  
 AIRCRAFTS EH R K R AE F TS  
 AIR EH R  
 AIRFARE EH R F EH R  
 AIRFARES EH R F EH R Z  
 AIRLINE EH R L AY N  
 AIRLINE'S EH R L AY N Z  
 AIRLINES EH R L AY N Z  
 AIRPLANE EH R P L EY N  
 AIRPLANES EH R P L EY N Z  
 AIRPORT EH R P AO R TD  
 AIRPORT'S EH R P AO R TS  
 AIRPORTS EH R P AO R TS  
 AIR'S EH R Z  
 AIRWAYS EH R W EY Z  
 ALASKA AX L AE S K AX  
 ALASKA'S AX L AE S K AX Z  
 ALICE AE L AX S

ALL AO L  
 ALLOW AX L AW  
 ALLOWED AX L AW DD  
 ALONG AX L AO NG  
 ALPHA AE L F AX  
 ALSO AO L S OW  
 AM AE M  
 A M AX EH M  
 AMERICA AX M EH R AX K AX  
 AMERICAN AX M EH R AX K AX N  
 AMERICAN'S AX M EH R AX K AX N Z  
 AMERICA'S AX M EH R AX K AX Z  
 AMOUNT AX M AW N TD  
 AN AE N  
 ANCHORAGE AE NG K AXR AX JH  
 AND AE N DD  
 ANGELES AE N JH AX L AX S  
 ANOTHER AX N AH DH AXR  
 ANSWER AE N S AXR  
 ANTONIO AE N T OW N IY OW  
 ANXIOUS AE NG K SH AX S  
 ANY EH N IY  
 ANYMORE EH N IY M AO R  
 ANYTHING EH N IY TH IH NG  
 ANYTIME EH N IY T AY M  
 ANYWAY EH N IY W EY  
 ANYWAYS EH N IY W EY Z  
 ANYWHERE EH N IY W EH R  
 APART AX P AA R TD  
 APPLE AE P AX L  
 APPLY AX P L AY  
 APPRECIATE AX P R IY SH IY EY TD  
 APPROPRIATE AX P R OW P R IY AX TD  
 APPROXIMATE AX P R AA K S AX M AX TD  
 APPROXIMATELY AX P R AA K S AX M AX TD L IY  
 APRIL EY P R AX L  
 ARE AA R  
 AREA EH R IY AX  
 AREAS EH R IY AX Z  
 AREN'T AA R AX N TD  
 ARIZONA EH R AX Z OW N AX  
 AROUND AXR AW N DD  
 ARRANGE AXR EY N JH  
 ARRANGED AXR EY N JH DD  
 ARRANGEMENT AXR EY N JH M AX N TD  
 ARRANGEMENTS AXR EY N JH M AX N TS  
 ARRANGING AXR EY N JH IX NG  
 ARRIVAL AXR AY V AX L  
 ARRIVALS AXR AY V AX L Z  
 ARRIVE AXR AY V  
 ARRIVES AXR AY V Z  
 ARRIVING AXR AY V IX NG  
 AS AE Z

A'S EY Z  
 ASIDE AX S AY DD  
 ASK AE S KD  
 ASKING AE S K IX NG  
 ASPEN AE S P AX N  
 ASSIGNMENTS AX S AY N M AX N TS  
 ASSOCIATED AX S OW S IY EY DX AX DD  
 ASSUMING AX S UW M IX NG  
 AT AE TD  
 ATLANTA AE TD L AE N T AX  
 ATLANTA'S AE TD L AE N T AX Z  
 A T L AX T IY EH L  
 AUGUST AA G AX S TD  
 AVAILABILITIES AX V EY L AX B IH L AX DX IY Z  
 AVAILABILITY AX V EY L AX B IH L AX DX IY  
 AVAILABLE AX V EY L AX B AX L  
 AVERAGE AE V AXR IX JH  
 AVIS EY V IH S  
 AVOID AX V OY DD  
 AVOIDING AX V OY DX IX NG  
 BACK B AE KD  
 BAD B AE DD  
 BAG B AE GD  
 BAGELS B EY G AX L Z  
 BAGS B AE G Z  
 BALTIMORE B AO L T AX M AO R  
 BANGOR B AE NG G AXR  
 BASED B EY S TD  
 BASIS B EY S IH S  
 BAY B EY  
 B B IY  
 BEACH B IY CH  
 BE B IY  
 BECAUSE B IX K AO Z  
 BEEF B IY F  
 BEEN B IH N  
 BEFORE B AX F AO R  
 BEGIN B IX G IH N  
 BEGINNING B IX G IH N IX NG  
 BEGINS B IX G IH N Z  
 BEING B IY IX NG  
 BELIEVE B AX L IY V  
 BELONG B AX L AO NG  
 BELOW B AX L OW  
 BESIDES B AX S AY D Z  
 BEST B EH S TD  
 BETCHA B EH CH AX  
 BETTER B EH DX AXR  
 BETWEEN B AX T W IY N  
 BIG B IH GD  
 BIGGER B IH G AXR  
 BIGGEST B IH G AX S TD  
 BIT B IH TD

BLOWING B L OW IX NG  
 B\_N\_A B IY EH N AX  
 BOARD B AO R DD  
 BODIES B AA DX IY Z  
 BODY B AA DX IY  
 BOEING B OW IX NG  
 BOOK B UH KD  
 BOOKED B UH KD TD  
 BOOKING B UH K IX NG  
 B\_O\_S B IY OW EH S  
 BOSTON B AA S T AX N  
 BOSTON'S B AA S T AX N Z  
 BOTH B OW TH  
 BOUND B AW N DD  
 BOY B OY  
 BRANIFF B R AE N IH F  
 BRANSON B R AE N S AX N  
 BREAK B R EY KD  
 BREAKFAST B R EH K F AX S TD  
 BREAKING B R EY K IX NG  
 BRING B R IH NG  
 BROAD B R AO DD  
 BUCKS B AH K S  
 BUDGET B AH JH AX TD  
 BUILT B IH L TD  
 BURBANK B ER B AE NG KD  
 BUS B AH S  
 BUSES B AH S AX Z  
 BUSIEST B IH Z IY AX S TD  
 BUSINESS B IH Z N AX S  
 BUSSES B AH S AX Z  
 BUSY B IH Z IY  
 BUT B AH TD  
 BUTTON B AH T AX N  
 BUY B AY  
 B\_W\_I B IY D AH B AX L Y UW AY  
 BY B AY  
 CAB K AE BD  
 CALIFORNIA K AE L AX F AO R N Y AX  
 CALLED K AO L DD  
 CALLING K AO L IX NG  
 CALL K AO L  
 CANADA K AE N AX DX AX  
 CANADIAN K AX N EY DX IY AX N  
 CANCEL K AE N S AX L  
 CAN K AE N  
 CAN'T K AE N TD  
 CAPACITIES K AX P AE S AX DX IY Z  
 CAPACITY K AX P AE S AX DX IY  
 CAPTAIN K AE PD T AX N  
 CARD K AA R DD  
 CARE K EH R  
 CAR K AA R

CAROLINA K EH R AX L AY N AX  
 CARRIED K AE R IY DD  
 CARRIER K AE R IY AXR  
 CARRIERS K AE R IY AXR Z  
 CARRIES K AE R IY Z  
 CARRY K AE R IY  
 CARS K AA R Z  
 CASE K EY S  
 CATCH K AE CH  
 CATEGORIES K AE DX AX G AO R IY Z  
 CATEGORY K AE DX AX G AO R IY  
 CENTER S EH N T AXR  
 CERTAIN S ER T AX N  
 CHANCE CH AE N S  
 CHANGE CH EY N JH  
 CHANGED CH EY N JH DD  
 CHARGE CH AA R JH  
 CHARGES CH AA R JH AX Z  
 CHARLOTTE SH AA R L AX TD  
 CHARLOTTESVILLE SH AA R L AX TS V IH L  
 CHARLOTTEVILLE SH AA R L AX T V IH L  
 CHEAP CH IY PD  
 CHEAPER CH IY P AXR  
 CHEAPEST CH IY P AX S TD  
 CHEAPLY CH IY P L IY  
 CHECK CH EH KD  
 CHECKING CH EH K IX NG  
 CHEERS CH IH R Z  
 CHEESE CH IY Z  
 CHICAGO SH AX K AA G OW  
 CHICKEN CH IH K AX N  
 CHILD CH AY L DD  
 CHILDREN'S CH IH L D R AX N Z  
 CHOICE CH OY S  
 CHOICES CH OY S AX Z  
 CHOLESTEROL K AX L EH S T AXR AO L  
 CHOOSE CH UW Z  
 CHOSEN CH OW Z AX N  
 CHRISTMAS K R IH S M AX S  
 CINCINNATI S IH N S AX N AE DX IY  
 CITIES S IH DX IY Z  
 CITY S IH DX IY  
 CLASSED K L AE S TD  
 CLASSES K L AE S AX Z  
 CLASS K L AE S  
 CLEAR K L IH R  
 CLEVELAND K L IY V L AX N DD  
 CLICK K L IH KD  
 CLOSE K L OW S  
 CLOSER K L OW S AXR  
 CLOSEST K L OW S AX S TD  
 COACH K OW CH  
 COAST K OW S TD



CODE K OW DD  
 CODES K OW D Z  
 COFFEE K AA F IY  
 COLORADO K AA L AXR AA DX OW  
 COLUMBIA K AX L AH M B IY AX  
 COLUMBUS K AX L AH M B AX S  
 COLUMN K AA L AX M  
 COMBINATION K AA M B AX N EY SH AX N  
 COMBINED K AX M B AY N DD  
 COME K AH M  
 COMES K AH M Z  
 COMFORTABLE K AH M F AXR DX AX B AX L  
 COMFORT K AH M F AXR TD  
 COMING K AH M IX NG  
 COMMERCIAL K AX M ER SH AX L  
 COMMERCIALS K AX M ER SH AX L Z  
 COMMON K AA M AX N  
 COMMUTER K AX M Y UW DX AXR  
 COMPANIES K AH M P AX N IY Z  
 COMPANION K AX M P AE N Y AX N  
 COMPANY K AH M P AX N IY  
 COMPARE K AX M P EH R  
 COMPETITION K AA M P AX T IH SH AX N  
 COMPLETE K AX M P L IY TD  
 COMPUTER K AX M P Y UW DX AXR  
 CONCERNING K AX N S ER N IX NG  
 CONDITIONS K AX N D IH SH AX N Z  
 CONFIRM K AX N F ER M  
 CONFUSED K AX N F Y UW Z DD  
 CONGESTION K AX N JH EH S CH AX N  
 CONNECTICUT K AX N EH DX AX K AX TD  
 CONNECTING K AX N EH KD T IX NG  
 CONNECTION K AX N EH K SH AX N  
 CONNECTIONS K AX N EH K SH AX N Z  
 CONNECT K AX N EH KD TD  
 CONNECTS K AX N EH KD TS  
 CONSECUTIVE K AX N S EH K Y AX DX IX V  
 CONSIST K AX N S IH S TD  
 CONTEXT K AA N T EH K S TD  
 CONTINENTAL K AA N T AX N EH N T AX L  
 CONTINENTAL'S K AA N T AX N EH N T AX L Z  
 CONTINENT K AA N T AX N AX N TD  
 CONTINUE K AX N T IH N Y UW  
 CONTINUES K AX N T IH N Y UW Z  
 CONTINUING K AX N T IH N Y UW IX NG  
 CONVENIENT K AX N V IY N Y AX N TD  
 COOKED K UH KD TD  
 COOL K UW L  
 COORDINATE K OW AO R D AX N AX TD  
 CORRECTION K AXR EH K SH AX N  
 CORRECT K AXR EH KD TD  
 CORRESPOND K AO R AX S P AA N DD  
 CORRESPONDS K AO R AX S P AA N D Z

COSTING K AO S T IX NG  
 COST K AA S TD  
 COSTS K AA S TS  
 COULD K UH DD  
 COUNT K AW N TD  
 COUNTRY K AH N T R IY  
 COUNTY K AW N T IY  
 COVERED K AH V AXR DD  
 COVER K AH V AXR  
 CRAFT K R AE F TD  
 CREAM K R IY M  
 CREW K R UW  
 CROSS K R AO S  
 C S IY  
 C'S S IY Z  
 CURRENT K ER AX N TD  
 CURRENTLY K ER AX N TD L IY  
 C\_V\_G S IY V IY JH IY  
 DAILIES D EY L IY Z  
 DAILY D EY L IY  
 DALLAS D AE L AX S  
 DANG D AE NG  
 DARN D AA R N  
 DASH D AE SH  
 DATABASE D EY DX AX B EY S  
 DATA D EY DX AX  
 DATE D EY TD  
 DATES D EY TS  
 DAY D EY  
 DAYS D EY Z  
 D\_C D IY S IY  
 D D IY  
 DEAR D IH R  
 DECEMBER D AX S EH M B AXR  
 DECIDE D AX S AY DD  
 DEFINE D AX F AY N  
 DELAYS D AX L EY Z  
 DELETE D AX L IY TD  
 DELTA D EH L T AX  
 DELTA'S D EH L T AX Z  
 DEN D EH N  
 D\_E\_N D IY IY EH N  
 DENMARK D EH N M AA R KD  
 DENVER D EH N V AXR  
 DEPART D AX P AA R TD  
 DEPARTING D AX P AA R DX IX NG  
 DEPARTS D AX P AA R TS  
 DEPARTURE D AX P AA R CH AXR  
 DEPARTURES D AX P AA R CH AXR Z  
 DEPEND D AX P EH N DD  
 DEPENDENT D AX P EH N D AX N TD  
 DESCENDING D AX S EH N D IX NG  
 DESCRIBE D AX S K R AY BD

DESCRIPTION D AX S K R IH P SH AX N  
 DESCRIPTIONS D AX S K R IH P SH AX N Z  
 DESIGNATED D EH Z IX G N EY DX AX DD  
 DESIGNATE D EH Z IX G N EY TD  
 DESIGNATION D EH Z IX G N EY SH AX N  
 DESIGNATIONS D EH Z IX G N EY SH AX N Z  
 DESIRE D AX Z AY AXR  
 DESTINATION D EH S T AX N EY SH AX N  
 DESTINATIONS D EH S T AX N EY SH AX N Z  
 DETAILS D AX T EY L Z  
 DETERMINE D AX T ER M AX N  
 DETROIT D AX T R OY TD  
 D\_F\_W D IY EH F D AH B AX L Y UW  
 DIABETIC D AY AX B EH DX IX KD  
 DID D IH DD  
 DIDN'T D IH D AX N TD  
 DIEGO D IX EY G OW  
 DIEGO'S D IY EY G OW Z  
 DIET D AY AX TD  
 DIETETIC D AY AX T EH DX IX KD  
 DIFFERENCE D IH F AXR AX N S  
 DIFFERENCES D IH F AXR AX N S AX Z  
 DIFFERENT D IH F AXR AX N TD  
 DINNER D IH N AXR  
 DINNERS D IH N AXR Z  
 DINNERTIME D IH N AXR T AY M  
 DIRECT D AXR EH KD TD  
 DIRECTIONS D AXR EH K SH AX N Z  
 DIRECTLY D AXR EH KD TD L IY  
 DISCOUNT D IH S K AW N TD  
 DISCOUNTED D IH S K AW N T AX DD  
 DISCOUNTS D IH S K AW N TS  
 DISNEYLAND D IH Z N IY L AE N DD  
 DISPLAY D IH S P L EY  
 DISPLAYED D IH S P L EY DD  
 DISPLAYING D IH S P L EY IX NG  
 DISTANCE D IH S T AX N S  
 DISTANCES D IH S T AX N S AX Z  
 DISTRICT D IH S T R IH KD TD  
 DO D UW  
 DOES D AH Z  
 DOESN'T D AH Z AX N TD  
 DOING D UW IX NG  
 DOLLAR D AA L AXR  
 DOLLARS D AA L AXR Z  
 DONE D AH N  
 DON'T D OW N TD  
 DOUBLE D AH B AX L  
 DOUGLAS D AH G L AX S  
 DOWN D AW N  
 DOWNTOWN D AW N T AW N  
 DRINKS D R IH NG K S  
 DRIVE D R AY V

D\_T\_W D IY T IY D AH B AX L Y UW  
 DULLES D AH L AX S  
 DURATION D UH R EY SH AX N  
 DURING D UH R IX NG  
 EACH IY CH  
 EARLIER ER L IY AXR  
 EARLIEST ER L IY AX S TD  
 EARLY ER L IY  
 EASILY IY Z AX L IY  
 EASTERN IY S T AXR N  
 EASTERN'S IY S T AXR N Z  
 EAST IY S TD  
 EAT IY TD  
 ECONOMICAL EH K AX N AA M IX K AX L  
 ECONOMIC EH K AX N AA M IX KD  
 ECONOMY IX K AA N AX M IY  
 EIGHTEEN EY T IY N  
 EIGHTEENTH EY T IY N TH  
 EIGHT EY TD  
 EIGHTH EY TD TH  
 EIGHTY EY DX IY  
 EITHER IY DH AXR  
 E IY  
 ELAPSED AX L AE P S TD  
 EL EH L  
 ELEVEN AX L EH V AX N  
 ELEVENS AX L EH V AX N Z  
 ELEVENTH AX L EH V AX N TH  
 ELIGIBLE EH L AX JH AX B AX L  
 ELIMINATE AX L IH M AX N EY TD  
 ELSE EH L S  
 END EH N DD  
 ENDING EH N D IX NG  
 ENDS EH N D Z  
 ENGINE EH N JH AX N  
 ENGLAND IH NG G L AX N DD  
 ENGLISH IH NG G L IH SH  
 ENOUGH AX N AH F  
 ENROUTE EH N R UW TD  
 ENTER EH N T AXR  
 ENTIRE EH N T AY AXR  
 EQUAL IY K W AX L  
 EQUIPMENT IX K W IH P M AX N TD  
 EQUIVALENT IX K W IH V AX L AX N TD  
 ERROR EH R AXR  
 EVENING IY V N IX NG  
 EVENINGS IY V N IX NG Z  
 EVER EH V AXR  
 EVERYDAY EH V R IY DX EY  
 EVERY EH V AXR IY  
 EVERYTHING EH V R IY TH IH NG  
 EVERYWHERE EH V R IY W EH R  
 E\_W\_R IY D AH B AX L Y UW AA R

EXAMPLE IX G Z AE M P AX L  
 EXCEEDING IX K S IY DX IX NG  
 EXCLUDE IX K S K L UW DD  
 EXCURSION IX K S K ER ZH AX N  
 EXCUSE IX K S K Y UW S  
 EXIST IX G Z IH S TD  
 EXPENSIVE IX K S P EH N S IX V  
 EXPLAIN IX K S P L EY N  
 EXPLANATION EH K S P L AX N EY SH AX N  
 EXPRESS IX K S P R EH S  
 EXTRA EH K S T R AX  
 EYE AY  
 FAIR F EH R  
 FALL F AA L  
 FAMILY F AE M AX L IY  
 FARE F EH R  
 FARES F EH R Z  
 FAR F AA R  
 FASTEST F AE S T AX S TD  
 FAVORITE F EY V AXR AX TD  
 FEBRUARY F EH B Y AX W EH R IY  
 FEES F IY Z  
 F EH F  
 FEWER F Y UW AXR  
 FEWEST F Y UW AX S TD  
 FEW F Y UW  
 FIELD F IY L DD  
 FIFTEEN F IH F T IY N  
 FIFTEENTH F IH F T IY N TH  
 FIFTH F IH F TH  
 FIFTY F IH F T IY  
 FIGHT F AY TD  
 FILLED F IH L DD  
 FINAL F AY N AX L  
 FINALLY F AY N AX L IY  
 FIND F AY N DD  
 FINDING F AY N D IX NG  
 FINE F AY N  
 FINISHED F IH N IX SH TD  
 FIRST F ER S TD  
 FIT F IH TD  
 FITTING F IH DX IX NG  
 FIVE F AY V  
 FLARE F L EH R  
 FLARES F L EH R Z  
 FLIERS F L AY AXR Z  
 FLIES F L AY Z  
 FLIGHT F L AY TD  
 FLIGHTS F L AY TS  
 FLORIDA F L AO R AX DX AX  
 FLOWN F L OW N  
 FLYER F L AY AXR  
 FLY F L AY

FLYING F L AY IX NG  
 FOKKER F AA K AXR  
 FOLLOWING F AA L OW IX NG  
 FOOD F UW DD  
 FOR F AO R  
 FORGET F AXR G EH TD  
 FORM F AO R M  
 FORT F AO R TD  
 FORTY F AO R DX IY  
 FOUND F AW N DD  
 FOUR F AO R  
 FOURTEEN F AO R T IY N  
 FOURTEENTH F AO R T IY N TH  
 FOURTH F AO R TH  
 FRAME F R EY M  
 FRANCE F R AE N S  
 FRANCISCO F R AE N S IH S K OW  
 FRAN F R AE N  
 FRANKFURT F R AE NG K F AXR TD  
 FREE F R IY  
 FREQUENT F R IY K W AX N TD  
 FREQUENTLY F R IY K W AX N TD L IY  
 FRIDAY F R AY DX IY  
 FRIDAY'S F R AY DX IY Z  
 FRIDAYS F R AY DX IY Z  
 FRIEND F R EH N DD  
 FRIENDS F R EH N D Z  
 FRIENDSHIP F R EH N D SH IH PD  
 FROM F R AH M  
 FRUIT F R UW TD  
 FRUSTRATING F R AH S T R EY DX IX NG  
 FUCK F AH KD  
 FULL F UH L  
 FUN F AH N  
 FURTHER F ER DH AXR  
 GAVE G EY V  
 GENERAL JH EH N AXR AX L  
 GEORGIA JH AO R JH AX  
 GERMANY JH ER M AX N IY  
 GET G EH TD  
 GETS G EH TS  
 GETTING G EH DX IX NG  
 GIVE G IH V  
 GIVEN G IH V AX N  
 GIVES G IH V Z  
 GOD G AA DD  
 GOES G OW Z  
 GO G OW  
 GOING G OW IX NG  
 GOODBYE G UH DD B AY  
 GOOD G UH DD  
 GOT G AA TD  
 GRAND G R AE N DD

GREATER G R EY DX AXR  
 GREATEST G R EY DX AX S TD  
 GREAT G R EY TD  
 GROUND G R AW N DD  
 GROUNDS G R AW N D Z  
 GUARDIA G W AA R DX IY AX  
 GUESS G EH S  
 GYMNASTIC JH IH M N AE S T IX KD  
 GYMNASTICS JH IH M N AE S T IX K S  
 HAD HH AE DD  
 HALF HH AE F  
 HAMPSHIRE HH AE M P SH AXR  
 HAPPEN HH AE P AX N  
 HARTFIELD HH AA R T F IY L DD  
 HARTFORD HH AA R T F AXR DD  
 HAS HH AE Z  
 HAVE HH AE V  
 HAVING HH AE V IX NG  
 HAWAII HH AX W AY IY  
 HEAD HH EH DD  
 HEADING HH EH DX IX NG  
 HEADQUARTERS HH EH DD K W AO R DX AXR Z  
 HEAR HH IH R  
 HEATHROW HH IY TH R OW  
 HEAVIEST HH EH V IY AX S TD  
 HELL HH EH L  
 HELLO HH AX L OW  
 HELP HH EH L PD  
 HERE HH IH R  
 HERTZ HH EH R TS  
 H EY CH  
 HEY HH EY  
 HIGHER HH AY AXR  
 HIGHEST HH AY AX S TD  
 HI HH AY  
 HILTON HH IH L T AX N  
 HISTORY HH IH S T AXR IY  
 HIT HH IH TD  
 HOLD HH OW L DD  
 HOLDS HH OW L D Z  
 HOME HH OW M  
 HONOLULU HH AA N AX L UW L UW  
 HOPEFULLY HH OW P F AX L IY  
 HORSE HH AO R S  
 HOTEL HH OW T EH L  
 HOTELS HH OW T EH L Z  
 HOT HH AA TD  
 H\_O\_U EY CH OW Y UW  
 HOUR AW AXR  
 HOURS AW AXR Z  
 HOUSTON HH Y UW S T AX N  
 HOW HH AW  
 H\_P\_N EY CH P IY EH N

HUB HH AH BD  
 HUBS HH AH B Z  
 HUNDRED HH AH N D R AX DD  
 HUNG HH AH NG  
 I\_A\_D AY AX D IY  
 I\_A\_H AY AX EY CH  
 I\_AY  
 I'D AY DD  
 IDENTIFY AY D EH N T AX F AY  
 IF IH F  
 IGNORE IX G N AO R  
 I'LL AY L  
 ILLINOIS IH L AX N OY  
 ILLNESS IH L N AX S  
 I'M AY M  
 IN AX N  
 INCLUDED IH N K L UW DX AX DD  
 INCLUDE IH N K L UW DD  
 INCLUDES IH N K L UW D Z  
 INCLUDING IH N K L UW DX IX NG  
 INCORPORATED IH N K AO R P AXR EY DX AX DD  
 INCORRECT IH N K AXR EH KD TD  
 INCREASING IH N K R IY S IX NG  
 I\_N\_D AY EH N D IY  
 INDEPENDENCE IH N D AX P EH N D AX N S  
 INDIANA IH N D IY AE N AX  
 INDIANAPOLIS IH N D IY AX N AE P AX L IH S  
 INDICATE IH N D AX K EY TD  
 INDIVIDUAL IH N D AX V IH JH AX W AX L  
 INEXPENSIVE IH N IX K S P EH N S IX V  
 INFLIGHT IH N F L AY TD  
 INFO IH N F OW  
 INFORMATION IH N F AXR M EY SH AX N  
 INFORMATIONS IH N F AXR M EY SH AX N Z  
 INFORM IH N F AO R M  
 INITIALS IH N IH SH AX L Z  
 INNER IH N AXR  
 INQUIRE IH N K W AY R  
 INQUIRING IH N K W AY AXR IX NG  
 INSTEAD IH N S T EH DD  
 INTERCONTINENTAL IH N T AXR K AA N T AX N EH N T AX L  
 INTERESTED IH N T R AX S T AX DD  
 INTERMEDIATE IH N T AXR M IY DX IY AX TD  
 INTERNATIONAL IH N T AXR N AE SH AX N AX L  
 INTERVIEW IH N T AXR V Y UW  
 INTO IH N T UW  
 INVOLVES IH N V AA L V Z  
 IRRELEVANT IX R EH L AX V AX N TD  
 IS IH Z  
 ISN'T IH Z AX N TD  
 ISSUE IH SH UW  
 IT IH TD  
 ITINERARIES AY T IH N AXR EH R IY Z



ITINERARY AY T IH N AXR EH R IY  
 IT'S IH TS  
 ITS IH TS  
 I'VE AY V  
 JANUARY JH AE N Y UW EH R IY  
 JAW JH AO  
 JERSEY JH ER Z IY  
 JET JH EH TD  
 JETS JH EH TS  
 J\_F\_K JH EY EH F K EY  
 J JH EY  
 JOB JH AA BD  
 JOE JH OW  
 JOHN JH AA N  
 JOINING JH OY N IX NG  
 JOSE HH OW Z EY  
 JULY JH UW L AY  
 JUNE JH UW N  
 JUST JH AH S TD  
 KANSAS K AE N Z AX S  
 KATE K EY TD  
 KEEPING K IY P IX NG  
 KEEP K IY PD  
 KENNEDY K EH N AX DX IY  
 KICK K IH KD  
 KIND K AY N DD  
 KINDLY K AY N D L IY  
 KINDS K AY N D Z  
 K K EY  
 KNOWN N OW N  
 KNOW N OW  
 KOSHER K OW SH AXR  
 L\_A EH L AX  
 LAGUARDIA L AX G W AA R DX IY AX  
 LAKE L EY KD  
 LA L AA  
 LANDING L AE N D IX NG  
 LANDINGS L AE N D IX NG Z  
 LAND L AE N DD  
 LANDS L AE N D Z  
 LARGE L AA R JH  
 LARGER L AA R JH AXR  
 LARGEST L AA R JH AX S TD  
 L\_A\_S EH L AX EH S  
 LAS L AA S  
 LAST L AE S TD  
 LATE L EY TD  
 LATER L EY DX AXR  
 LATEST L EY DX AX S TD  
 LAUNCH L AO N CH  
 LAW L AA  
 L\_A\_X EH L AX EH K S  
 LAYING L EY IX NG

LAYOVER L EY OW V AXR  
 LAYOVERS L EY OW V AXR Z  
 LEAST L IY S TD  
 LEAVE L IY V  
 LEAVES L IY V Z  
 LEAVING L IY V IX NG  
 LEFT L EH F TD  
 LEG L EH GD  
 L EH L  
 LENGTH L EH NG KD TH  
 LESS L EH S  
 LESTER L EH S T AXR  
 LET L EH TD  
 LET'S L EH TS  
 LETTER L EH DX AXR  
 LEVEL L EH V AX L  
 LEVELS L EH V AX L Z  
 L G A EH L JH IY AX  
 LIGHT L AY TD  
 LIKE L AY KD  
 LIMIT L IH M AX TD  
 LIMO L IH M OW  
 LIMOUSINE L IH M AX Z IY N  
 LIMOUSINES L IH M AX Z IY N Z  
 LINE L AY N  
 LINKING L IH NG K IX NG  
 LISTED L IH S T AX DD  
 LISTENING L IH S AX N IX NG  
 LISTEN L IH S AX N  
 LISTING L IH S T IX NG  
 LISTINGS L IH S T IX NG Z  
 LIST L IH S TD  
 LITTLE L IH DX AX L  
 LIVE L AY V  
 LIVES L IH V Z  
 LIVING L IH V IX NG  
 LIZ L IH Z  
 LOCAL L OW K AX L  
 LOCATED L OW K EY DX AX DD  
 LOCATE L OW K EY TD  
 LOCATION L OW K EY SH AX N  
 LOCATIONS L OW K EY SH AX N Z  
 LOGAN L OW G AX N  
 LOGO L OW G OW  
 LONDON L AH N D AX N  
 LONGER L AO NG G AXR  
 LONGEST L AO NG G AX S TD  
 LONG L AO NG  
 LOOKING L UH K IX NG  
 LOOK L UH KD  
 LOOKS L UH K S  
 LOOP L UW PD  
 LOS L OW S

LOT L AA TD  
 LOTS L AA TS  
 LOUIS L UW AX S  
 LOVE L AH V  
 LOWER L OW AXR  
 LOWEST L OW AX S TD  
 LOW L OW  
 LUFTHANSA L AX F T AE N Z AX  
 LUNCH L AH N CH  
 LUNCHTIME L AH N CH T AY M  
 MACHINE M IX SH IY N  
 MADE M EY DD  
 MAKE M EY KD  
 MAKES M EY K S  
 MAKING M EY K IX NG  
 MALUSO M AX L UW S OW  
 MANUFACTURER M AE N Y AX F AE K CH AXR AXR  
 MANY M EH N IY  
 MARCH M AA R CH  
 MARYLAND M EH R AX L AX N DD  
 MASSACHUSETTS M AE S AX CH UW S AX TS  
 MASTER M AE S T AXR  
 MATCH M AE CH  
 MAXIMUM M AE K S AX M AX M  
 MAYBE M EY B IY  
 MAY M EY  
 MCDONNELL M AX KD D AA N AX L  
 M\_C\_I EH M S IY AY  
 M\_C\_O EH M S IY OW  
 M\_D\_W EH M D IY D AH B AX L Y UW  
 MEAL M IY L  
 MEAL'S M IY L Z  
 MEALS M IY L Z  
 MEALTIME M IY L T AY M  
 MEANING M IY N IX NG  
 MEANINGS M IY N IX NG Z  
 MEAN M IY N  
 MEANS M IY N Z  
 MEANT M EH N TD  
 MEET M IY TD  
 M EH M  
 ME M IY  
 MEMPHIS M EH M F AX S  
 MENTIONED M EH N SH AX N DD  
 MENU M EH N Y UW  
 METRO M EH T R OW  
 M\_I\_A EH M AY AX  
 MIAMI M AY AE M IY  
 MICHIGAN M IH SH IX G AX N  
 MIDDLE M IH DX AX L  
 MID M IH DD  
 MIDNIGHT M IH D N AY TD  
 MIDPOINT M IH DD P OY N TD

MIDWAY M IH D W EY  
 MIDWEEK M IH D W IY KD  
 MIDWEST M IH D W EH S TD  
 MIGHT M AY TD  
 MILEAGE M AY L AX JH  
 MILES M AY L Z  
 MILWAUKEE M IH L W AO K IY  
 MIND M AY N DD  
 MINIMAL M IH N AX M AX L  
 MINIMUM M IH N AX M AX M  
 MINNEAPOLIS M IH N IY AE P AX L AX S  
 MINNESOTA M IH N AX S OW DX AX  
 MINUS M AY N AX S  
 MINUTES M IH N AX TS  
 MISSOURI M AX Z UH R IY  
 MISTAKE M IH S T EY KD  
 MISUNDERSTAND M IH S AX N D AXR S T AE N DD  
 MISUNDERSTOOD M IH S AX N D AXR S T UH DD  
 MITCHELL M IH CH AX L  
 MODEL M AA DX AX L  
 MONDAY M AH N D IY  
 MONDAY'S M AH N D IY Z  
 MONDAYS M AH N D IY Z  
 MONEY M AH N IY  
 MONTH M AH N TH  
 MONTREAL M AH N T R IY AO L  
 MORE M AO R  
 MORNING M AO R N IX NG  
 MORNINGS M AO R N IX NG Z  
 MOST M OW S TD  
 MOTHER M AH DH AXR  
 MOVIE M UW V IY  
 MOVIES M UW V IY Z  
 M\_S\_P EH M EH S P IY  
 MUCH M AH CH  
 MUFFINS M AH F AX N Z  
 MUST M AH S TD  
 MY M AY  
 MYSELF M AY S EH L F  
 NAME N EY M  
 NAMES N EY M Z  
 NASHVILLE N AE SH V IH L  
 NATIONAIR N EY SH AX N EH R  
 NATIONAL N AE SH AX N AX L  
 NEAREST N IH R AX S TD  
 NEAR N IH R  
 NECESSARILY N EH S AX S EH R AX L IY  
 NEEDED N IY DX AX DD  
 NEEDING N IY DX IX NG  
 NEED N IY DD  
 NEEDS N IY D Z  
 N EH N  
 NEVADA N AX V AA DX AX

NEVER N EH V AXR  
 NEWARK N UW AXR KD  
 NEW N UW  
 NEXT N EH K S TD  
 NICE N AY S  
 NIGHT N AY TD  
 NIGHTS N AY TS  
 NIGHTTIME N AY TD T AY M  
 NINE N AY N  
 NINER N AY N AXR  
 NINETEEN N AY N T IY N  
 NINETEENTH N AY N T IY N TH  
 NINETY N AY N T IY  
 NINTH N AY N TH  
 NONDIRECT N AA N D AXR EH KD TD  
 NONJETS N AA N JH EH TS  
 NON N AA N  
 NO N OW  
 NONPOSTSTOP N AA N P OW S TS T AA PD  
 NONSTOP N AA N S T AA PD  
 NONSTOPS N AA N S T AA P S  
 NOON N UW N  
 NOONTIME N UW N T AY M  
 NOPE N OW PD  
 NORTH N AO R TH  
 NORTHWESTERN N AO R TH W EH S T AXR N  
 NORTHWEST N AO R TH W EH S TD  
 NOTHING N AH TH IX NG  
 NOTICED N OW DX AX S TD  
 NOT N AA TD  
 NOVEMBER N OW V EH M B AXR  
 NOW N AW  
 NULL N AH L  
 NUMBER N AH M B AXR  
 NUMBERS N AH M B AXR Z  
 OAKLAND OW K L AX N DD  
 OBTAIN AX BD T EY N  
 OCCUR AX K ER  
 O'CLOCK AX K L AA KD  
 OCTOBER AA KD T OW B AXR  
 OF AH V  
 OFF AO F  
 OFFER AO F AXR  
 OFFERED AO F AXR DD  
 OFFERS AO F AXR Z  
 OFFICE AO F AX S  
 OFFICES AO F AX S AX Z  
 OFTEN AO F AX N  
 O'HARE OW HH EH R  
 OHIO OW HH AY OW  
 OH OW  
 OKAY OW K EY  
 OKLAHOMA OW K L AX HH OW M AX

OK OW K EY  
 OMIT OW M IH TD  
 ON AA N  
 ONCE W AH N S  
 ONE'S W AH N Z  
 ONES W AH N Z  
 ONE W AH N  
 ONLY OW N L IY  
 ONTARIO AA N T EH R IY OW  
 OOPS UW P S  
 OOP UW PD  
 O OW  
 OPEN OW P AX N  
 OPERATE AA P AXR EY TD  
 OPERATES AA P AXR EY TS  
 OPERATING AA P AXR EY DX IX NG  
 OPERATION AA P AXR EY SH AX N  
 OPTION AA P SH AX N  
 OPTIONAL AA P SH AX N AX L  
 OPTIONS AA P SH AX N Z  
 ORANGE AO R AX N JH  
 OR AO R  
 ORD AO R DD  
 ORDER AO R DX AXR  
 ORDERED AO R DX AXR DD  
 O\_R\_D OW AA R D IY  
 ORGANIZE AO R G AX N AY Z  
 ORIENT AO R IY EH N TD  
 ORIGINAL AXR IH JH AX N AX L  
 ORIGIN AO R AX JH AX N  
 ORIGINATE AXR IH JH AX N EY TD  
 ORIGINATING AXR IH JH AX N EY DX IX NG  
 ORIGINATION AXR IH JH AX N EY SH AX N  
 ORIGINS AO R AX JH IH N Z  
 ORLANDO AO R L AE N D OW  
 ORLEANS AO R L IY AX N Z  
 OTHER AH DH AXR  
 OTHERS AH DH AXR Z  
 OUR AW AXR  
 OUT AW TD  
 OVERNIGHT OW V AXR N AY TD  
 OVER OW V AXR  
 OWN OW N  
 PACIFIC P AX S IH F IX KD  
 PAN\_AM P AE N AE M  
 PARDON P AA R D AX N  
 PARIS P AE R IH S  
 PARKING P AA R K IX NG  
 PARTICULAR P AXR T IH K Y AX L AXR  
 PARTY P AA R DX IY  
 PASO P AE S OW  
 PASSAGE P AE S AX JH  
 PASSAGES P AE S AX JH AX Z

PASSENGER P AE S AX N JH AXR  
 PASSENGERS P AE S AX N JH AXR Z  
 PAUL P AO L  
 PAY P EY  
 PEAK P IY KD  
 PEARSON P IH R S AX N  
 PENNSYLVANIA P EH N S AX L V EY N Y AX  
 PEOPLE P IY P AX L  
 PERIOD P IH R IY AX DD  
 PERKS P ER K S  
 PER P ER  
 PERSON P ER S AX N  
 PETERSBURG P IY DX AXR Z B AXR GD  
 PHILADELPHIA F IH L AX D EH L F IY AX  
 PHILLY F IH L IY  
 P\_H\_L P IY EY CH EH L  
 PHOENIX F IY N IX K S  
 PICKED P IH KD TD  
 PICK P IH KD  
 PITT P IH TD  
 PITTSBURGH P IH TS B AXR GD  
 PLACE P L EY S  
 PLACES P L EY S AX Z  
 PLANE P L EY N  
 PLANES P L EY N Z  
 PLANNING P L AE N IX NG  
 PLAN P L AE N  
 PLANS P L AE N Z  
 PLATES P L EY TS  
 PLEASE P L IY Z  
 PLEASURE P L EH ZH AXR  
 PLUS P L AH S  
 P\_M P IY EH M  
 POINT P OY N TD  
 POINTS P OY N TS  
 PORTION P AO R SH AX N  
 PORTLAND P AO R TD L AX N DD  
 POSSIBILITIES P AA S AX B IH L AX DX IY Z  
 POSSIBLE P AA S AX B AX L  
 P P IY  
 PREFERABLY P R EH F AXR AX B L IY  
 PREFERENCE P R EH F AXR AX N S  
 PREFER P R AX F ER  
 PREFIX P R IY F IX K S  
 PRESS P R EH S  
 PREVIOUS P R IY V IY AX S  
 PRICED P R AY S TD  
 PRICE P R AY S  
 PRICES P R AY S AX Z  
 PRICING P R AY S IX NG  
 PRIOR P R AY AXR  
 PROBABLY P R AA B AX B L IY  
 PROBLEM P R AA B L AX M

PROBLEMS P R AA B L AX M Z  
 PROCESSING P R AA S EH S IX NG  
 PROGRAMMED P R OW G R AE M DD  
 PROGRAM P R OW G R AE M  
 PROMOTIONS P R AX M OW SH AX N Z  
 PROPELLED P R AX P EH L DD  
 PROPER P R AA P AXR  
 PROP P R AA PD  
 PROPULSION P R AX P AH L SH AX N  
 PROVIDED P R AX V AY DX AX DD  
 PROVIDE P R AX V AY DD  
 PROVIDES P R AX V AY D Z  
 PROVIDING P R AX V AY DX IX NG  
 PUBLIC P AH B L IX KD  
 PURCHASE P ER CH AX S  
 PUSHED P UH SH TD  
 PUT P UH TD  
 Q K Y UW  
 QUALIFY K W AA L AX F AY  
 QUEBEC K W AX B EH KD  
 QUERIES K W IH R IY Z  
 QUERY K W IY R IY  
 QUESTION K W EH S CH AX N  
 QUICK K W IH KD  
 QUICKLY K W IH K L IY  
 QUIET K W AY AX TD  
 QUIT K W IH TD  
 QUOTED K W OW DX AX DD  
 R AA R  
 RATE R EY TD  
 RATES R EY TS  
 RATHER R AE DH AXR  
 RATS R AE TS  
 REACHES R IY CH AX Z  
 REACHING R IY CH IX NG  
 REACH R IY CH  
 READ R EH DD  
 READY R EH DX IY  
 REALLY R IH L IY  
 RECEIVE R AX S IY V  
 RECOGNIZE R EH K AX G N AY Z  
 RECOMMEND R EH K AX M EH N DD  
 RECONNECT R IY K AX N EH KD TD  
 RECORDED R AX K AO R DX AX DD  
 RECORDING R AX K AO R DX IX NG  
 RECORD R AX K AO R DD  
 RECORDS R AX K AO R D Z  
 REDEYE R IY DX AY  
 RED R EH DD  
 REDUCED R AX D UW S TD  
 REFUNDABLE R AX F AH N D AX B AX L  
 REFUND R AX F AH N DD  
 REGARDING R AX G AA R DX IX NG



REGARDLESS R AX G AA R D L AX S  
 REGULAR R EH G Y AX L AXR  
 RELATES R AX L EY TS  
 RELATIVES R EH L AX DX IX V Z  
 RELAX R AX L AE K S  
 REMAIN R AX M EY N  
 REMEMBER R AX M EH M B AXR  
 REMOVE R IY M UW V  
 RENO R IY N OW  
 RENTAL R EH N T AX L  
 RENTALS R EH N T AX L Z  
 RENTED R EH N T AX DD  
 RENTING R EH N T IX NG  
 RENT R EH N TD  
 REORDER R IY AO R DX AXR  
 REPEATING R AX P IY DX IX NG  
 REPEAT R AX P IY TD  
 REPHRASE R IY F R EY Z  
 REPRESENTED R EH P R AX Z EH N T AX DD  
 REPRESENTING R EH P R AX Z EH N T IX NG  
 REQUESTED R IX K W EH S T AX DD  
 REQUESTING R IX K W EH S T IX NG  
 REQUEST R IX K W EH S TD  
 REQUIRED R IY K W AY AXR DD  
 REQUIREMENTS R IX K W AY R M AX N TS  
 REQUIRE R IY K W AY AXR  
 REQUIRES R IY K W AY AXR Z  
 REQUIRING R IY K W AY AXR IX NG  
 RESERVATION R EH Z AXR V EY SH AX N  
 RESERVATIONS R EH Z AXR V EY SH AX N Z  
 RESERVED R AX Z ER V DD  
 RESERVE R AX Z ER V  
 RESET R IY S EH TD  
 RESIDE R AX Z AY DD  
 RESORT R AX Z AO R TD  
 RESORTS R AX Z AO R TS  
 RESPEAK R IY S P IY KD  
 RESPECTIVELY R AX S P EH KD T IX V L IY  
 RESPONSE R AX S P AA N S  
 REST R EH S TD  
 RESTRICTED R IY S T R IH KD T AX DD  
 RESTRICTION R IY S T R IH K SH AX N  
 RESTRICTIONS R IY S T R IH K SH AX N Z  
 RESTRICT R IY S T R IH KD TD  
 RESULTS R AX Z AH L TS  
 RETURNING R AX T ER N IX NG  
 RETURN R AX T ER N  
 RETURNS R AX T ER N Z  
 REVERSE R IX V ER S  
 REVIEW R IY V Y UW  
 RICHARDSON R IH CH AXR D S AX N  
 RICHMOND R IH CH M AX N DD  
 RIDE R AY DD

RIGHT R AY TD  
 RIVERSIDE R IH V AXR S AY DD  
 ROOM R UW M  
 ROUGHLY R AH F L IY  
 ROUND R AW N DD  
 ROUTE R UW TD  
 RUN R AH N  
 RUNS R AH N Z  
 RUSH R AH SH  
 SACRIFICE S AE K R AX F AY S  
 SAFE S EY F  
 SAFEST S EY F AX S TD  
 SAFETY S EY F T IY  
 SAID S EH DD  
 SAINT S EY N TD  
 SALAD S AE L AX DD  
 SALT S AO L TD  
 SAME S EY M  
 SAM S AE M  
 SAN S AE N  
 SATURDAY S AE DX AXR DX IY  
 SATURDAY'S S AE DX AXR DX IY Z  
 SATURDAYS S AE DX AXR DX IY Z  
 SAVER S EY V AXR  
 SAVE S EY V  
 SAVING S EY V IX NG  
 SAYING S EY IX NG  
 SAY S EY  
 SAYS S EH Z  
 SCENARIO S AX N EH R IY OW  
 SCHEDULED S K EH JH UH L DD  
 SCHEDULE S K EH JH UH L  
 SCHEDULES S K EH JH UH L Z  
 SCHEDULING S K EH JH UH L IX NG  
 SCONE S K OW N  
 SCRATCH S K R AE CH  
 SCREEN S K R IY N  
 SCREWED S K R UW DD  
 SCROLL S K R OW L  
 S\_E\_A EH S IY AX  
 SEAFOOD S IY F UW DD  
 SEARCH S ER CH  
 SEATING S IY DX IX NG  
 SEAT S IY TD  
 SEATS S IY TS  
 SEATTLE S IY AE DX AX L  
 SECOND S EH K AX N DD  
 SECTION S EH K SH AX N  
 SECTIONS S EH K SH AX N Z  
 SEEM S IY M  
 SEEMS S IY M Z  
 SEEN S IY N  
 SEE S IY

S EH S  
 SELECTED S AX L EH KD T AX DD  
 SELECTIONS S AX L EH K SH AX N Z  
 SELECT S AX L EH KD TD  
 SEND S EH N DD  
 SENTENCE S EH N T AX N S  
 SEPARATELY S EH P AXR AX TD L IY  
 SEPTEMBER S EH PD T EH M B AXR  
 SERIES S IH R IY Z  
 SERVED S ER V DD  
 SERVE S ER V  
 SERVES S ER V Z  
 SERVICED S ER V AX S TD  
 SERVICE S ER V AX S  
 SERVICES S ER V AX S AX Z  
 SERVING S ER V IX NG  
 SESSION S EH SH AX N  
 SET S EH TD  
 SEVEN S EH V AX N  
 SEVENS S EH V AX N Z  
 SEVENTEEN S EH V AX N T IY N  
 SEVENTEENTH S EH V AX N T IY N TH  
 SEVENTH S EH V AX N TH  
 SEVENTY S EH V AX N T IY  
 SEVERAL S EH V R AX L  
 S\_F\_O EH S EH F OW  
 SHALL SH AE L  
 SHE SH IY  
 SHOOT SH UW TD  
 SHORTER SH AO R DX AXR  
 SHORTEST SH AO R DX AX S TD  
 SHORTLY SH AO R TD L IY  
 SHOULD SH UH DD  
 SHOWED SH OW DD  
 SHOWING SH OW IX NG  
 SHOWN SH OW N  
 SHOW SH OW  
 SHOWS SH OW Z  
 SHUCKS SH AH K S  
 SHUTTLE SH AH DX AX L  
 SIERRA S IY EH R AX  
 SIGNIFY S IH G N AX F AY  
 SIMILAR S IH M AX L AXR  
 SINGLE S IH NG G AX L  
 SIT S IH TD  
 SIX S IH K S  
 SIX'S S IH K S AX Z  
 SIXTEEN S IX K S T IY N  
 SIXTEENTH S IX K S T IY N TH  
 SIXTH S IH K S TH  
 SIXTY S IH K S T IY  
 SIZED S AY Z DD  
 SIZE S AY Z

SIZES S AY Z AX Z  
 SKIP S K IH PD  
 SLASH S L AE SH  
 SLOWEST S L OW AX S TD  
 SMALLER S M AO L AXR  
 SMALLEST S M AO L AX S TD  
 SMALL S M AO L  
 SMOKING S M OW K IX NG  
 SNACK S N AE KD  
 SNACKS S N AE K S  
 SOMEBODY S AH M B AA DX IY  
 SOME S AH M  
 SOMETHING S AH M TH IX NG  
 SOMETIME S AH M T AY M  
 SOMEWHERE S AH M W EH R  
 SOONER S UW N AXR  
 SOONEST S UW N AX S TD  
 SOON S UW N  
 SORRY S AA R IY  
 SORTED S AO R DX AX DD  
 SORT S AO R TD  
 SO S OW  
 SOUNDS S AW N D Z  
 SOUTHERN S AH DH AXR N  
 SOUTHWESTERN S AW TH W EH S T AXR N  
 SOUTHWEST S AW TH W EH S TD  
 SPACE S P EY S  
 SPECIAL S P EH SH AX L  
 SPECIALS S P EH SH AX L Z  
 SPECIFICATIONS S P EH S AX F IX K EY SH AX N Z  
 SPECIFIC S P AX S IH F IX KD  
 SPECIFICS S P AX S IH F IX K S  
 SPECIFIED S P EH S AX F AY DD  
 SPECIFY S P EH S AX F AY  
 SPEECH S P IY CH  
 SPEED S P IY DD  
 SPEND S P EH N DD  
 SPLIT S P L IH TD  
 SPOUSE S P AW S  
 STANDARD S T AE N D AXR DD  
 STANDBY S T AE N DD B AY  
 STANDS S T AE N D Z  
 STAND S T AE N DD  
 STAPLETON S T EY P AX L T AX N  
 STARTING S T AA R DX IX NG  
 STARTS S T AA R TS  
 START S T AA R TD  
 STATES S T EY TS  
 STATE S T EY TD  
 STATUS S T AE DX AX S  
 STAYING S T EY IX NG  
 STAYOVER S T EY OW V AXR  
 STAYS S T EY Z

STAY S T EY  
 STEAK S T EY KD  
 STILL S T IH L  
 STOCKHOLM S T AA K HH OW L M  
 STOPOVERS S T AA P OW V AXR Z  
 STOPOVER S T AA P OW V AXR  
 STOPPING S T AA P IX NG  
 STOPS S T AA P S  
 STOP S T AA PD  
 STRAIGHT S T R EY TD  
 STREET S T R IY TD  
 STRICT S T R IH KD TD  
 STUCK S T AH KD  
 STUDENT S T UW D AX N TD  
 STUPID S T UW P AX DD  
 SUBWAY S AH B W EY  
 SUBWAYS S AH B W EY Z  
 SUCH S AH CH  
 SUGAR SH UH G AXR  
 SUGGEST S AX G JH EH S TD  
 SUMMER S AH M AXR  
 SUNDAY S AH N D EY  
 SUNDAY'S S AH N D EY Z  
 SUNDAYS S AH N D EY Z  
 SUPERSAVER S UW P AXR S EY V AXR  
 SUPER S UW P AXR  
 SUPPER S AH P AXR  
 SUPPLY S AX P L AY  
 SURE SH UH R  
 SYMBOL S IH M B AX L  
 SYMBOLS S IH M B AX L Z  
 SYSTEM S IH S T AX M  
 TABLE T EY B AX L  
 TACOMA T AX K OW M AX  
 TAKEOFFS T EY K AO F S  
 TAKEOFF T EY K AO F  
 TAKES T EY K S  
 TAKE T EY KD  
 TAKING T EY K IX NG  
 TALKING T AO K IX NG  
 TALK T AO KD  
 TALLAHASSEE T AE L AX HH AE S IY  
 TAMPA T AE M P AX  
 TAXICABS T AE K S IY K AE B Z  
 TAXIS T AE K S IY Z  
 TAXI T AE K S IY  
 TEAM T IY M  
 TELEPHONE T EH L AX F OW N  
 TELL T EH L  
 TENNESSEE T EH N AX S IY  
 TENS T EH N Z  
 TEN T EH N  
 TENTH T EH N TH

TERMINAL T ER M AX N AX L  
 TERMINATES T ER M AX N EY TS  
 TERMINATING T ER M AX N EY DX IX NG  
 TERMS T ER M Z  
 TEXAS T EH K S AX S  
 TEXTBOOK T EH K S TD B UH KD  
 THAN DH AE N  
 THANKS TH AE NG K S  
 THANK TH AE NG KD  
 THAT DH AE TD  
 THAT'LL DH AE DX AX L  
 THAT'S DH AE TS  
 THE DH AX  
 THEIR DH EH R  
 THEM DH EH M  
 THEN DH EH N  
 THEREAFTER DH EH R AE F T AXR  
 THERE DH EH R  
 THERE'S DH EH R Z  
 THESE DH IY Z  
 THEY DH EY  
 THEY'RE DH EH R  
 THING TH IH NG  
 THINK TH IH NG KD  
 THIRD TH ER DD  
 THIRTEEN TH ER T IY N  
 THIRTEENTH TH ER T IY N TH  
 THIRTIETH TH ER DX IY AX TH  
 THIRTY TH ER DX IY  
 THIS DH IH S  
 THOSE DH OW Z  
 THOUGHT TH AO TD  
 THOUSAND TH AW Z AX N DD  
 THREES TH R IY Z  
 THREE TH R IY  
 THRIFT TH R IH F TD  
 THRIFTY TH R IH F T IY  
 THROUGH TH R UW  
 THURSDAYS TH ER Z D EY Z  
 THURSDAY TH ER Z D EY  
 TICKETS T IH K AX TS  
 TICKET T IH K AX TD  
 TILL T IH L  
 TIME'S T AY M Z  
 TIMES T AY M Z  
 TIME T AY M  
 TIP T IH PD  
 TOAST T OW S TD  
 TODAY'S T AX D EY Z  
 TODAY T AX D EY  
 TOLD T OW L DD  
 TOMORROW'S T AX M AA R OW Z  
 TOMORROW T AX M AA R OW

TONIGHT T AX N AY TD  
 TOOK T UH KD  
 TOO T UW  
 TORONTO T AXR AA N T OW  
 TOTAL T OW DX AX L  
 TO T UW  
 TOUGH T AH F  
 TOURING T UH R IX NG  
 TOURIST T UH R AX S TD  
 TOUR T UH R  
 TOWARDS T AX W AO R D Z  
 TOWARD T AX W AO R DD  
 TOWER T AW AXR  
 TOWN T AW N  
 T\_P\_A T IY P IY AX  
 TRAFFIC T R AE F IX KD  
 TRAINS T R EY N Z  
 TRAIN T R EY N  
 TRANSCONTINENTAL T R AE N Z K AA N T AX N EH N T AX L  
 TRANSFERRING T R AE N S F ER IX NG  
 TRANSFER T R AE N S F ER  
 TRANSPORTATIONS T R AE N S P R T EY SH AX N Z  
 TRANSPORTATION T R AE N S P AXR T EY SH AX N  
 TRANSPORT T R AE N S P AO R TD  
 TRANS T R AE N Z  
 TRAP T R AE PD  
 TRAVELED T R AE V AX L DD  
 TRAVELING T R AE V AX L IX NG  
 TRAVELS T R AE V AX L Z  
 TRAVEL T R AE V AX L  
 TRIANGLE T R AY AE NG G AX L  
 TRIPS T R IH P S  
 TRIP T R IH PD  
 TRYING T R AY IX NG  
 TRY T R AY  
 T T IY  
 TUCSON T UW S AA N  
 TUESDAYS T UW Z D EY Z  
 TUESDAY T UW Z D IY  
 TURBOPROP T ER B OW P R AA PD  
 TURNS T ER N Z  
 T\_W\_A T IY D AH B AX L Y UW AX  
 TWELFTH T W EH L F TH  
 TWELVE T W EH L V  
 TWENTIETH T W EH N T IY AX TH  
 TWENTY T W EH N T IY  
 TWOS T UW Z  
 TWO T UW  
 TYPES T AY P S  
 TYPE T AY PD  
 UNABLE AX N EY B AX L  
 UNBOOK AX N B UH KD  
 UNDER AH N D AXR

UNDERGROUND AH N D AXR G R AW N DD  
 UNDERSTAND AH N D AXR S T AE N DD  
 UNITED'S Y UW N AY DX AX D Z  
 UNITED Y UW N AY DX AX DD  
 UNPRESSURIZED AX N P R EH SH AXR AY Z DD  
 UNTIL AX N T IH L  
 UP AH PD  
 UPON AX P AA N  
 US AH S  
 U\_S\_AIR Y UW EH S EH R  
 USED Y UW Z DD  
 USES Y UW S AX Z  
 USE Y UW S  
 USING Y UW Z IX NG  
 UTAH Y UW T AO  
 U Y UW  
 VACATION V EY K EY SH AX N  
 VALID V AE L AX DD  
 VANS V AE N Z  
 VARIOUS V EH R IY AX S  
 VEGAS V EY G AX S  
 VEGETARIAN V EH JH AX T EH R IY AX N  
 VERY V EH R IY  
 VIA V AY AX  
 VICINITY V AX S IH N AX DX IY  
 VIRGINIA V AXR JH IH N Y AX  
 VISA V IY Z AX  
 VISIT V IH Z AX TD  
 VOCABULARY V OW K AE B Y AX L EH R IY  
 V V IY  
 WAITING W EY DX IX NG  
 WAIT W EY TD  
 WANNA W AA N AX  
 WANTED W AO N T AX DD  
 WANTS W AA N TS  
 WANT W AA N TD  
 WASHINGTON'S W AA SH IX NG T AX N Z  
 WASHINGTON W AA SH IX NG T AX N  
 WASN'T W AA Z AX N TD  
 WAS W AA Z  
 WAYS W EY Z  
 WAY W EY  
 W D AH B AX L Y UW  
 WEDNESDAYS W EH N Z D EY Z  
 WEDNESDAY'S W EH N Z D IY Z  
 WEDNESDAY W EH N Z D IY  
 WE'D W IY DD  
 WEEKDAYS W IY KD D EY Z  
 WEEKDAY W IY KD D EY  
 WEEKENDS W IY K EH N D Z  
 WEEKEND W IY K EH N DD  
 WEEKLY W IY K L IY  
 WEEKS W IY K S



WEEK W IY KD  
 WEIGHS W EY Z  
 WEIGHT W EY TD  
 WELL W EH L  
 WENT W EH N TD  
 WERE W AXR  
 WE'RE W IY R  
 WESTCHESTER W EH S T CH EH S T AXR  
 WESTERN W EH S T AXR N  
 WEST W EH S TD  
 WE'VE W IY V  
 WE W IY  
 WHAT'RE W AH DX AXR  
 WHAT'S W AH TS  
 WHATS W AX TS  
 WHAT W AH TD  
 WHEN'S W EH N Z  
 WHEN W EH N  
 WHERE'S W EH R Z  
 WHERE W EH R  
 WHETHER W EH DH AXR  
 WHICH W IH CH  
 WHILE W AY L  
 WHO HH UW  
 WHOLE HH OW L  
 WHOOPS W UW P S  
 WHOSE HH UW Z  
 WHY'S W AY Z  
 WHY W AY  
 WIDE W AY DD  
 WILL W IH L  
 WINDOW W IH N D OW  
 WINGSPAN W IH NG S P AE N  
 WISCONSIN W IH S K AA N S AX N  
 WISH W IH SH  
 WITHIN W AX DH IH N  
 WITHOUT W IH TH AW TD  
 WITH W IH DH  
 WONDERING W AH N D AXR IX NG  
 WONDER W AH N D AXR  
 WOOPS W UW P S  
 WORDS W ER D Z  
 WORD W ER DD  
 WORKING W ER K IX NG  
 WORK W ER KD  
 WORLD W ER L DD  
 WORRY W ER IY  
 WORST W ER S TD  
 WORTH W ER TH  
 WOULD W UH DD  
 WOW W AW  
 WRONG R AO NG  
 WROTE R OW TD

X EH K S  
 YEAH Y AE  
 YEAR Y IH R  
 YES Y EH S  
 YET Y EH TD  
 Y\_K\_T W AY K EY T IY  
 YÖRK'S Y AO R K S  
 YORK Y AO R KD  
 YOU'RE Y UH R  
 YOUR Y AO R  
 YOU'VE Y UW V  
 YOU Y UW  
 Y W AY  
 Y\_Y\_Z W AY W AY Z IY  
 ZĒRÖ Z IH R OW  
 ZONES Z OW N Z  
 ZONE Z OW N  
 Z Z IY  
 <UNK> rej  
 <SIL> SIL  
 <BR> br  
 <GA> ga  
 <LS> ls  
 <LG> lg  
 <REG> reg

## Appendix K Word-Syllabification Lexicon

a:ax  
 abbreviated:ax b\_r\_iy v\_iy ey dx\_ax\_dd  
 abbreviation:ax b\_r\_iy v\_iy ey sh\_ax\_n  
 abbreviations:ax b\_r\_iy v\_iy ey sh\_ax\_n\_z  
 able:ey b\_ax\_l  
 about:ax b\_aw\_td  
 above:ax b\_ah\_v  
 accept:ae\_k s\_eh\_pd\_td  
 accommodate:ax k\_aa m\_ax d\_ey\_td  
 accommodations:ax k\_aa m\_ax d\_ey sh\_ax\_n\_z  
 accompanying:ax k\_ah\_m p\_ax n\_iy ix\_ng  
 across:ax k\_r\_ao\_s  
 actual:ae\_k ch\_ax w\_ax\_l  
 actually:ae\_k ch\_ax w\_ax l\_iy  
 add:ae\_dd  
 added:ae dx\_ax\_dd  
 additional:ax d\_ih sh\_ax n\_ax\_l  
 adult:ax d\_ah\_l\_td  
 adults:ax d\_ah\_l\_ts  
 advance:ax\_d v\_ae\_n\_s  
 advantage:ae\_d v\_ae\_n t\_ix\_jh  
 advertises:ae\_d v\_axr t\_ay z\_ax\_z  
 advise:ae\_d v\_ay\_z  
 after:ae\_f t\_axr  
 afternoon:ae\_f t\_axr n\_uw\_n  
 afternoons:ae\_f t\_axr n\_uw\_n\_z  
 afterwards:ae\_f t\_axr w\_axr\_d\_z  
 again:ax g\_eh\_n  
 agencies:ey jh\_ax\_n s\_iy\_z  
 ah:aa  
 airbus:eh\_r b\_ax\_s  
 aircraft:eh\_r k\_r\_ae\_f\_td  
 aircrafts:eh\_r k\_r\_ae\_f\_ts  
 air:eh\_r  
 airfare:eh\_r f\_eh\_r  
 airfares:eh\_r f\_eh\_r\_z  
 airline:eh\_r l\_ay\_n  
 airline's:eh\_r l\_ay\_n\_z  
 airlines:eh\_r l\_ay\_n\_z  
 airplane:eh\_r p\_l\_ey\_n  
 airplanes:eh\_r p\_l\_ey\_n\_z  
 airport:eh\_r p\_ao\_r\_td  
 airport's:eh\_r p\_ao\_r\_ts  
 airports:eh\_r p\_ao\_r\_ts  
 air's:eh\_r\_z  
 airways:eh\_r w\_ey\_z  
 alaska:ax l\_ae s\_k\_ax  
 alaska's:ax l\_ae s\_k\_ax\_z  
 alice:ae l\_ax\_s

all:ao\_l  
 allow:ax l\_aw  
 allowed:ax l\_aw\_dd  
 along:ax l\_ao\_ng  
 alpha:ae\_l\_f\_ax  
 also:ao\_l\_s\_ow  
 am:ae\_m  
 a\_m:ax eh\_m  
 america:ax m\_eh\_r\_ax\_k\_ax  
 american:ax m\_eh\_r\_ax\_k\_ax\_n  
 american's:ax m\_eh\_r\_ax\_k\_ax\_n\_z  
 america's:ax m\_eh\_r\_ax\_k\_ax\_z  
 amount:ax m\_aw\_n\_td  
 an:ae\_n  
 anchorage:ae\_ng\_k\_axr\_ax\_jh  
 and:ae\_n\_dd  
 angeles:ae\_n\_jh\_ax\_l\_ax\_s  
 another:ax n\_ah\_dh\_axr  
 answer:ae\_n\_s\_axr  
 antonio:ae\_n\_t\_ow\_n\_iy\_ow  
 anxious:ae\_ng\_k\_sh\_ax\_s  
 any:eh\_n\_iy  
 anymore:eh\_n\_iy\_m\_ao\_r  
 anything:eh\_n\_iy\_th\_ih\_ng  
 anytime:eh\_n\_iy\_t\_ay\_m  
 anyway:eh\_n\_iy\_w\_ey  
 anyways:eh\_n\_iy\_w\_ey\_z  
 anywhere:eh\_n\_iy\_w\_eh\_r  
 apart:ax\_p\_aa\_r\_td  
 apple:ae\_p\_ax\_l  
 apply:ax\_p\_l\_ay  
 appreciate:ax\_p\_r\_iy\_sh\_iy\_ey\_td  
 appropriate:ax\_p\_r\_ow\_p\_r\_iy\_ax\_td  
 approximate:ax\_p\_r\_aa\_k\_s\_ax\_m\_ax\_td  
 approximately:ax\_p\_r\_aa\_k\_s\_ax\_m\_ax\_td\_l\_iy  
 april:ey\_p\_r\_ax\_l  
 are:aa\_r  
 area:eh\_r\_iy\_ax  
 areas:eh\_r\_iy\_ax\_z  
 aren't:aa\_r\_ax\_n\_td  
 arizona:eh\_r\_ax\_z\_ow\_n\_ax  
 around:axr\_aw\_n\_dd  
 arrange:axr\_ey\_n\_jh  
 arranged:axr\_ey\_n\_jh\_dd  
 arrangement:axr\_ey\_n\_jh\_m\_ax\_n\_td  
 arrangements:axr\_ey\_n\_jh\_m\_ax\_n\_ts  
 arranging:axr\_ey\_n\_jh\_ix\_ng  
 arrival:axr\_ay\_v\_ax\_l  
 arrivals:axr\_ay\_v\_ax\_l\_z  
 arrive:axr\_ay\_v  
 arrives:axr\_ay\_v\_z  
 arriving:axr\_ay\_v\_ix\_ng  
 as:ae\_z

a's:ey\_z  
 aside:ax s\_ay\_dd  
 ask:ae\_s\_kd  
 asking:ae s\_k\_ix\_ng  
 aspen:ae s\_p\_ax\_n  
 assignments:ax s\_ay\_n m\_ax\_n\_ts  
 associated:ax s\_ow s\_iy ey dx\_ax\_dd  
 assuming:ax s\_uw m\_ix\_ng  
 at:ae\_td  
 atlanta:ae\_td l\_ae\_n t\_ax  
 atlanta's:ae\_td l\_ae\_n t\_ax\_z  
 a\_t\_l:ax t\_iy eh\_l  
 august:aa g\_ax\_s\_td  
 availabilities:ax v\_ey l\_ax b\_ih l\_ax dx\_iy\_z  
 availability:ax v\_ey l\_ax b\_ih l\_ax dx\_iy  
 available:ax v\_ey l\_ax b\_ax\_l  
 average:ae v\_axr ix\_jh  
 avis:ey v\_ih\_s  
 avoid:ax v\_oy\_dd  
 avoiding:ax v\_oy dx\_ix\_ng  
 back:b\_ae\_kd  
 bad:b\_ae\_dd  
 bag:b\_ae\_gd  
 bagels:b\_ey g\_ax\_l\_z  
 bags:b\_ae\_g\_z  
 baltimore:b\_ao\_l t\_ax m\_ao\_r  
 bangor:b\_ae\_ng g\_axr  
 based:b\_ey\_s\_td  
 basis:b\_ey s\_ih\_s  
 bay:b\_ey  
 b:b\_iy  
 beach:b\_iy\_ch  
 be:b\_iy  
 because:b\_ix k\_ao\_z  
 beef:b\_iy\_f  
 been:b\_ih\_n  
 before:b\_ax f\_ao\_r  
 begin:b\_ix g\_ih\_n  
 beginning:b\_ix g\_ih n\_ix\_ng  
 begins:b\_ix g\_ih\_n\_z  
 being:b\_iy ix\_ng  
 believe:b\_ax l\_iy\_v  
 belong:b\_ax l\_ao\_ng  
 below:b\_ax l\_ow  
 besides:b\_ax s\_ay\_d\_z  
 best:b\_eh\_s\_td  
 betcha:b\_eh ch\_ax  
 better:b\_eh dx\_axr  
 between:b\_ax t\_w\_iy\_n  
 big:b\_ih\_gd  
 bigger:b\_ih g\_axr  
 biggest:b\_ih g\_ax\_s\_td  
 bit:b\_ih\_td

blowing:b\_l\_ow ix\_ng  
 b\_n\_a:b\_iy eh n\_ax  
 board:b\_ao\_r\_dd  
 bodies:b\_aa dx\_iy\_z  
 body:b\_aa dx\_iy  
 boeing:b\_ow ix\_ng  
 book:b\_uh\_kd  
 booked:b\_uh\_kd\_td  
 booking:b\_uh k\_ix\_ng  
 b\_o\_s:b\_iy ow eh\_s  
 boston:b\_aa s\_t\_ax\_n  
 boston's:b\_aa s\_t\_ax\_n\_z  
 both:b\_ow\_th  
 bound:b\_aw\_n\_dd  
 boy:b\_oy  
 braniff:b\_r\_ae n\_ih\_f  
 branson:b\_r\_ae n\_s\_ax\_n  
 break:b\_r\_ey\_kd  
 breakfast:b\_r\_eh\_k f\_ax\_s\_td  
 breaking:b\_r\_ey k\_ix\_ng  
 bring:b\_r\_ih\_ng  
 broad:b\_r\_ao\_dd  
 bucks:b\_ah\_k\_s  
 budget:b\_ah\_jh\_ax\_td  
 built:b\_ih\_l\_td  
 burbank:b\_er b\_ae\_ng\_kd  
 bus:b\_ah\_s  
 buses:b\_ah s\_ax\_z  
 busiest:b\_ih z\_iy ax\_s\_td  
 business:b\_ih\_z n\_ax\_s  
 busses:b\_ah s\_ax\_z  
 busy:b\_ih z\_iy  
 but:b\_ah\_td  
 button:b\_ah t\_ax\_n  
 buy:b\_ay  
 b\_w\_i:b\_iy d\_ah b\_ax\_l y\_uw ay  
 by:b\_ay  
 cab:k\_ae\_bd  
 california:k\_ae l\_ax f\_ao\_r n\_y\_ax  
 called:k\_ao\_l\_dd  
 calling:k\_ao l\_ix\_ng  
 call:k\_ao\_l  
 canada:k\_ae n\_ax dx\_ax  
 canadian:k\_ax n\_ey dx\_iy ax\_n  
 cancel:k\_ae\_n s\_ax\_l  
 can:k\_ae\_n  
 can't:k\_ae\_n\_td  
 capacities:k\_ax p\_ae s\_ax dx\_iy\_z  
 capacity:k\_ax p\_ae s\_ax dx\_iy  
 captain:k\_ae\_pd t\_ax\_n  
 card:k\_aa\_r\_dd  
 care:k\_eh\_r  
 car:k\_aa\_r

carolina:k\_eh r\_ax l\_ay n\_ax  
 carried:k\_ae r\_iy\_dd  
 carrier:k\_ae r\_iy\_axr  
 carriers:k\_ae r\_iy\_axr\_z  
 carries:k\_ae r\_iy\_z  
 carry:k\_ae r\_iy  
 cars:k\_aa\_r\_z  
 case:k\_ey\_s  
 catch:k\_ae\_ch  
 categories:k\_ae dx\_ax g\_ao r\_iy\_z  
 category:k\_ae dx\_ax g\_ao r\_iy  
 center:s\_eh\_n t\_axr  
 certain:s\_er t\_ax\_n  
 chance:ch\_ae\_n\_s  
 change:ch\_ey\_n\_jh  
 changed:ch\_ey\_n\_jh\_dd  
 charge:ch\_aa\_r\_jh  
 charges:ch\_aa\_r\_jh\_ax\_z  
 charlotte:sh\_aa\_r l\_ax\_td  
 charlottesville:sh\_aa\_r l\_ax\_ts v\_ih\_l  
 charlotteville:sh\_aa\_r l\_ax\_t v\_ih\_l  
 cheap:ch\_iy\_pd  
 cheaper:ch\_iy\_p\_axr  
 cheapest:ch\_iy\_p\_ax\_s\_td  
 cheaply:ch\_iy\_p\_l\_iy  
 check:ch\_eh\_kd  
 checking:ch\_eh k\_ix\_ng  
 cheers:ch\_ih\_r\_z  
 cheese:ch\_iy\_z  
 chicago:sh\_ax k\_aa g\_ow  
 chicken:ch\_ih k\_ax\_n  
 child:ch\_ay\_l\_dd  
 children's:ch\_ih\_l d\_r\_ax\_n\_z  
 choice:ch\_oy\_s  
 choices:ch\_oy\_s\_ax\_z  
 cholesterol:k\_ax l\_eh s\_t\_axr ao\_l  
 choose:ch\_uw\_z  
 chosen:ch\_ow\_z\_ax\_n  
 christmas:k\_r\_ih s\_m\_ax\_s  
 cincinnati:s\_ih\_n s\_ax\_n\_ae dx\_iy  
 cities:s\_ih dx\_iy\_z  
 city:s\_ih dx\_iy  
 classed:k\_l\_ae\_s\_td  
 classes:k\_l\_ae s\_ax\_z  
 class:k\_l\_ae\_s  
 clear:k\_l\_ih\_r  
 cleveland:k\_l\_iy\_v l\_ax\_n\_dd  
 click:k\_l\_ih\_kd  
 close:k\_l\_ow\_s  
 closer:k\_l\_ow s\_axr  
 closest:k\_l\_ow s\_ax\_s\_td  
 coach:k\_ow\_ch  
 coast:k\_ow\_s\_td

code:k\_ow\_dd  
codes:k\_ow\_d\_z  
coffee:k\_aa\_f\_iy  
colorado:k\_aa\_l\_axr aa\_dx\_ow  
columbia:k\_ax\_l\_ah\_m\_b\_iy\_ax  
columbus:k\_ax\_l\_ah\_m\_b\_ax\_s  
column:k\_aa\_l\_ax\_m  
combination:k\_aa\_m\_b\_ax\_n\_ey\_sh\_ax\_n  
combined:k\_ax\_m\_b\_ay\_n\_dd  
come:k\_ah\_m  
comes:k\_ah\_m\_z  
comfortable:k\_ah\_m\_f\_axr\_dx\_ax\_b\_ax\_l  
comfort:k\_ah\_m\_f\_axr\_td  
coming:k\_ah\_m\_ix\_ng  
commercial:k\_ax\_m\_er\_sh\_ax\_l  
commercials:k\_ax\_m\_er\_sh\_ax\_l\_z  
common:k\_aa\_m\_ax\_n  
commuter:k\_ax\_m\_y\_uw\_dx\_axr  
companies:k\_ah\_m\_p\_ax\_n\_iy\_z  
companion:k\_ax\_m\_p\_ae\_n\_y\_ax\_n  
company:k\_ah\_m\_p\_ax\_n\_iy  
compare:k\_ax\_m\_p\_eh\_r  
competition:k\_aa\_m\_p\_ax\_t\_ih\_sh\_ax\_n  
complete:k\_ax\_m\_p\_l\_iy\_td  
computer:k\_ax\_m\_p\_y\_uw\_dx\_axr  
concerning:k\_ax\_n\_s\_er\_n\_ix\_ng  
conditions:k\_ax\_n\_d\_ih\_sh\_ax\_n\_z  
confirm:k\_ax\_n\_f\_er\_m  
confused:k\_ax\_n\_f\_y\_uw\_z\_dd  
congestion:k\_ax\_n\_jh\_eh\_s\_ch\_ax\_n  
connecticut:k\_ax\_n\_eh\_dx\_ax\_k\_ax\_td  
connecting:k\_ax\_n\_eh\_kd\_t\_ix\_ng  
connection:k\_ax\_n\_eh\_k\_sh\_ax\_n  
connections:k\_ax\_n\_eh\_k\_sh\_ax\_n\_z  
connect:k\_ax\_n\_eh\_kd\_td  
connects:k\_ax\_n\_eh\_kd\_ts  
consecutive:k\_ax\_n\_s\_eh\_k\_y\_ax\_dx\_ix\_v  
consist:k\_ax\_n\_s\_ih\_s\_td  
context:k\_aa\_n\_t\_eh\_k\_s\_td  
continental:k\_aa\_n\_t\_ax\_n\_eh\_n\_t\_ax\_l  
continental's:k\_aa\_n\_t\_ax\_n\_eh\_n\_t\_ax\_l\_z  
continent:k\_aa\_n\_t\_ax\_n\_ax\_n\_td  
continue:k\_ax\_n\_t\_ih\_n\_y\_uw  
continues:k\_ax\_n\_t\_ih\_n\_y\_uw\_z  
continuing:k\_ax\_n\_t\_ih\_n\_y\_uw\_ix\_ng  
convenient:k\_ax\_n\_v\_iy\_n\_y\_ax\_n\_td  
cooked:k\_uh\_kd\_td  
cool:k\_uw\_l  
coordinate:k\_ow\_ao\_r\_d\_ax\_n\_ax\_td  
correction:k\_axr\_eh\_k\_sh\_ax\_n  
correct:k\_axr\_eh\_kd\_td  
correspond:k\_ao\_r\_ax\_s\_p\_aa\_n\_dd  
corresponds:k\_ao\_r\_ax\_s\_p\_aa\_n\_d\_z



costing:k\_ao s\_t\_ix\_ng  
 cost:k\_aa\_s\_td  
 costs:k\_aa\_s\_ts  
 could:k\_uh\_dd  
 count:k\_aw\_n\_td  
 country:k\_ah\_n t\_r\_iy  
 county:k\_aw\_n t\_iy  
 covered:k\_ah v\_axr\_dd  
 cover:k\_ah v\_axr  
 craft:k\_r\_ae\_f\_td  
 cream:k\_r\_iy\_m  
 crew:k\_r\_uw  
 cross:k\_r\_ao\_s  
 c:s\_iy  
 c's:s\_iy\_z  
 current:k\_er ax\_n\_td  
 currently:k\_er ax\_n td l\_iy  
 c\_v\_g:s\_iy v\_iy jh\_iy  
 dailies:d\_ey l\_iy\_z  
 daily:d\_ey l\_iy  
 dallas:d\_ae l\_ax\_s  
 dang:d\_ae\_ng  
 darn:d\_aa\_r\_n  
 dash:d\_ae\_sh  
 database:d\_ey dx\_ax b\_ey\_s  
 data:d\_ey dx\_ax  
 date:d\_ey\_td  
 dates:d\_ey\_ts  
 day:d\_ey  
 days:d\_ey\_z  
 d\_c:d\_iy s\_iy  
 d:d\_iy  
 dear:d\_ih\_r  
 december:d\_ax s\_ah\_m b\_axr  
 decide:d\_ax s\_ay\_dd  
 define:d\_ax f\_ay\_n  
 delays:d\_ax l\_ey\_z  
 delete:d\_ax l\_iy\_td  
 delta:d\_ah\_l t\_ax  
 delta's:d\_ah\_l t\_ax\_z  
 den:d\_ah\_n  
 d\_e\_n:d\_iy iy ah\_n  
 denmark:d\_ah\_n m\_aa\_r\_kd  
 denver:d\_ah\_n v\_axr  
 depart:d\_ax p\_aa\_r\_td  
 departing:d\_ax p\_aa\_r dx\_ix\_ng  
 departs:d\_ax p\_aa\_r\_ts  
 departure:d\_ax p\_aa\_r ch\_axr  
 departures:d\_ax p\_aa\_r ch\_axr\_z  
 depend:d\_ax p\_ah\_n\_dd  
 dependent:d\_ax p\_ah\_n d\_ax\_n\_td  
 descending:d\_ax s\_ah\_n d\_ix\_ng  
 describe:d\_ax s\_k\_r\_ay\_bd

description:d\_ax s\_k\_r\_ih\_p sh\_ax\_n  
 descriptions:d\_ax s\_k\_r\_ih\_p sh\_ax\_n\_z  
 designated:d\_eh z\_ix\_g n\_ey dx\_ax\_dd  
 designate:d\_eh z\_ix\_g n\_ey\_td  
 designation:d\_eh z\_ix\_g n\_ey sh\_ax\_n  
 designations:d\_eh z\_ix\_g n\_ey sh\_ax\_n\_z  
 desire:d\_ax z\_ay axr  
 destination:d\_eh s\_t\_ax n\_ey sh\_ax\_n  
 destinations:d\_eh s\_t\_ax n\_ey sh\_ax\_n\_z  
 details:d\_ax t\_ey\_l\_z  
 determine:d\_ax t\_er m\_ax\_n  
 detroit:d\_ax t\_r\_oy\_td  
 d\_fw:d\_iy eh\_f d\_ah b\_ax\_l y\_uw  
 diabetic:d\_ay ax b\_eh dx\_ix\_kd  
 did:d\_ih dd  
 didn't:d\_ih d\_ax\_n\_td  
 diego:d\_ix ey g\_ow  
 diego's:d\_iy ey g\_ow\_z  
 diet:d\_ay ax\_td  
 dietetic:d\_ay ax t\_eh dx\_ix\_kd  
 difference:d\_ih f\_axr ax\_n\_s  
 differences:d\_ih f\_axr ax\_n s\_ax\_z  
 different:d\_ih f\_axr ax\_n\_td  
 dinner:d\_ih n\_axr  
 dinners:d\_ih n\_axr\_z  
 dinnertime:d\_ih n\_axr t\_ay\_m  
 direct:d\_axr eh\_kd\_td  
 directions:d\_axr eh\_k sh\_ax\_n\_z  
 directly:d\_axr eh\_kd\_td l\_iy  
 discount:d\_ih s\_k\_aw\_n\_td  
 discounted:d\_ih s\_k\_aw\_n t\_ax\_dd  
 discounts:d\_ih s\_k\_aw\_n ts  
 disneyland:d\_ih\_z n\_iy l\_ae\_n\_dd  
 display:d\_ih s\_p\_l\_ey  
 displayed:d\_ih s\_p\_l\_ey\_dd  
 displaying:d\_ih s\_p\_l\_ey ix\_ng  
 distance:d\_ih s\_t\_ax\_n\_s  
 distances:d\_ih s\_t\_ax\_n s\_ax\_z  
 district:d\_ih s\_t\_r\_ih\_kd\_td  
 do:d\_uw  
 does:d\_ah\_z  
 doesn't:d\_ah z\_ax\_n\_td  
 doing:d\_uw ix\_ng  
 dollar:d\_aa l\_axr  
 dollars:d\_aa l\_axr\_z  
 done:d\_ah\_n  
 don't:d\_ow\_n\_td  
 double:d\_ah b\_ax\_l  
 douglas:d\_ah g\_l\_ax\_s  
 down:d\_aw\_n  
 downtown:d\_aw\_n t\_aw\_n  
 drinks:d\_r\_ih\_ng\_k\_s  
 drive:d\_r\_ay\_v

d\_t\_w:d\_iy t\_iy d\_ah b\_ax\_l y\_uw  
 dulles:d\_ah l\_ax\_s  
 duration:d\_uh r\_ey sh\_ax\_n  
 during:d\_uh r\_ix\_ng  
 each:iy\_ch  
 earlier:er l\_iy axr  
 earliest:er l\_iy ax\_s\_td  
 early:er l\_iy  
 easily:iy z\_ax l\_iy  
 eastern:iy s\_t\_axr\_n  
 eastern's:iy s\_t\_axr\_n\_z  
 east:iy\_s\_td  
 eat:iy\_td  
 economical:eh k\_ax n\_aa m\_ix k\_ax\_l  
 economic:eh k\_ax n\_aa m\_ix\_kd  
 economy:ix k\_aa n\_ax m\_iy  
 eighteen:ey t\_iy\_n  
 eighteenth:ey t\_iy\_n\_th  
 eight:ey\_td  
 eighth:ey\_td\_th  
 eighty:ey dx\_iy  
 either:iy dh\_axr  
 e:iy  
 elapsed:ax l\_ae\_p\_s\_td  
 el:eh\_l  
 eleven:ax l\_eh v\_ax\_n  
 elevens:ax l\_eh v\_ax\_n\_z  
 eleventh:ax l\_eh v\_ax\_n\_th  
 eligible:eh l\_ax jh\_ax b\_ax\_l  
 eliminate:ax l\_ih m\_ax n\_ey\_td  
 else:eh\_l\_s  
 end:eh\_n\_dd  
 ending:eh\_n d\_ix\_ng  
 ends:eh\_n\_d\_z  
 engine:eh\_n jh\_ax\_n  
 england:ih\_ng g\_l\_ax\_n\_dd  
 english:ih\_ng g\_l\_ih\_sh  
 enough:ax n\_ah\_f  
 enroute:eh\_n r\_uw\_td  
 enter:eh\_n t\_axr  
 entire:eh\_n t\_ay axr  
 equal:iy k\_w\_ax\_l  
 equipment:ix k\_w\_ih\_p m\_ax\_n\_td  
 equivalent:ix k\_w\_ih v\_ax l\_ax\_n\_td  
 error:eh r\_axr  
 evening:iy\_v n\_ix\_ng  
 evenings:iy\_v n\_ix\_ng\_z  
 ever:eh v\_axr  
 everyday:eh\_v r\_iy dx\_ey  
 every:eh v\_axr iy  
 everything:eh\_v r\_iy th\_ih\_ng  
 everywhere:eh\_v r\_iy w\_eh\_r  
 e\_w\_r:iy d\_ah b\_ax\_l y\_uw aa\_r

example:ix\_g z\_ae\_m p\_ax\_l  
 exceeding:ix\_k s\_iy dx\_ix\_ng  
 exclude:ix\_k s\_k\_l uw\_dd  
 excursion:ix\_k s\_k\_er zh\_ax\_n  
 excuse:ix\_k s\_k\_y uw\_s  
 exist:ix\_g z\_ih\_s\_td  
 expensive:ix\_k s\_p\_eh\_n s\_ix\_v  
 explain:ix\_k s\_p\_l\_ey\_n  
 explanation:eh\_k s\_p\_l\_ax n\_ey sh\_ax\_n  
 express:ix\_k s\_p\_r\_eh\_s  
 extra:eh\_k s\_t\_r\_ax  
 eye:ay  
 fair:f\_eh\_r  
 fall:f\_aa\_l  
 family:f\_ae m\_ax\_l\_iy  
 fare:f\_eh\_r  
 fares:f\_eh\_r\_z  
 far:f\_aa\_r  
 fastest:f\_ae s\_t\_ax\_s\_td  
 favorite:f\_ey v\_axr ax\_td  
 february:f\_eh b\_y\_ax w\_eh r\_iy  
 fees:f\_iy\_z  
 f:eh\_f  
 fewer:f\_y\_uw axr  
 fewest:f\_y\_uw ax\_s\_td  
 few:f\_y\_uw  
 field:f\_iy\_l\_dd  
 fifteen:f\_ih\_f t\_iy\_n  
 fifteenth:f\_ih\_f t\_iy\_n\_th  
 fifth:f\_ih\_f\_th  
 fifty:f\_ih\_f t\_iy  
 fight:f\_ay\_td  
 filled:f\_ih\_l\_dd  
 final:f\_ay n\_ax\_l  
 finally:f\_ay n\_ax\_l\_iy  
 find:f\_ay\_n\_dd  
 finding:f\_ay\_n d\_ix\_ng  
 fine:f\_ay\_n  
 finished:f\_ih n\_ix\_sh\_td  
 first:f\_er\_s\_td  
 fit:f\_ih\_td  
 fitting:f\_ih dx\_ix\_ng  
 five:f\_ay\_v  
 flare:f\_l\_eh\_r  
 flares:f\_l\_eh\_r\_z  
 fliers:f\_l\_ay axr\_z  
 flies:f\_l\_ay\_z  
 flight:f\_l\_ay\_td  
 flights:f\_l\_ay\_ts  
 florida:f\_l\_ao r\_ax dx\_ax  
 flown:f\_l\_ow\_n  
 flyer:f\_l\_ay axr  
 fly:f\_l\_ay

flying:f\_l\_ay ix\_ng  
 fokker:f\_aa k\_axr  
 following:f\_aa l\_ow ix\_ng  
 food:f\_uw\_dd  
 for:f\_ao\_r  
 forget:f\_axr g\_eh\_td  
 form:f\_ao\_r\_m  
 fort:f\_ao\_r\_td  
 forty:f\_ao\_r dx\_iy  
 found:f\_aw\_n\_dd  
 four:f\_ao\_r  
 fourteen:f\_ao\_r t\_iy\_n  
 fourteenth:f\_ao\_r t\_iy\_n\_th  
 fourth:f\_ao\_r\_th  
 frame:f\_r\_ey\_m  
 france:f\_r\_ae\_n\_s  
 francisco:f\_r\_ae\_n s\_ih s\_k\_ow  
 fran:f\_r\_ae\_n  
 frankfurt:f\_r\_ae\_ng\_k f\_axr\_td  
 free:f\_r\_iy  
 frequent:f\_r\_iy k\_w\_ax\_n\_td  
 frequently:f\_r\_iy k\_w\_ax\_n\_td l\_iy  
 friday:f\_r\_ay dx\_iy  
 friday's:f\_r\_ay dx\_iy\_z  
 fridays:f\_r\_ay dx\_iy\_z  
 friend:f\_r\_eh\_n\_dd  
 friends:f\_r\_eh\_n\_d\_z  
 friendship:f\_r\_eh\_n\_d sh\_ih\_pd  
 from:f\_r\_ah\_m  
 fruit:f\_r\_uw\_td  
 frustrating:f\_r\_ah s\_t\_r\_ey dx\_ix\_ng  
 fuck:f\_ah\_kd  
 full:f\_uh\_l  
 fun:f\_ah\_n  
 further:f\_er dh\_axr  
 gave:g\_ey\_v  
 general:jh\_eh n\_axr ax\_l  
 georgia:jh\_ao\_r jh\_ax  
 germany:jh\_er m\_ax n\_iy  
 get:g\_eh\_td  
 gets:g\_eh\_ts  
 getting:g\_eh dx\_ix\_ng  
 give:g\_ih\_v  
 given:g\_ih v\_ax\_n  
 gives:g\_ih v\_z  
 god:g\_aa\_dd  
 goes:g\_ow\_z  
 go:g\_ow  
 going:g\_ow ix\_ng  
 goodbye:g\_uh\_dd b\_ay  
 good:g\_uh\_dd  
 got:g\_aa\_td  
 grand:g\_r\_ae\_n\_dd

greater:g\_r\_ey dx\_axr  
 greatest:g\_r\_ey dx\_ax\_s\_td  
 great:g\_r\_ey\_td  
 ground:g\_r\_aw\_n\_dd  
 grounds:g\_r\_aw\_n\_d\_z  
 guardia:g\_w\_aa\_r dx\_iy ax  
 guess:g\_eh\_s  
 gymnastic:jh\_ih\_m n\_ae s\_t\_ix\_kd  
 gymnastics:jh\_ih\_m n\_ae s\_t\_ix\_k\_s  
 had:hh\_ae\_dd  
 half:hh\_ae\_f  
 hampshire:hh\_ae\_m\_p sh\_axr  
 happen:hh\_ae p\_ax\_n  
 hartfield:hh\_aa\_r\_t f\_iy\_l\_dd  
 hartford:hh\_aa\_r\_t f\_axr\_dd  
 has:hh\_ae\_z  
 have:hh\_ae\_v  
 having:hh\_ae v\_ix\_ng  
 hawaii:hh\_ax w\_ay iy  
 head:hh\_eh\_dd  
 heading:hh\_eh dx\_ix\_ng  
 headquarters:hh\_eh\_dd k\_w\_ao\_r dx\_axr\_z  
 hear:hh\_ih\_r  
 heathrow:hh\_iy th\_r\_ow  
 heaviest:hh\_eh v\_iy ax\_s\_td  
 hell:hh\_eh\_l  
 hello:hh\_ax l\_ow  
 help:hh\_eh\_l\_pd  
 here:hh\_ih\_r  
 hertz:hh\_eh\_r\_ts  
 h:ey\_ch  
 hey:hh\_ey  
 higher:hh\_ay axr  
 highest:hh\_ay ax\_s\_td  
 hi:hh\_ay  
 hilton:hh\_ih\_l t\_ax\_n  
 history:hh\_ih s\_t\_axr iy  
 hit:hh\_ih\_td  
 hold:hh\_ow\_l\_dd  
 holds:hh\_ow\_l\_d\_z  
 home:hh\_ow\_m  
 honolulu:hh\_aa n\_ax l\_uw l\_uw  
 hopefully:hh\_ow\_p f\_ax l\_iy  
 horse:hh\_ao\_r\_s  
 hotel:hh\_ow t\_eh\_l  
 hotels:hh\_ow t\_eh\_l\_z  
 hot:hh\_aa\_td  
 h\_o\_u:ey ch\_ow y\_uw  
 hour:aw axr  
 hours:aw axr\_z  
 houston:hh\_y\_uw s\_t\_ax\_n  
 how:hh\_aw  
 h\_p\_n:ey\_ch p\_iy eh\_n

hub:hh\_ah\_bd  
 hubs:hh\_ah\_b\_z  
 hundred:hh\_ah\_n d\_r\_ax\_dd  
 hung:hh\_ah\_ng  
 i\_a\_d:ay ax d\_iy  
 i\_a\_h:ay ax ey\_ch  
 i:ay  
 i'd:ay\_dd  
 identify:ay d\_eh\_n t\_ax f\_ay  
 if:ih\_f  
 ignore:ix\_g n\_ao\_r  
 i'll:ay\_l  
 illinois:ih l\_ax n\_oy  
 illness:ih\_l n\_ax\_s  
 i'm:ay\_m  
 in:ax\_n  
 included:ih\_n k\_l\_uw dx\_ax\_dd  
 include:ih\_n k\_l\_uw\_dd  
 includes:ih\_n k\_l\_uw\_d\_z  
 including:ih\_n k\_l\_uw dx\_ix\_ng  
 incorporated:ih\_n k\_ao\_r p\_axr ey dx\_ax\_dd  
 incorrect:ih\_n k\_axr eh\_kd\_td  
 increasing:ih\_n k\_r\_iy s\_ix\_ng  
 i\_n\_d:ay eh\_n d\_iy  
 independence:ih\_n d\_ax p\_eh\_n d\_ax\_n\_s  
 indiana:ih\_n d\_iy ae\_n\_ax  
 indianapolis:ih\_n d\_iy ax\_n\_ao p\_ax\_l\_ah\_s  
 indicate:ih\_n d\_ax k\_ey\_td  
 individual:ih\_n d\_ax v\_ih jh\_ax w\_ax\_l  
 inexpensive:ih\_n ix\_k s\_p\_eh\_n s\_ix\_v  
 inflight:ih\_n f\_l\_ay\_td  
 info:ih\_n f\_ow  
 information:ih\_n f\_axr m\_ey sh\_ax\_n  
 informations:ih\_n f\_axr m\_ey sh\_ax\_n\_z  
 inform:ih\_n f\_ao\_r\_m  
 initials:ih\_n ih sh\_ax\_l\_z  
 inner:ih\_n\_axr  
 inquire:ih\_n k\_w\_ay\_r  
 inquiring:ih\_n k\_w\_ay axr ix\_ng  
 instead:ih\_n s\_t\_eh\_dd  
 intercontinental:ih\_n t\_axr k\_aa\_n t\_ax\_n\_eh\_n t\_ax\_l  
 interested:ih\_n t\_r\_ax s\_t\_ax\_dd  
 intermediate:ih\_n t\_axr m\_iy dx\_iy ax\_td  
 international:ih\_n t\_axr n\_ao sh\_ax\_n\_ax\_l  
 interview:ih\_n t\_axr v\_y\_uw  
 into:ih\_n t\_uw  
 involves:ih\_n v\_aa\_l\_v\_z  
 irrelevant:ix\_r\_eh\_l\_ax v\_ax\_n\_td  
 is:ih\_z  
 isn't:ih\_z\_ax\_n\_td  
 issue:ih sh\_uw  
 it:ih\_td  
 itineraries:ay t\_ih\_n\_axr eh\_r\_iy\_z

itinerary:ay t\_ih n\_axr eh r\_iy  
 it's:ih\_ts  
 its:ih\_ts  
 i've:ay\_v  
 january:jh\_ae n\_y\_uw eh r\_iy  
 jaw:jh\_ao  
 jersey:jh\_er z\_iy  
 jet:jh\_eh\_td  
 jets:jh\_eh\_ts  
 j\_f\_k:jh\_ey eh\_f k\_ey  
 j:jh\_ey  
 job:jh\_aa\_bd  
 joe:jh\_ow  
 john:jh\_aa\_n  
 joining:jh\_oy n\_ix\_ng  
 jose:hh\_ow z\_ey  
 july:jh\_uw l\_ay  
 june:jh\_uw\_n  
 just:jh\_ah\_s\_td  
 kansas:k\_ae\_n z\_ax\_s  
 kate:k\_ey\_td  
 keeping:k\_iy p\_ix\_ng  
 keep:k\_iy\_pd  
 kennedy:k\_eh n\_ax dx\_iy  
 kick:k\_ih\_kd  
 kind:k\_ay\_n\_dd  
 kindly:k\_ay\_n\_d l\_iy  
 kinds:k\_ay\_n\_d\_z  
 k:k\_ey  
 known:n\_ow\_n  
 know:n\_ow  
 kosher:k\_ow sh\_axr  
 l\_a:eh l\_ax  
 laguardia:l\_ax g\_w\_aa\_r dx\_iy ax  
 lake:l\_ey\_kd  
 la:l\_aa  
 landing:l\_ae\_n d\_ix\_ng  
 landings:l\_ae\_n d\_ix\_ng\_z  
 land:l\_ae\_n\_dd  
 lands:l\_ae\_n\_d\_z  
 large:l\_aa\_r\_jh  
 larger:l\_aa\_r jh\_axr  
 largest:l\_aa\_r jh\_ax\_s\_td  
 l\_a\_s:eh l\_ax eh\_s  
 las:l\_aa\_s  
 last:l\_ae\_s\_td  
 late:l\_ey\_td  
 later:l\_ey dx\_axr  
 latest:l\_ey dx\_ax\_s\_td  
 launch:l\_ao\_n\_ch  
 law:l\_aa  
 l\_a\_x:eh l\_ax eh\_k\_s  
 laying:l\_ey ix\_ng



layover:l\_ey ow v\_axr  
 layovers:l\_ey ow v\_axr\_z  
 least:l\_iy\_s\_td  
 leave:l\_iy\_v  
 leaves:l\_iy\_v\_z  
 leaving:l\_iy v\_ix\_ng  
 left:l\_eh\_f\_td  
 leg:l\_eh\_gd  
 l:eh\_l  
 length:l\_eh\_ng\_kd\_th  
 less:l\_eh\_s  
 lester:l\_eh s\_t\_axr  
 let:l\_eh\_td  
 let's:l\_eh\_ts  
 letter:l\_eh dx\_axr  
 level:l\_eh v\_ax\_l  
 levels:l\_eh v\_ax\_l\_z  
 l\_g\_a:eh\_l jh\_iy\_ax  
 light:l\_ay\_td  
 like:l\_ay\_kd  
 limit:l\_ih m\_ax\_td  
 limo:l\_ih m\_ow  
 limousine:l\_ih m\_ax z\_iy\_n  
 limousines:l\_ih m\_ax z\_iy\_n\_z  
 line:l\_ay\_n  
 linking:l\_ih\_ng k\_ix\_ng  
 listed:l\_ih s\_t\_ax\_dd  
 listening:l\_ih s\_ax n\_ix\_ng  
 listen:l\_ih s\_ax\_n  
 listing:l\_ih s\_t\_ix\_ng  
 listings:l\_ih s\_t\_ix\_ng\_z  
 list:l\_ih\_s\_td  
 little:l\_ih dx\_ax\_l  
 live:l\_ay\_v  
 lives:l\_ih\_v\_z  
 living:l\_ih v\_ix\_ng  
 liz:l\_ih\_z  
 local:l\_ow k\_ax\_l  
 located:l\_ow k\_ey dx\_ax\_dd  
 locate:l\_ow k\_ey\_td  
 location:l\_ow k\_ey sh\_ax\_n  
 locations:l\_ow k\_ey sh\_ax\_n\_z  
 logan:l\_ow g\_ax\_n  
 logo:l\_ow g\_ow  
 london:l\_ah\_n d\_ax\_n  
 longer:l\_ao\_ng g\_axr  
 longest:l\_ao\_ng g\_ax\_s\_td  
 long:l\_ao\_ng  
 looking:l\_uh k\_ix\_ng  
 look:l\_uh\_kd  
 looks:l\_uh\_k\_s  
 loop:l\_uw\_pd  
 los:l\_ow\_s

lot:l\_aa\_td  
 lots:l\_aa\_ts  
 louis:l\_uw\_ax\_s  
 love:l\_ah\_v  
 lower:l\_ow\_axr  
 lowest:l\_ow\_ax\_s\_td  
 low:l\_ow  
 lufthansa:l\_ax\_f\_t\_ae\_n\_z\_ax  
 lunch:l\_ah\_n\_ch  
 lunchtime:l\_ah\_n\_ch\_t\_ay\_m  
 machine:m\_ix\_sh\_iy\_n  
 made:m\_ey\_dd  
 make:m\_ey\_kd  
 makes:m\_ey\_k\_s  
 making:m\_ey\_k\_ix\_ng  
 maluso:m\_ax\_l\_uw\_s\_ow  
 manufacturer:m\_ae\_n\_y\_ax\_f\_ae\_k\_ch\_axr\_axr  
 many:m\_eh\_n\_iy  
 march:m\_aa\_r\_ch  
 maryland:m\_eh\_r\_ax\_l\_ax\_n\_dd  
 massachusetts:m\_ae\_s\_ax\_ch\_uw\_s\_ax\_ts  
 master:m\_ae\_s\_t\_axr  
 match:m\_ae\_ch  
 maximum:m\_ae\_k\_s\_ax\_m\_ax\_m  
 maybe:m\_ey\_b\_iy  
 may:m\_ey  
 mcdonnell:m\_ax\_kd\_d\_aa\_n\_ax\_l  
 m\_c\_i:eh\_m\_s\_iy\_ay  
 m\_c\_o:eh\_m\_s\_iy\_ow  
 m\_d\_w:eh\_m\_d\_iy\_d\_ah\_b\_ax\_l\_y\_uw  
 meal:m\_iy\_l  
 meal's:m\_iy\_l\_z  
 meals:m\_iy\_l\_z  
 mealtime:m\_iy\_l\_t\_ay\_m  
 meaning:m\_iy\_n\_ix\_ng  
 meanings:m\_iy\_n\_ix\_ng\_z  
 mean:m\_iy\_n  
 means:m\_iy\_n\_z  
 meant:m\_eh\_n\_td  
 meet:m\_iy\_td  
 m:eh\_m  
 me:m\_iy  
 memphis:m\_eh\_m\_f\_ax\_s  
 mentioned:m\_eh\_n\_sh\_ax\_n\_dd  
 menu:m\_eh\_n\_y\_uw  
 metro:m\_eh\_t\_r\_ow  
 m\_i\_a:eh\_m\_ay\_ax  
 miami:m\_ay\_ae\_m\_iy  
 michigan:m\_ih\_sh\_ix\_g\_ax\_n  
 middle:m\_ih\_dx\_ax\_l  
 mid:m\_ih\_dd  
 midnight:m\_ih\_d\_n\_ay\_td  
 midpoint:m\_ih\_dd\_p\_oy\_n\_td

midway:m\_ih d\_w\_ey  
 midweek:m\_ih d\_w\_iy\_kd  
 midwest:m\_ih d\_w\_eh\_s\_td  
 might:m\_ay\_td  
 mileage:m\_ay l\_ax\_jh  
 miles:m\_ay\_l\_z  
 milwaukee:m\_ih\_l w\_ao k\_iy  
 mind:m\_ay\_n\_dd  
 minimal:m\_ih n\_ax m\_ax\_l  
 minimum:m\_ih n\_ax m\_ax\_m  
 minneapolis:m\_ih n\_iy ae p\_ax l\_ax\_s  
 minnesota:m\_ih n\_ax s\_ow dx\_ax  
 minus:m\_ay n\_ax\_s  
 minutes:m\_ih n\_ax\_ts  
 missouri:m\_ax z\_uh r\_iy  
 mistake:m\_ih s\_t\_ey\_kd  
 misunderstand:m\_ih s\_ax\_n d\_axr s\_t\_ah\_n\_dd  
 misunderstood:m\_ih s\_ax\_n d\_axr s\_t\_uh\_dd  
 mitchell:m\_ih ch\_ax\_l  
 model:m\_aa dx\_ax\_l  
 monday:m\_ah\_n d\_iy  
 monday's:m\_ah\_n d\_iy\_z  
 mondays:m\_ah\_n d\_iy\_z  
 money:m\_ah\_n\_iy  
 month:m\_ah\_n\_th  
 montreal:m\_ah\_n t\_r\_iy ao\_l  
 more:m\_ao\_r  
 morning:m\_ao\_r n\_ix\_ng  
 mornings:m\_ao\_r n\_ix\_ng\_z  
 most:m\_ow\_s\_td  
 mother:m\_ah dh\_axr  
 movie:m\_uw v\_iy  
 movies:m\_uw v\_iy\_z  
 m\_s\_p:eh m\_eh s\_p\_iy  
 much:m\_ah\_ch  
 muffins:m\_ah f\_ax\_n\_z  
 must:m\_ah\_s\_td  
 my:m\_ay  
 myself:m\_ay s\_ah\_l\_f  
 name:n\_ey\_m  
 names:n\_ey\_m\_z  
 nashville:n\_ah\_sh v\_ih\_l  
 nationair:n\_ey sh\_ax n\_ah\_r  
 national:n\_ah sh\_ax n\_ax\_l  
 nearest:n\_ih r\_ax\_s\_td  
 near:n\_ih\_r  
 necessarily:n\_ah s\_ax s\_ah r\_ax l\_iy  
 needed:n\_iy dx\_ax\_dd  
 needing:n\_iy dx\_ix\_ng  
 need:n\_iy\_dd  
 needs:n\_iy\_d\_z  
 n:eh\_n  
 nevada:n\_ax v\_aa dx\_ax

never:n\_eh v\_axr  
 newark:n\_uw axr\_kd  
 new:n\_uw  
 next:n\_eh\_k\_s\_td  
 nice:n\_ay\_s  
 night:n\_ay\_td  
 nights:n\_ay\_ts  
 nighttime:n\_ay\_td t\_ay\_m  
 nine:n\_ay\_n  
 niner:n\_ay n\_axr  
 nineteen:n\_ay\_n t\_iy\_n  
 nineteenth:n\_ay\_n t\_iy\_n\_th  
 ninety:n\_ay\_n t\_iy  
 ninth:n\_ay\_n\_th  
 nondirect:n\_aa\_n d\_axr eh\_kd\_td  
 nonjets:n\_aa\_n jh\_eh\_ts  
 non:n\_aa\_n  
 no:n\_ow  
 nonpoststop:n\_aa\_n p\_ow\_s\_ts\_t\_aa\_pd  
 nonstop:n\_aa\_n s\_t\_aa\_pd  
 nonstops:n\_aa\_n s\_t\_aa\_p\_s  
 noon:n\_uw\_n  
 noontime:n\_uw\_n t\_ay\_m  
 nope:n\_ow\_pd  
 north:n\_ao\_r\_th  
 northwestern:n\_ao\_r th\_w\_eh s\_t\_axr\_n  
 northwest:n\_ao\_r th\_w\_eh\_s\_td  
 nothing:n\_ah th\_ix\_ng  
 noticed:n\_ow dx\_ax\_s\_td  
 not:n\_aa\_td  
 november:n\_ow v\_eh\_m b\_axr  
 now:n\_ow  
 null:n\_ah\_l  
 number:n\_ah\_m b\_axr  
 numbers:n\_ah\_m b\_axr\_z  
 oakland:ow k\_l\_ax\_n\_dd  
 obtain:ax\_bd t\_ey\_n  
 occur:ax k\_er  
 o'clock:ax k\_l\_aa\_kd  
 october:aa\_kd t\_ow b\_axr  
 of:ah\_v  
 off:ao\_f  
 offer:ao f\_axr  
 offered:ao f\_axr\_dd  
 offers:ao f\_axr\_z  
 office:ao f\_ax\_s  
 offices:ao f\_ax s\_ax\_z  
 often:ao f\_ax\_n  
 o'hare:ow hh\_eh\_r  
 ohio:ow hh\_ay ow  
 oh:ow  
 okay:ow k\_ey  
 oklahoma:ow k\_l\_ax hh\_ow m\_ax

ok:ow k\_ey  
 omit:ow m\_ih\_td  
 on:aa\_n  
 once:w\_ah\_n\_s  
 one's:w\_ah\_n\_z  
 ones:w\_ah\_n\_z  
 one:w\_ah\_n  
 only:ow\_n l\_iy  
 ontario:aa\_n t\_eh r\_iy ow  
 oops:uw\_p\_s  
 oop:uw\_pd  
 o:ow  
 open:ow p\_ax\_n  
 operate:aa p\_axr ey\_td  
 operates:aa p\_axr ey\_ts  
 operating:aa p\_axr ey dx\_ix\_ng  
 operation:aa p\_axr ey sh\_ax\_n  
 option:aa\_p sh\_ax\_n  
 optional:aa\_p sh\_ax\_n\_ax\_l  
 options:aa\_p sh\_ax\_n\_z  
 orange:ao r\_ax\_n\_jh  
 or:ao\_r  
 ord:ao\_r\_dd  
 order:ao\_r dx\_axr  
 ordered:ao\_r dx\_axr\_dd  
 o\_r\_d:ow aa\_r d\_iy  
 organize:ao\_r g\_ax\_n\_ay\_z  
 orient:ao r\_iy eh\_n\_td  
 original:axr ih jh\_ax\_n\_ax\_l  
 origin:ao r\_ax jh\_ax\_n  
 originate:axr ih jh\_ax\_n\_ey\_td  
 originating:axr ih jh\_ax\_n\_ey dx\_ix\_ng  
 origination:axr ih jh\_ax\_n\_ey sh\_ax\_n  
 origins:ao r\_ax jh\_ih\_n\_z  
 orlando:ao\_r l\_ae\_n d\_ow  
 orleans:ao\_r l\_iy ax\_n\_z  
 other:ah dh\_axr  
 others:ah dh\_axr\_z  
 our:aw\_axr  
 out:aw\_td  
 overnight:ow v\_axr n\_ay\_td  
 over:ow v\_axr  
 own:ow\_n  
 pacific:p\_ax s\_ih f\_ix\_kd  
 pan\_am:p\_ae n\_ae\_m  
 pardon:p\_aa\_r d\_ax\_n  
 paris:p\_ae r\_ih\_s  
 parking:p\_aa\_r k\_ix\_ng  
 particular:p\_axr t\_ih k\_y\_ax l\_axr  
 party:p\_aa\_r dx\_iy  
 paso:p\_ae s\_ow  
 passage:p\_ae s\_ax\_jh  
 passages:p\_ae s\_ax jh\_ax\_z

passenger:p\_ae s\_ax\_n jh\_axr  
 passengers:p\_ae s\_ax\_n jh\_axr\_z  
 paul:p\_ao\_l  
 pay:p\_ey  
 peak:p\_iy\_kd  
 pearson:p\_ih\_r s\_ax\_n  
 pennsylvania:p\_eh\_n s\_ax\_l v\_ey n\_y\_ax  
 people:p\_iy p\_ax\_l  
 period:p\_ih r\_iy ax\_dd  
 perks:p\_er\_k\_s  
 per:p\_er  
 person:p\_er s\_ax\_n  
 petersburg:p\_iy dx\_axr\_z b\_axr\_gd  
 philadelphia:f\_ih l\_ax d\_eh\_l f\_iy ax  
 philly:f\_ih l\_iy  
 p\_h\_l:p\_iy ey ch\_eh\_l  
 phoenix:f\_iy n\_ix\_k\_s  
 picked:p\_ih\_kd\_td  
 pick:p\_ih\_kd  
 pitt:p\_ih\_td  
 pittsburgh:p\_ih\_ts b\_axr\_gd  
 place:p\_l\_ey\_s  
 places:p\_l\_ey s\_ax\_z  
 plane:p\_l\_ey\_n  
 planes:p\_l\_ey\_n\_z  
 planning:p\_l\_ae n\_ix\_ng  
 plan:p\_l\_ae\_n  
 plans:p\_l\_ae\_n\_z  
 plates:p\_l\_ey\_ts  
 please:p\_l\_iy\_z  
 pleasure:p\_l\_eh zh\_axr  
 plus:p\_l\_ah\_s  
 p\_m:p\_iy eh\_m  
 point:p\_oy\_n\_td  
 points:p\_oy\_n\_ts  
 portion:p\_ao\_r sh\_ax\_n  
 portland:p\_ao\_r\_td l\_ax\_n\_dd  
 possibilities:p\_aa s\_ax b\_ih l\_ax dx\_iy\_z  
 possible:p\_aa s\_ax b\_ax\_l  
 p:p\_iy  
 preferably:p\_r\_eh f\_axr ax b\_l\_iy  
 preference:p\_r\_eh f\_axr ax\_n\_s  
 prefer:p\_r\_ax f\_er  
 prefix:p\_r\_iy f\_ix\_k\_s  
 press:p\_r\_eh\_s  
 previous:p\_r\_iy v\_iy ax\_s  
 priced:p\_r\_ay\_s\_td  
 price:p\_r\_ay\_s  
 prices:p\_r\_ay s\_ax\_z  
 pricing:p\_r\_ay s\_ix\_ng  
 prior:p\_r\_ay axr  
 probably:p\_r\_aa b\_ax b\_l\_iy  
 problem:p\_r\_aa b\_l\_ax\_m

problems:p\_r\_aa b\_l\_ax\_m\_z  
 processing:p\_r\_aa s\_eh s\_ix\_ng  
 programmed:p\_r\_ow g\_r\_ae\_m\_dd  
 program:p\_r\_ow g\_r\_ae\_m  
 promotions:p\_r\_ax m\_ow sh\_ax\_n\_z  
 propelled:p\_r\_ax p\_eh\_l\_dd  
 proper:p\_r\_aa p\_axr  
 prop:p\_r\_aa\_pd  
 propulsion:p\_r\_ax p\_ah\_l sh\_ax\_n  
 provided:p\_r\_ax v\_ay dx\_ax\_dd  
 provide:p\_r\_ax v\_ay\_dd  
 provides:p\_r\_ax v\_ay\_d\_z  
 providing:p\_r\_ax v\_ay dx\_ix\_ng  
 public:p\_ah b\_l\_ix\_kd  
 purchase:p\_er ch\_ax\_s  
 pushed:p\_uh\_sh\_td  
 put:p\_uh\_td  
 q:k\_y\_uw  
 qualify:k\_w\_aa l\_ax f\_ay  
 quebec:k\_w\_ax b\_eh\_kd  
 queries:k\_w\_ih r\_iy\_z  
 query:k\_w\_iy r\_iy  
 question:k\_w\_eh\_s ch\_ax\_n  
 quick:k\_w\_ih\_kd  
 quickly:k\_w\_ih k\_l\_iy  
 quiet:k\_w\_ay ax\_td  
 quit:k\_w\_ih\_td  
 quoted:k\_w\_ow dx\_ax\_dd  
 r:aa\_r  
 rate:r\_ey\_td  
 rates:r\_ey\_ts  
 rather:r\_ae dh\_axr  
 rats:r\_ae\_ts  
 reaches:r\_iy ch\_ax\_z  
 reaching:r\_iy ch\_ix\_ng  
 reach:r\_iy\_ch  
 read:r\_eh\_dd  
 ready:r\_eh dx\_iy  
 really:r\_ih l\_iy  
 receive:r\_ax s\_iy\_v  
 recognize:r\_eh k\_ax\_g n\_ay\_z  
 recommend:r\_eh k\_ax m\_eh\_n\_dd  
 reconnect:r\_iy k\_ax n\_eh\_kd\_td  
 recorded:r\_ax k\_ao\_r dx\_ax\_dd  
 recording:r\_ax k\_ao\_r dx\_ix\_ng  
 record:r\_ax k\_ao\_r\_dd  
 records:r\_ax k\_ao\_r\_d\_z  
 redaye:r\_iy dx\_ay  
 red:r\_eh\_dd  
 reduced:r\_ax d\_uw\_s\_td  
 refundable:r\_ax f\_ah\_n d\_ax b\_ax\_l  
 refund:r\_ax f\_ah\_n\_dd  
 regarding:r\_ax g\_aa\_r dx\_ix\_ng

regardless:r\_ax g\_aa\_r\_d l\_ax\_s  
 regular:r\_eh g\_y\_ax l\_axr  
 relates:r\_ax l\_ey\_ts  
 relatives:r\_eh l\_ax dx\_ix\_v\_z  
 relax:r\_ax l\_ae\_k\_s  
 remain:r\_ax m\_ey\_n  
 remember:r\_ax m\_eh\_m b\_axr  
 remove:r\_iy m\_uw\_v  
 reno:r\_iy n\_ow  
 rental:r\_eh\_n t\_ax\_l  
 rentals:r\_eh\_n t\_ax\_l\_z  
 rented:r\_eh\_n t\_ax\_dd  
 renting:r\_eh\_n t\_ix\_ng  
 rent:r\_eh\_n\_td  
 reorder:r\_iy ao\_r dx\_axr  
 repeating:r\_ax p\_iy dx\_ix\_ng  
 repeat:r\_ax p\_iy\_td  
 rephrase:r\_iy f\_r\_ey\_z  
 represented:r\_eh p\_r\_ax z\_eh\_n t\_ax\_dd  
 representing:r\_eh p\_r\_ax z\_eh\_n t\_ix\_ng  
 requested:r\_ix k\_w\_eh s\_t\_ax\_dd  
 requesting:r\_ix k\_w\_eh s\_t\_ix\_ng  
 request:r\_ix k\_w\_eh\_s\_td  
 required:r\_iy k\_w\_ay axr\_dd  
 requirements:r\_ix k\_w\_ay\_r m\_ax\_n\_ts  
 require:r\_iy k\_w\_ay axr  
 requires:r\_iy k\_w\_ay axr\_z  
 requiring:r\_iy k\_w\_ay axr ix\_ng  
 reservation:r\_eh z\_axr v\_ey sh\_ax\_n  
 reservations:r\_eh z\_axr v\_ey sh\_ax\_n\_z  
 reserved:r\_ax z\_er\_v\_dd  
 reserve:r\_ax z\_er\_v  
 reset:r\_iy s\_eh\_td  
 reside:r\_ax z\_ay\_dd  
 resort:r\_ax z\_ao\_r\_td  
 resorts:r\_ax z\_ao\_r\_ts  
 respeak:r\_iy s\_p\_iy\_kd  
 respectively:r\_ax s\_p\_eh\_kd t\_ix\_v l\_iy  
 response:r\_ax s\_p\_aa\_n\_s  
 rest:r\_eh\_s\_td  
 restricted:r\_iy s\_t\_r\_ih\_kd t\_ax\_dd  
 restriction:r\_iy s\_t\_r\_ih\_k sh\_ax\_n  
 restrictions:r\_iy s\_t\_r\_ih\_k sh\_ax\_n\_z  
 restrict:r\_iy s\_t\_r\_ih\_kd\_td  
 results:r\_ax z\_ah\_l\_ts  
 returning:r\_ax t\_er\_n\_ix\_ng  
 return:r\_ax t\_er\_n  
 returns:r\_ax t\_er\_n\_z  
 reverse:r\_ix v\_er\_s  
 review:r\_iy v\_y\_uw  
 richardson:r\_ih ch\_axr\_d s\_ax\_n  
 richmond:r\_ih\_ch m\_ax\_n\_dd  
 ride:r\_ay\_dd



right:r\_ay\_td  
 riverside:r\_ih v\_axr s\_ay\_dd  
 room:r\_uw\_m  
 roughly:r\_ah f\_l\_iy  
 round:r\_aw\_n\_dd  
 route:r\_uw\_td  
 run:r\_ah\_n  
 runs:r\_ah\_n\_z  
 rush:r\_ah\_sh  
 sacrifice:s\_ae k\_r\_ax f\_ay\_s  
 safe:s\_ey\_f  
 safest:s\_ey f\_ax\_s\_td  
 safety:s\_ey\_f t\_iy  
 said:s\_eh\_dd  
 saint:s\_ey\_n\_td  
 salad:s\_ae l\_ax\_dd  
 salt:s\_ao\_l\_td  
 same:s\_ey\_m  
 sam:s\_ae\_m  
 san:s\_ae\_n  
 saturday:s\_ae dx\_axr dx\_iy  
 saturday's:s\_ae dx\_axr dx\_iy\_z  
 saturdays:s\_ae dx\_axr dx\_iy\_z  
 saver:s\_ey v\_axr  
 save:s\_ey\_v  
 saving:s\_ey v\_ix\_ng  
 saying:s\_ey ix\_ng  
 say:s\_ey  
 says:s\_eh\_z  
 scenario:s\_ax n\_eh r\_iy ow  
 scheduled:s\_k\_eh jh\_uh\_l\_dd  
 schedule:s\_k\_eh jh\_uh\_l  
 schedules:s\_k\_eh jh\_uh\_l\_z  
 scheduling:s\_k\_eh jh\_uh\_l\_ix\_ng  
 scone:s\_k\_ow\_n  
 scratch:s\_k\_r\_ae\_ch  
 screen:s\_k\_r\_iy\_n  
 screwed:s\_k\_r\_uw\_dd  
 scroll:s\_k\_r\_ow\_l  
 s\_e\_a:eh s\_iy ax  
 seafood:s\_iy f\_uw\_dd  
 search:s\_er\_ch  
 seating:s\_iy dx\_ix\_ng  
 seat:s\_iy\_td  
 seats:s\_iy\_ts  
 seattle:s\_iy ae dx\_ax\_l  
 second:s\_eh k\_ax\_n\_dd  
 section:s\_eh\_k sh\_ax\_n  
 sections:s\_eh\_k sh\_ax\_n\_z  
 seem:s\_iy\_m  
 seems:s\_iy\_m\_z  
 seen:s\_iy\_n  
 see:s\_iy

s:eh\_s  
 selected:s\_ax\_l\_eh\_kd\_t\_ax\_dd  
 selections:s\_ax\_l\_eh\_k\_sh\_ax\_n\_z  
 select:s\_ax\_l\_eh\_kd\_td  
 send:s\_eh\_n\_dd  
 sentence:s\_eh\_n\_t\_ax\_n\_s  
 separately:s\_eh\_p\_axr\_ax\_td\_l\_iy  
 september:s\_eh\_pd\_t\_eh\_m\_b\_axr  
 series:s\_ih\_r\_iy\_z  
 served:s\_er\_v\_dd  
 serve:s\_er\_v  
 serves:s\_er\_v\_z  
 serviced:s\_er\_v\_ax\_s\_td  
 service:s\_er\_v\_ax\_s  
 services:s\_er\_v\_ax\_s\_ax\_z  
 serving:s\_er\_v\_ix\_ng  
 session:s\_eh\_sh\_ax\_n  
 set:s\_eh\_td  
 seven:s\_eh\_v\_ax\_n  
 sevens:s\_eh\_v\_ax\_n\_z  
 seventeen:s\_eh\_v\_ax\_n\_t\_iy\_n  
 seventeenth:s\_eh\_v\_ax\_n\_t\_iy\_n\_th  
 seventh:s\_eh\_v\_ax\_n\_th  
 seventy:s\_eh\_v\_ax\_n\_t\_iy  
 several:s\_eh\_v\_r\_ax\_l  
 s\_f\_o:eh\_s\_eh\_f\_ow  
 shall:sh\_ae\_l  
 she:sh\_iy  
 shoot:sh\_uw\_td  
 shorter:sh\_ao\_r\_dx\_axr  
 shortest:sh\_ao\_r\_dx\_ax\_s\_td  
 shortly:sh\_ao\_r\_td\_l\_iy  
 should:sh\_uh\_dd  
 showed:sh\_ow\_dd  
 showing:sh\_ow\_ix\_ng  
 shown:sh\_ow\_n  
 show:sh\_ow  
 shows:sh\_ow\_z  
 shucks:sh\_ah\_k\_s  
 shuttle:sh\_ah\_dx\_ax\_l  
 sierra:s\_iy\_eh\_r\_ax  
 signify:s\_ih\_g\_n\_ax\_f\_ay  
 similar:s\_ih\_m\_ax\_l\_axr  
 single:s\_ih\_ng\_g\_ax\_l  
 sit:s\_ih\_td  
 six:s\_ih\_k\_s  
 six's:s\_ih\_k\_s\_ax\_z  
 sixteen:s\_ix\_k\_s\_t\_iy\_n  
 sixteenth:s\_ix\_k\_s\_t\_iy\_n\_th  
 sixth:s\_ih\_k\_s\_th  
 sixty:s\_ih\_k\_s\_t\_iy  
 sized:s\_ay\_z\_dd  
 size:s\_ay\_z

sizes:s\_ay z\_ax\_z  
 skip:s\_k\_ih\_pd  
 slash:s\_l\_ae\_sh  
 slowest:s\_l\_ow ax\_s\_td  
 smaller:s\_m\_ao l\_axr  
 smallest:s\_m\_ao l\_ax\_s\_td  
 small:s\_m\_ao\_l  
 smoking:s\_m\_ow k\_ix\_ng  
 snack:s\_n\_ae\_kd  
 snacks:s\_n\_ae\_k\_s  
 somebody:s\_ah\_m b\_aa dx\_iy  
 some:s\_ah\_m  
 something:s\_ah\_m th\_ix\_ng  
 sometime:s\_ah\_m t\_ay\_m  
 somewhere:s\_ah\_m w\_eh\_r  
 sooner:s\_uw n\_axr  
 soonest:s\_uw n\_ax\_s\_td  
 soon:s\_uw\_n  
 sorry:s\_aa\_r\_iy  
 sorted:s\_ao\_r dx\_ax\_dd  
 sort:s\_ao\_r\_td  
 so:s\_ow  
 sounds:s\_aw\_n\_d\_z  
 southern:s\_ah dh\_axr\_n  
 southwestern:s\_aw th\_w\_eh s\_t\_axr\_n  
 southwest:s\_aw th\_w\_eh\_s\_td  
 space:s\_p\_ey\_s  
 special:s\_p\_eh sh\_ax\_l  
 specials:s\_p\_eh sh\_ax\_l\_z  
 specifications:s\_p\_eh s\_ax f\_ix k\_ey sh\_ax\_n\_z  
 specific:s\_p\_ax s\_ih f\_ix\_kd  
 specifics:s\_p\_ax s\_ih f\_ix\_k\_s  
 specified:s\_p\_eh s\_ax f\_ay\_dd  
 specify:s\_p\_eh s\_ax f\_ay  
 speech:s\_p\_iy\_ch  
 speed:s\_p\_iy\_dd  
 spend:s\_p\_eh\_n\_dd  
 split:s\_p\_l\_ih\_td  
 spouse:s\_p\_aw\_s  
 standard:s\_t\_ae\_n d\_axr\_dd  
 standby:s\_t\_ae\_n\_dd b\_ay  
 stands:s\_t\_ae\_n\_d\_z  
 stand:s\_t\_ae\_n\_dd  
 stapleton:s\_t\_ey p\_ax\_l t\_ax\_n  
 starting:s\_t\_aa\_r dx\_ix\_ng  
 starts:s\_t\_aa\_r\_ts  
 start:s\_t\_aa\_r\_td  
 states:s\_t\_ey\_ts  
 state:s\_t\_ey\_td  
 status:s\_t\_ae dx\_ax\_s  
 staying:s\_t\_ey ix\_ng  
 stayover:s\_t\_ey ow v\_axr  
 stays:s\_t\_ey\_z

stay:s\_t\_ey  
 steak:s\_t\_ey\_kd  
 still:s\_t\_ih\_l  
 stockholm:s\_t\_aa\_k hh\_ow\_l\_m  
 stopovers:s\_t\_aa p\_ow v\_axr\_z  
 stopover:s\_t\_aa p\_ow v\_axr  
 stopping:s\_t\_aa p\_ix\_ng  
 stops:s\_t\_aa\_p\_s  
 stop:s\_t\_aa\_pd  
 straight:s\_t\_r\_ey\_td  
 street:s\_t\_r\_iy\_td  
 strict:s\_t\_r\_ih\_kd\_td  
 stuck:s\_t\_ah\_kd  
 student:s\_t\_uw d\_ax\_n\_td  
 stupid:s\_t\_uw p\_ax\_dd  
 subway:s\_ah\_b w\_ey  
 subways:s\_ah\_b w\_ey\_z  
 such:s\_ah\_ch  
 sugar:sh\_uh g\_axr  
 suggest:s\_ax\_g jh\_eh\_s\_td  
 summer:s\_ah m\_axr  
 sunday:s\_ah\_n d\_ey  
 sunday's:s\_ah\_n d\_ey\_z  
 sundays:s\_ah\_n d\_ey\_z  
 supersaver:s\_uw p\_axr s\_ey v\_axr  
 super:s\_uw p\_axr  
 supper:s\_ah p\_axr  
 supply:s\_ax p\_l\_ay  
 sure:sh\_uh\_r  
 symbol:s\_ih\_m b\_ax\_l  
 symbols:s\_ih\_m b\_ax\_l\_z  
 system:s\_ih s\_t\_ax\_m  
 table:t\_ey b\_ax\_l  
 tacoma:t\_ax k\_ow m\_ax  
 takeoffs:t\_ey k\_ao\_f\_s  
 takeoff:t\_ey k\_ao\_f  
 takes:t\_ey\_k\_s  
 take:t\_ey\_kd  
 taking:t\_ey k\_ix\_ng  
 talking:t\_ao k\_ix\_ng  
 talk:t\_ao\_kd  
 tallahassee:t\_ae l\_ax hh\_ae s\_iy  
 tampa:t\_ae\_m p\_ax  
 taxicabs:t\_ae\_k s\_iy k\_ae\_b\_z  
 taxis:t\_ae\_k s\_iy\_z  
 taxi:t\_ae\_k s\_iy  
 team:t\_iy\_m  
 telephone:t\_eh l\_ax f\_ow\_n  
 tell:t\_eh\_l  
 tennessee:t\_eh n\_ax s\_iy  
 tens:t\_eh\_n\_z  
 ten:t\_eh\_n  
 tenth:t\_eh\_n\_th

terminal:t\_er m\_ax n\_ax\_l  
 terminates:t\_er m\_ax n\_ey\_ts  
 terminating:t\_er m\_ax n\_ey dx\_ix\_ng  
 terms:t\_er\_m\_z  
 texas:t\_eh\_k\_s\_ax\_s  
 textbook:t\_eh\_k\_s\_td b\_uh\_kd  
 than:dh\_ae\_n  
 thanks:th\_ae\_ng\_k\_s  
 thank:th\_ae\_ng\_kd  
 that:dh\_ae\_td  
 that'll:dh\_ae dx\_ax\_l  
 that's:dh\_ae\_ts  
 the:dh\_ax  
 their:dh\_eh\_r  
 them:dh\_eh\_m  
 then:dh\_eh\_n  
 thereafter:dh\_eh\_r\_ae\_f t\_axr  
 there:dh\_eh\_r  
 there's:dh\_eh\_r\_z  
 these:dh\_iy\_z  
 they:dh\_ey  
 they're:dh\_eh\_r  
 thing:th\_ih\_ng  
 think:th\_ih\_ng\_kd  
 third:th\_er\_dd  
 thirteen:th\_er t\_iy\_n  
 thirteenth:th\_er t\_iy\_n\_th  
 thirtieth:th\_er dx\_iy\_ax\_th  
 thirty:th\_er dx\_iy  
 this:dh\_ih\_s  
 those:dh\_ow\_z  
 thought:th\_ao\_td  
 thousand:th\_aw\_z\_ax\_n\_dd  
 threes:th\_r\_iy\_z  
 three:th\_r\_iy  
 thrift:th\_r\_ih\_f\_td  
 thrifty:th\_r\_ih\_f t\_iy  
 through:th\_r\_uw  
 thursdays:th\_er\_z d\_ey\_z  
 thursday:th\_er\_z d\_ey  
 tickets:t\_ih\_k\_ax\_ts  
 ticket:t\_ih\_k\_ax\_td  
 till:t\_ih\_l  
 time's:t\_ay\_m\_z  
 times:t\_ay\_m\_z  
 time:t\_ay\_m  
 tip:t\_ih\_pd  
 toast:t\_ow\_s\_td  
 today's:t\_ax\_d\_ey\_z  
 today:t\_ax\_d\_ey  
 told:t\_ow\_l\_dd  
 tomorrow's:t\_ax\_m\_aa\_r\_ow\_z  
 tomorrow:t\_ax\_m\_aa\_r\_ow

tonight:t\_ax n\_ay\_td  
 took:t\_uh\_kd  
 too:t\_uw  
 toronto:t\_axr aa\_n t\_ow  
 total:t\_ow dx\_ax\_l  
 to:t\_uw  
 tough:t\_ah\_f  
 touring:t\_uh r\_ix\_ng  
 tourist:t\_uh r\_ax\_s\_td  
 tour:t\_uh\_r  
 towards:t\_ax w\_ao\_r\_d\_z  
 toward:t\_ax w\_ao\_r\_dd  
 tower:t\_aw\_axr  
 town:t\_aw\_n  
 t\_p\_a:t\_iy p\_iy\_ax  
 traffic:t\_r\_ae f\_ix\_kd  
 trains:t\_r\_ey\_n\_z  
 train:t\_r\_ey\_n  
 transcontinental:t\_r\_ae\_n\_z k\_aa\_n t\_ax\_n\_ah\_n t\_ax\_l  
 transferring:t\_r\_ae\_n s\_f\_er ix\_ng  
 transfer:t\_r\_ae\_n s\_f\_er  
 transportations:t\_r\_ae\_n s\_p\_r\_t\_ey sh\_ax\_n\_z  
 transportation:t\_r\_ae\_n s\_p\_axr t\_ey sh\_ax\_n  
 transport:t\_r\_ae\_n s\_p\_ao\_r\_td  
 trans:t\_r\_ae\_n\_z  
 trap:t\_r\_ae\_pd  
 traveled:t\_r\_ae v\_ax\_l\_dd  
 traveling:t\_r\_ae v\_ax\_l\_ix\_ng  
 travels:t\_r\_ae v\_ax\_l\_z  
 travel:t\_r\_ae v\_ax\_l  
 triangle:t\_r\_ay ae\_ng g\_ax\_l  
 trips:t\_r\_ih\_p\_s  
 trip:t\_r\_ih\_pd  
 trying:t\_r\_ay ix\_ng  
 try:t\_r\_ay  
 t:t\_iy  
 tucson:t\_uw s\_aa\_n  
 tuesdays:t\_uw\_z d\_ey\_z  
 tuesday:t\_uw\_z d\_iy  
 turboprop:t\_er b\_ow p\_r\_aa\_pd  
 turns:t\_er\_n\_z  
 t\_w\_a:t\_iy d\_ah b\_ax\_l y\_uw\_ax  
 twelfth:t\_w\_ah\_l\_f\_th  
 twelve:t\_w\_ah\_l\_v  
 twentieth:t\_w\_ah\_n t\_iy\_ax\_th  
 twenty:t\_w\_ah\_n t\_iy  
 twos:t\_uw\_z  
 two:t\_uw  
 types:t\_ay\_p\_s  
 type:t\_ay\_pd  
 unable:ax\_n\_ey b\_ax\_l  
 unbook:ax\_n b\_uh\_kd  
 under:ah\_n d\_axr

underground:ah\_n d\_axr g\_r\_aw\_n\_dd  
 understand:ah\_n d\_axr s\_t\_ae\_n\_dd  
 united's:y\_uw n\_ay dx\_ax\_d\_z  
 united:y\_uw n\_ay dx\_ax\_dd  
 unpressurized:ax\_n p\_r\_eh sh\_axr ay\_z\_dd  
 until:ax\_n t\_ih\_l  
 up:ah\_pd  
 upon:ax p\_aa\_n  
 us:ah\_s  
 u\_s\_air:y\_uw eh s\_eh\_r  
 used:y\_uw\_z\_dd  
 uses:y\_uw s\_ax\_z  
 use:y\_uw\_s  
 using:y\_uw z\_ix\_ng  
 utah:y\_uw t\_ao  
 u:y\_uw  
 vacation:v\_ey k\_ey sh\_ax\_n  
 valid:v\_ae l\_ax\_dd  
 vans:v\_ae\_n\_z  
 various:v\_eh r\_iy ax\_s  
 vegas:v\_ey g\_ax\_s  
 vegetarian:v\_eh jh\_ax t\_eh r\_iy ax\_n  
 very:v\_eh r\_iy  
 via:v\_ay ax  
 vicinity:v\_ax s\_ih n\_ax dx\_iy  
 virginia:v\_axr jh\_ih n\_y\_ax  
 visa:v\_iy z\_ax  
 visit:v\_ih z\_ax\_td  
 vocabulary:v\_ow k\_ae b\_y\_ax l\_eh r\_iy  
 v:v\_iy  
 waiting:w\_ey dx\_ix\_ng  
 wait:w\_ey\_td  
 wanna:w\_aa n\_ax  
 wanted:w\_ao\_n t\_ax\_dd  
 wants:w\_aa\_n\_ts  
 want:w\_aa\_n\_td  
 washington's:w\_aa sh\_ix\_ng t\_ax\_n\_z  
 washington:w\_aa sh\_ix\_ng t\_ax\_n  
 wasn't:w\_aa z\_ax\_n\_td  
 was:w\_aa\_z  
 ways:w\_ey\_z  
 way:w\_ey  
 w:d\_ah b\_ax\_l y\_uw  
 wednesdays:w\_eh\_n\_z d\_ey\_z  
 wednesday's:w\_eh\_n\_z d\_iy\_z  
 wednesday:w\_eh\_n\_z d\_iy  
 we'd:w\_iy\_dd  
 weekdays:w\_iy\_kd d\_ey\_z  
 weekday:w\_iy\_kd d\_ey  
 weekends:w\_iy k\_eh\_n\_d\_z  
 weekend:w\_iy k\_eh\_n\_dd  
 weekly:w\_iy k\_l\_iy  
 weeks:w\_iy\_k\_s

week:w\_iy\_kd  
 weighs:w\_ey\_z  
 weight:w\_ey\_td  
 well:w\_eh\_l  
 went:w\_eh\_n\_td  
 were:w\_axr  
 we're:w\_iy\_r  
 westchester:w\_eh\_s\_t ch\_eh s\_t\_axr  
 western:w\_eh s\_t\_axr\_n  
 west:w\_eh\_s\_td  
 we've:w\_iy\_v  
 we:w\_iy  
 what're:w\_ah dx\_axr  
 what's:w\_ah\_ts  
 whats:w\_ax\_ts  
 what:w\_ah\_td  
 when's:w\_eh\_n\_z  
 when:w\_eh\_n  
 where's:w\_eh\_r\_z  
 where:w\_eh\_r  
 whether:w\_eh dh\_axr  
 which:w\_ih\_ch  
 while:w\_ay\_l  
 who:hh\_uw  
 whole:hh\_ow\_l  
 whoops:w\_uw\_p\_s  
 whose:hh\_uw\_z  
 why's:w\_ay\_z  
 why:w\_ay  
 wide:w\_ay\_dd  
 will:w\_ih\_l  
 window:w\_ih\_n d\_ow  
 wingspan:w\_ih\_ng s\_p\_ae\_n  
 wisconsin:w\_ih s\_k\_aa\_n s\_ax\_n  
 wish:w\_ih\_sh  
 within:w\_ax dh\_ih\_n  
 without:w\_ih th\_aw\_td  
 with:w\_ih\_dh  
 wondering:w\_ah\_n d\_axr ix\_ng  
 wonder:w\_ah\_n d\_axr  
 woops:w\_uw\_p\_s  
 words:w\_er\_d\_z  
 word:w\_er\_dd  
 working:w\_er k\_ix\_ng  
 work:w\_er\_kd  
 world:w\_er\_l\_dd  
 worry:w\_er iy  
 worst:w\_er\_s\_td  
 worth:w\_er\_th  
 would:w\_uh\_dd  
 wow:w\_aw  
 wrong:r\_ao\_ng  
 wrote:r\_ow\_td



x:eh\_k\_s  
yeah:y\_ae  
year:y\_ih\_r  
yes:y\_eh\_s  
yet:y\_eh\_td  
y\_k\_t:w\_ay k\_ey t\_iy  
york's:y\_ao\_r\_k\_s  
york:y\_ao\_r\_kd  
you're:y\_uh\_r  
your:y\_ao\_r  
you've:y\_uw\_v  
you:y\_uw  
y:w\_ay  
y\_y\_z:w\_ay w\_ay z\_iy  
zero:z\_ih\_r\_ow  
zones:z\_ow\_n\_z  
zone:z\_ow\_n  
z:z\_iy

**Appendix L SONIC Symbol Set (from Pellom and Hacıoglu, 2005, p. 12)**

Phone	Example	Phone	Example	Phone	Example	Phone	Example
AA	f <u>a</u> ther	DX	bu <u>t</u> ter	KD	ta <u>k</u>	GD	mu <u>g</u>
AE	ma <u>d</u>	DH	th <u>e</u> m	JH	<u>J</u> erry	SH	sh <u>o</u> w
AH	bu <u>t</u>	EH	be <u>d</u>	K	<u>k</u> itten	T	to <u>t</u>
AO	fo <u>r</u>	ER	bi <u>r</u> d	L	<u>l</u> isten	TH	th <u>r</u> ead
AW	frow <u>n</u>	EY	sta <u>t</u> e	M	ma <u>n</u> ager	UH	ho <u>o</u> d
AX	al <u>o</u> ne	F	fr <u>i</u> end	N	na <u>n</u> cy	UW	mo <u>o</u> n
AXR	bu <u>t</u> ter	G	gro <u>w</u> n	NG	fi <u>s</u> hing	V	ve <u>r</u> y
AY	hi <u>r</u> e	HH	ha <u>d</u>	OW	co <u>n</u> e	W	wea <u>t</u> her
B	bo <u>b</u>	IH	bi <u>t</u> ter	OY	bo <u>y</u>	Y	ye <u>l</u> low
CH	ch <u>u</u> rch	IX	ro <u>s</u> es	P	po <u>p</u>	Z	bee <u>s</u>
D	do <u>n</u> 't	IY	bea <u>t</u>	R	re <u>d</u>	ZH	mea <u>s</u> ure
PD	to <u>p</u>	BD	ta <u>b</u>	S	so <u>n</u> ic	SIL	silence
TD	lo <u>t</u>	DD	ha <u>d</u>	TS	bi <u>t</u> s	br	<breathe>
ls	<lipsmack>	lg	<laughter>	ga	<garbage>		