Human-Machine Collaboration for Rapid Speech Transcription

by

Brandon C. Roy

Sc.B., Brown University (1999)

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
in partial fulfillment of the requirements for the degree of

Master of Science in Media Arts and Sciences

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2007

Author_____

Program in Media Arts and Sciences
September 5, 2007

Certified by_____

Deb Roy
Professor of Media Arts and Sciences
Program in Media Arts and Sciences
Thesis Supervisor

Accepted by_____

Prof. Deb Roy
Chairperson, Departmental Committee on Graduate Students

Human-Machine Collaboration for Rapid Speech Transcription

by

Brandon C. Roy

Submitted to the Program in Media Arts and Sciences,
School of Architecture and Planning,
on September 5, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science in Media Arts and Sciences

Abstract

Inexpensive storage and sensor technologies are yielding a new generation of massive multimedia datasets. The exponential growth in storage and processing power makes it possible to collect more data than ever before, yet without appropriate content annotation for search and analysis such corpora are of little use. While advances in data mining and machine learning have helped to automate some types of analysis, the need for human annotation still exists and remains expensive.

The Human Speechome Project is a heavily data-driven longitudinal study of language acquisition. More than 100,000 hours of audio and video recordings have been collected over a two year period to trace one child's language development at home. A critical first step in analyzing this corpus is to obtain high quality transcripts of all speech heard and produced by the child. Unfortunately, automatic speech transcription has proven to be inadequate for these recordings, and manual transcription with existing tools is extremely labor intensive and therefore expensive.

A new human-machine collaborative system for rapid speech transcription has been developed which leverages both the quality of human transcription and the speed of automatic speech processing. Machine algorithms sift through the massive dataset to find and segment speech. The results of automatic analysis are handed off to humans for transcription using newly designed tools with an optimized user interface. The automatic algorithms are tuned to optimize human performance, and errors are corrected by the human and used to iteratively improve the machine performance. When compared with other popular transcription tools, the new system is three- to six-fold faster, while preserving transcription quality. When applied to the Speechome audio corpus, over 100 hours of multitrack audio can be transcribed in about 12 hours by a single human transcriber.

Thesis Supervisor: Deb Roy
Title: Professor of Media Arts and Sciences, Program in Media Arts and Sciences

Human-Machine Collaboration for Rapid Speech Transcription

by

Brandon C. Roy

The following people served as readers for this thesis:

Thesis Reader_____

Barry Vercoe
Professor of Media Arts and Sciences
Program in Media Arts and Sciences

Thesis Reader_____

Yuri Ivanov
Principal Technical Staff
Mitsubishi Electric Research Lab

Contents

List of Figures

List of Tables

Acknowledgements

There are many people I would like to acknowledge as I write the final words of this thesis. To Prof. Deb Roy I owe the opportunity to come to the Media Lab and work with some of the most talented people I have ever met. Thanks not only for your enthusiasm for new ideas and sense of humor, but also for your honest critiques and guidance. Both have been invaluable to me over the past two years, and will be in the future as well. Also, thanks to Deb and his family for providing thousands of hours of speech for me to analyze.

Thanks to Dr. Yuri Ivanov and Prof. Barry Vercoe for graciously reading and commenting on the various incarnations of this thesis.

I'd especially like to acknowledge my friends in the Cognitive Machines group. Not only are they a great group to work with, but without them much of the system described here would not exist. In particular, Rony Kubat built the first version of TotalRecall, and left his mark on many other aspects of the system. Jethran Guinness not only wrote BlitZcribe, he also built and manages the massive disk array that is humming smoothly here at the Media Lab. Thanks to Mike Fleischman for research collaborations and coffee breaks. Philip DeCamp was kind enough to build the entire recording infrastructure before I got here. More than that, thanks to Phil(ip) for being a great office-mate, for bringing the exotic foods and wildlife of Montana and Alaska into our office, and for staying up later than me.

Special thanks to Erica Wojcik, who joined our group for the summer and helped make this thesis possible. Mutsumi Sullivan has also contributed to this work, besides keeping everything else running smoothly. Thanks to Dr. Michael Levit for providing initial code and guidance for the speech detector system. And thanks to Dr. Long Nguyen of BBN Technologies, who generously offered to help with a pilot study applying automatic speech recognition to the HSP data.

Thanks to all my friends, near and far. There are too many to list, but you know who you are. To Jeevan Kalanithi, I can only say, "I will meet you on the Mountain of Insanity."

Most of all, to my sister Ava and my parents, thanks for your inspiration and support, right from the very beginning.

Chapter 1

Introduction

Profound changes in science, technology and society are being driven by the increasing availability of sensors, storage, and processing power. Moore's Law, which predicts the exponential increase of processing power per unit cost, is being coupled with similar increases in storage capacity [58], making it possible to store and process more data than ever before. Products such as Apple's iPod or the ubiquity of digital cameras are two examples illustrating how the effect of these technological advances are not limited to technical fields, but are finding their way into our daily lives. Combined with high quality, low cost sensors such as digital video cameras and audio recorders, the necessary elements are in place for almost anyone to collect a huge, multimedia dataset. From shared video repositories such as YouTube, to personal digital archives such as MyLifeBits [20], we are sure to see exciting new applications built using these technologies.

Yet few applications are ever built with the intention of merely collecting and storing information. Although the cost of information is decreasing, it is not free. Information that is never used is only so much wasted hard drive space. One problem is that the information that is worth collecting is often hidden in a sea of information that is not of interest. Online processing can frequently serve to filter out what is unwanted, but in cases where it is unknown in advance exactly *which* information is of interest, it is better simply to keep everything. While technological advances are making it possible to *store* more information,

how to make such massive information stores easily accessible remains an open question.

The answer to this question, of course, critically depends on what sort of information is being stored, and what one wants to do with it. Finding documents on the web using a search engine is perhaps the most well known example of such a problem. Very good search engines can be built using purely automatic methods, which do not require a human librarian to index and catalog the documents according to their content. Companies like EveryZing and Blinkx are applying technologies such as speech recognition and image analysis to help annotate audio and video data, and make it searchable. Other approaches relying on human generated annotations, such as tags, are used by websites like Flickr and YouTube.

Speech transcripts are used for far more than audio and video indexing. The entertainment industry makes heavy use of transcription services to produce captions for television and movies. DVDs and many television broadcasts may optionally be viewed with these captions displayed. Medical transcription services are used to convert notes dictated by medical professionals to text records. In both of these applications, accurate transcription is critical and relies heavily on human review, if not completely manual transcription. Scientific research on human language acquisition also depends on accurate, detailed speech transcripts. This is not simply for the purpose of using the transcripts as a way to find audio (or video), but as the actual data for analysis. Statistical analyses of word frequency and grammatical analysis of the relation between words depends on these speech transcriptions. This thesis presents a system for transcribing a massive multimedia corpus for the purposes of studying human language acquisition. The work is driven by the Human Speechome Project, although the methodology should be more broadly applicable.

1.1 The Human Speechome Project

The Human Speechome Project (HSP) will study human language acquisition by collecting a new, ultra-dense audio/video corpus taken from the home of a family with a young child [46]. Data collection has been underway since the birth of the child, and will continue for between two to three years. At the time of this writing, over two years of data has been

collected. The home of the child has been augmented with eleven cameras and fourteen microphones mounted in the ceilings, in nearly every room of the house. Recording takes place for roughly ten hours per day, for nearly all of the child's waking hours. Audio is 16 bit resolution, recorded at 48KHz and video is approximately 1 megapixel resolution. This corresponds to about 200 - 300 gigabytes of data per day, at about 350 days per year, for a span of nearly 1000 days. This constitutes a dataset of unprecedented size for the study of human language learning.

There have been two major efforts toward fulfilling the goals of the Human Speechome Project. Building the recording and storage infrastructure has been a significant project [14], and work on the massive storage system continues. When complete, the disk array will be over 350 terabytes. Yet obtaining and storing the recorded data is only part of the problem. Appropriate tools and methodologies are necessary to support the kinds of studies intended for this dataset. In terms of language acquisition, speech transcripts will be essential in studying how how children learn language. Thus, the challenge presented by HSP is how to obtain transcripts of as much, if not *all*, speech from a corpus containing more than 100,000 hours of audio.

1.2 Thesis Goal, Approach, and Contributions

The goal of this thesis is to build a system that drastically decreases the time required to transcribe speech. The problem is raised by the Human Speechome Project, which will contribute the most comprehensive child language acquisition dataset to date. With fourteen audio channels recording daily, we expect to collect nearly 150,000 hours of audio over a two to three year timespan. Because of the amount of data, new tools and methodologies will have to be developed. While existing manual transcription tools are available, they are unsuitable for the HSP data for a number of reasons. First, reports of transcription times with these tools range anywhere from ten to fifty times the actual audio time being transcribed [53, 3]. That corresponds to over one million hours of labor if the tools are used without additional specialized processing, far beyond the timeframe acceptable to most researchers.

Assuming a cost of $20 per hour for human transcriber time, that is also far beyond the budget for most research groups. The nature of the data also poses problems for existing tools. While some tools are designed to handle multitrack audio [5], this is usually for audio split into separate tracks by speaker. In HSP, the audio channels overlap and tend to pick up the same sounds, and part of the problem is picking which audio channel to use for transcription. On the other hand, automatic speech recognition has advanced to a stage where it works well in specialized domains and under controlled acoustic environments, but the speaking style in a home environment, particularly one with a young child learning to speak, is still beyond the reach of even the best speech recognition systems. More detail on existing tools and experience with automatic speech recognition is provided in the next chapter, but these problems alone should illustrate the need for an approach suited to the data collected for the Human Speechome Project.

The approach this thesis takes is to combine human effort and machine processing to enable rapid transcription. The full transcription task is divided into subtasks that can be accurately and quickly performed by a machine, while the more difficult tasks relying on human judgment and understanding are performed by a human transcriber. Tools specifically designed to work with this semi-automatic approach are crucial in optimizing the human transcriber's efficiency, and in keeping the automatic system performing well. A sketch of the approach is as follows. Multichannel audio data collected in HSP is automatically distilled into into speech segments, using signal processing and pattern recognition algorithms. These speech segments are tuned to be easily transcribable by a human transcriber using the tools, in that they are not too long or too short. The transcription tool presents the detected speech segments in a list, and the transcriber need only "listen and type". This means that the human transcriber need not use the mouse to navigate the audio, or switch attention from anything other than listening to speech and typing transcriptions. Errors produced by the automatic system can be quickly identified, and the automatic system is specifically tuned to maximize the efficiency of the human transcriber.

The result of this work is a system which takes roughly two to three times the actual audio time to produce a complete transcription. For the HSP data, this is between 2.5 to 6

times faster than other tools. These are concrete numbers due to an empirical evaluation, however, in practice the factor may be even higher, since the existing tools were only evaluated on single channel audio data that they could handle. In addition, other effects such as human transcriber fatigue were not considered, but human transcribers reported preferring the system presented here over other systems. This thesis contributes a speech detection system, and presents a semi-automatic transcription methodology that tightly links human and machine performance. The resulting system will help fulfill the promise of the Human Speechome Project.

1.3 Thesis Summary

The thesis begins by presenting background material on the Human Speechome Project, and considers approaches to the transcription task. Transcription is considered from a functional standpoint, and is broken down into subtasks. This is the basis for comparing existing transcription approaches to the new approach. The specifics of the automatic components of the system are described in Chapter 3, and details on the transcription task left to the human are presented in Chapter 5. Chapter 4 provides a detailed evaluation and interpretation of the performance of the automatic components, and Chapter 6 analyzes the task which the human annotator performs. A strong link between the automatic and human elements of the system is made by studying the effects of errors resulting from the automatic system on the human annotator, and how the human can respond to these errors. Chapter 7 concludes the thesis, providing a summary of the results, directions for future work, and the potential impact on the field.

Chapter 2

Background and Motivation

The fundamental human capacity for language has long been of interest to scientific inquiry. The study of language acquisition focuses on how it is that humans, seemingly effortlessly, learn to communicate. It appears deceptively easy – we take it for granted that a newborn baby, in only a few years, will learn to speak and communicate with others. Yet building artificial systems with the same capability for language remains out of reach, nor is it obvious how it should be done. Our understanding of the mechanism for language acquisition is insufficient.

The Human Speechome Project is a study that is motivated by these questions. Broadly speaking, the goal is to understand the relationship between a child's linguistic development and the environmental stimuli they receive. While many theories of language acquisition exist, the data used to test and develop these theories have all suffered various shortcomings. One of the shortcomings the HSP dataset addresses is the problem of data *sparsity* [53]. Studies which track a child's progress with periodic samples of data taken over a long period of time fail to capture how apparently sudden changes in linguistic abilities occur (such as new vocabulary, grammatical constructions, etc.) The CHILDES archive [30] provides widely used corpora for language acquisition research, yet most of the datasets cover less than 1.5% of a child's linguistic experience, and little visual context. While data collected from a laboratory gives the researcher more control, the setting does not necessarily cap-

ture the child's natural behavior or caregiver-child interactions. For Bruner [10], studying language in "the clutter of life at home" was preferable to a well equipped laboratory. This motivates the Human Speechome Project: collect and study the most comprehensive record to date of a child's experience learning language in a natural environment. Not only will the HSP corpus be a unique scientific resource, addressing the many challenges along the way will help pave the way for future studies.

2.1 Collecting and Analyzing the HSP corpus

The recording infrastructure for the Human Speechome Project allows user control of audio and video recording, though it is designed to run continually and without intervention. Video is recorded from eleven fisheye lens cameras embedded in the ceilings, recording up to 15 frames per second at roughly 1 megapixel resolution. Fourteen boundary layer microphones in the ceilings throughout the house capture audio at 16 bit, 48KHz resolution. Because of the density of sensors and layout of the house, events usually register in multiple channels, particularly for audio.

The first step in analyzing the HSP data is to produce speech transcripts. These transcripts are needed not only to study language, they are also helpful for indexing the massive dataset, in the same way that companies like EveryZing and Blinkx use transcripts for video search. Studying how a word is used cannot be done without speech transcripts, which align blocks of text to the corresponding segments of audio. Each segment should be short enough to contain only a few words, so that from a transcript the speech can be localized. A speech transcript may also include additional information such as the speaker identity or prosody information. By linking transcripts to video annotations, such as speaker locations or head pose (the subject of a related project, see [14]), a more detailed analysis of speech in context can be performed. A first analysis of language use that highlights the possibilities of the dense HSP dataset is to trace the "lifetime of a word." For a particular word, we can look at how it was used by caregivers, the first "proto-word" versions of the word as uttered by the child, and study how it develops and how its meaning changes over time. This analysis

can only be done with speech transcripts that enable finding all the uses of a word in the corpus.

2.2 Speech Transcription

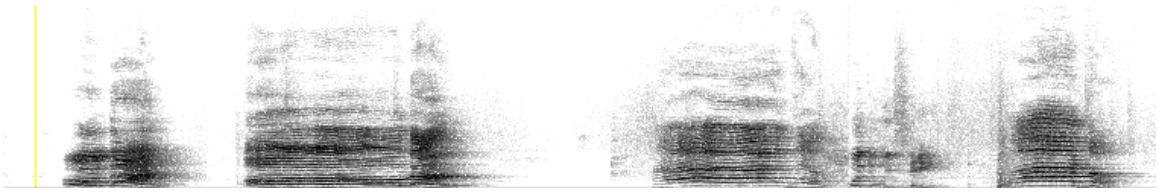
Transcribing speech is the essential task for annotating the HSP corpus, but how is it done? From a purely functional perspective, four basic tasks are involved:

1. FIND the speech in the audio stream.
2. MARK the speech boundaries, distinguishing the speech from the rest of the audio.
3. LISTEN to the audio for the speech segment.
4. TYPE the transcription for the speech segment.

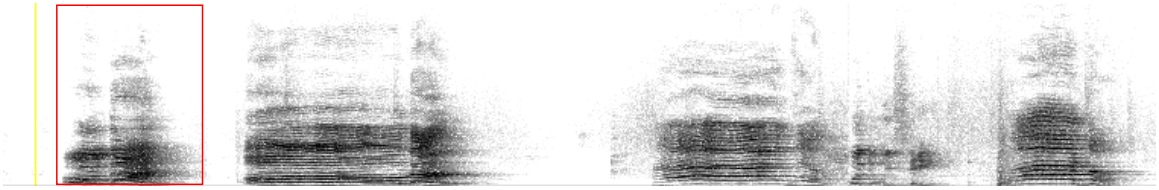
Audio can be represented visually using a *waveform*, which plots the signal directly, or with a *spectrogram*, which is an image showing the frequencies present in the audio over time. In a spectrogram, speech has a distinct visual structure due to the specific frequencies produced by a vibrating vocal tract, which a human annotator can learn to recognize. So a human transcriber can both listen to audio and visually inspect a spectrogram to find speech. Figures 2-1(a) and 2-1(b) shows the waveform and spectrogram of speech between a caregiver and child. Marking an utterance with a segment distinguishes it from the rest of the audio, as shown in figure 2-1(c). Next, the transcriber listens to the segment and types the transcript, shown in figure 2-1(d). Figure 2-1(e) shows the fully segmented and transcribed audio.

This is a preview sketch of a purely manual transcription process, and the next sections describe how these actions are performed using specific tools. Unfortunately, manual transcription is extremely time consuming, while purely automatic transcription using automatic speech recognition (ASR) technology produces too many transcription errors on the HSP corpus, largely due to the speaking style. Thus, a semi-automatic approach is developed,

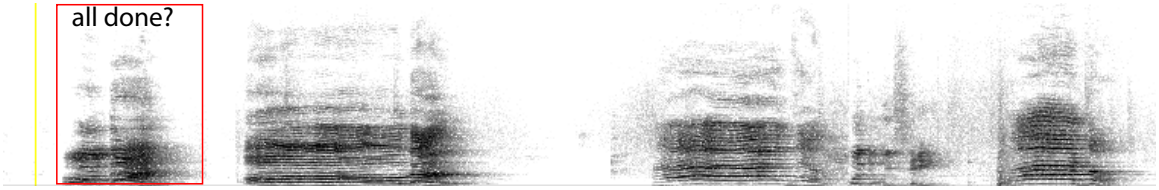
(a) Waveform of audio containing speech.



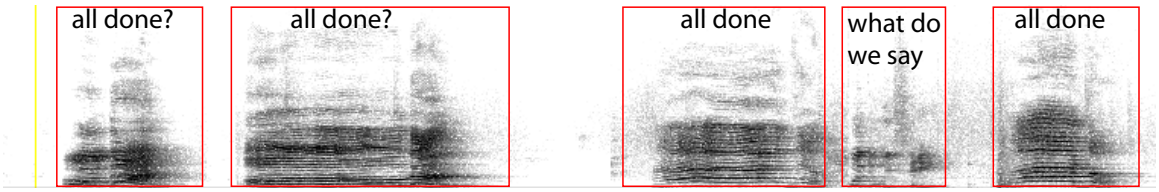
(b) Spectrogram of the same audio.



(c) Speech marked with a segment.



(d) A transcribed segment.



(e) The complete segmentation and transcription.

Figure 2-1: Waveform and spectrogram visualizations of audio, and the sequence of steps required to transcribe speech. This is an actual exchange between caregiver and child from the HSP corpus. The caregiver asks “All done? All done?” The child responds “All done!”, the caregiver asks the child again, and the child responds “All done!”

which automates the FIND and MARK stages, letting the transcriber focus on the LISTEN and TYPE tasks with specialized tools. Streamlining transcription in this way results in a significantly faster transcription system.

2.3 Manual Transcription

A number of manual tools for speech transcription are available, but two – CLAN and Transcriber – have been selected for consideration here. These tools were chosen based on their applicability to the task, and their popularity in the field. Procedurally, manual transcription tends to work the same way across different tools, following the FIND, MARK, LISTEN, TYPE (FMLT) paradigm. Other detail such as the speaker identity may also be annotated using most tools, which adds further complexity to the task.

Transcriber [3], provides a graphical interface which displays the current transcript and the audio waveform. Audio is played either continually, or by highlighting specific segments for playback. Segments are defined by creating markers in the audio stream, and each audio segment links to a line in a main transcription panel for entering the associated text. In addition, speaker turn taking can be entered in order to associate speaker identity with each transcript. Figure 2-2 shows the Transcriber user interface.

CLAN [30], is actually a suite of tools that were developed as part of the CHILDES research effort. This set of tools is one of the most widely used in the language acquisition research community. CLAN pre-dates Transcriber, and its user interface is less sophisticated in some ways. Like Transcriber, it provides a waveform display of the audio file and a text area for entering transcripts. One difference between CLAN and Transcriber is that the user does not see the entire segmentation of the audio file. Instead, the portion of audio corresponding to a given transcript may be highlighted if desired. This results in a segmentation that may contain overlapping segments, which may or may not be desirable. CLAN also makes more extensive use of codes embedded in the transcript to signify speaker identity and other information. Transcription roughly proceeds by playing through the audio, identifying and

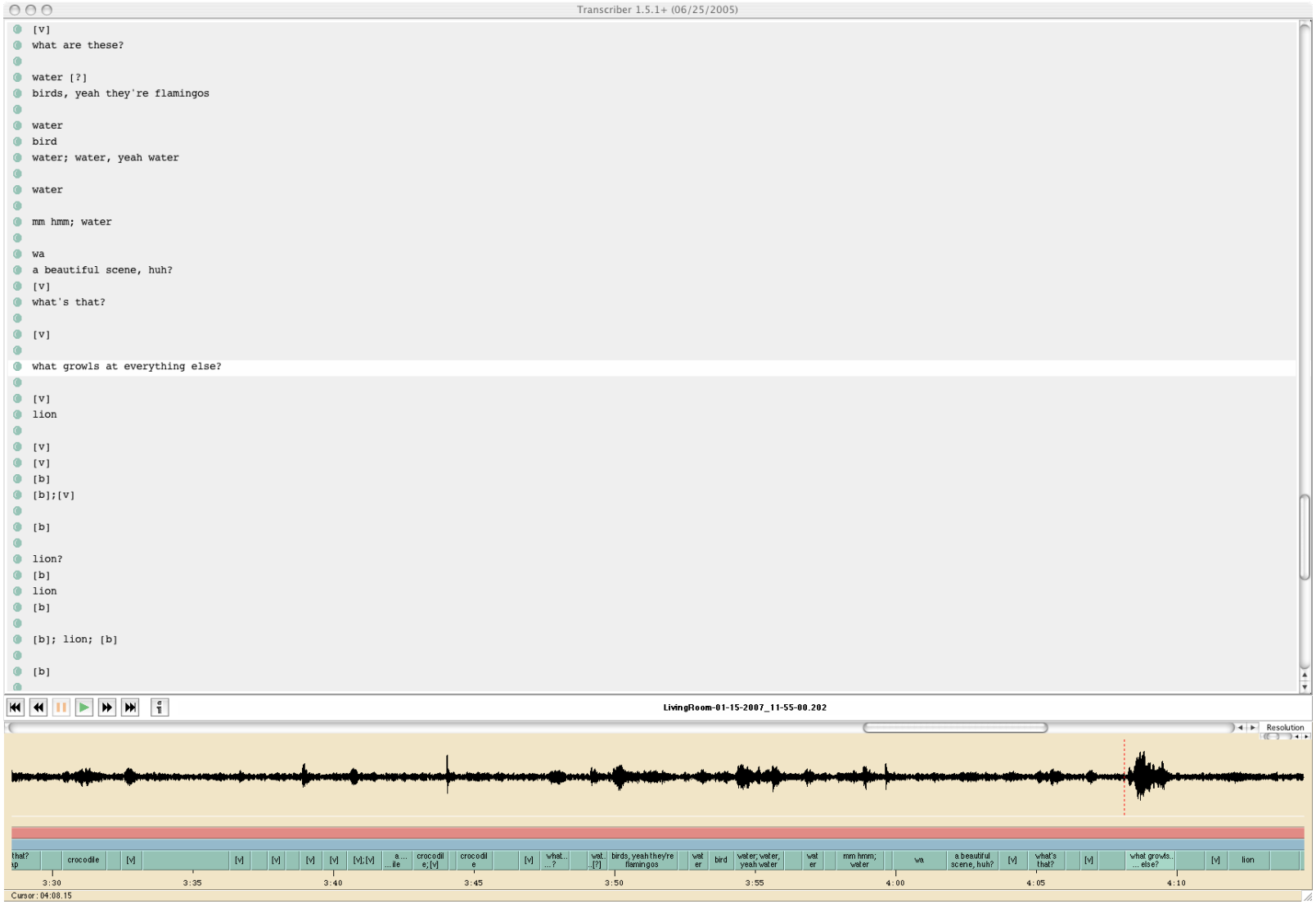


Figure 2-2: Transcriber screenshot. The top panel is the transcription panel, the bottom displays the waveform and the segments. Each segment shows a fragment of the transcript for that segment. Highlighting a line in the transcription panel highlights the corresponding segment.

transcribing a segment of speech, and associating the transcript to the segment of audio. Figure 2-3 shows the CLAN transcription interface.

2.4 Challenges of Manual Transcription

Manual transcription is a laborious process. To give a sense of how long it may take, in [3] the authors found that transcribing one hour of audio from various radio programs, newscasts, and televised news bulletins and documentaries took roughly fifty hours. They found this factor of fifty times real time to be consistent across transcribers, with variability depending on the program. A factor of ten to twenty times real time is reported in [53].

Why does manual transcription take so long? The FIND, MARK, LISTEN, TYPE sequence in CLAN and Transcriber requires physically moving one's hands between keyboard and mouse, and it also requires "cognitive switching" between modes of interacting with the system. The annotator starts with the "input" mode of visually scanning and listening to audio, decides on a segment, switches to "output" mode using the mouse to mark the segment, returns to the input mode of listening to and interpreting the speech, and then moves to the keyboard to output a transcription. Figure 2-4 illustrates the FMLT decomposition in terms of how the annotator interacts with the system.

It may be pointed out that, at least in Transcriber, the annotator could first segment all speech and then transcribe the segments, but in practice this did not occur. One explanation is that when marking speech, the annotator usually hears the speech and can start a transcription, needing to replay the segment fewer times.

The effect of cognitive switching, or more generally, cognitive load, introduced by user interfaces is discussed in depth in Oviatt [35]. In fact, Oviatt makes a very strong case for considering a user's unconscious cognitive abilities as a primary design principle for building user interfaces. Even apparently simple task-switching can create significant cognitive load for a user [33].

Transcription times that take a factor of ten to fifty times the actual audio time may be

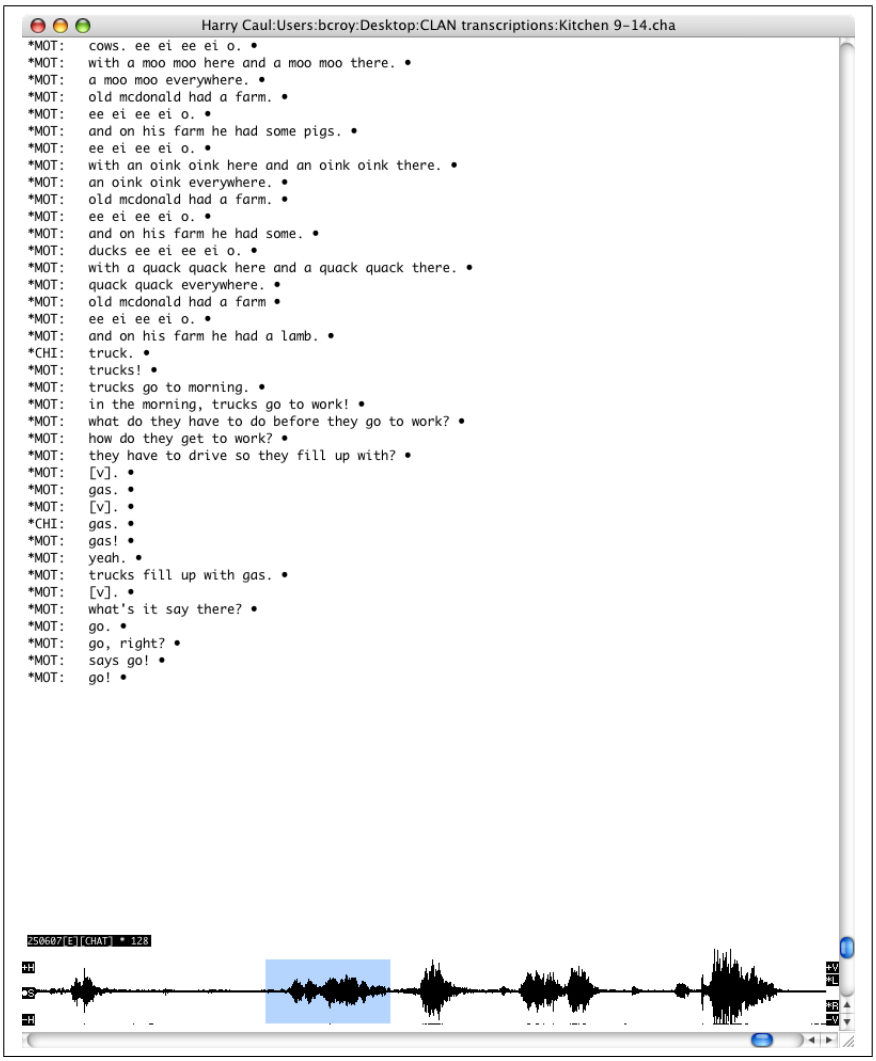


Figure 2-3: CLAN user interface. A segment in the waveform visualization at the bottom of the screen can be highlighted, and "bound" to a transcription in the top panel. Transcription follows a particular syntax, starting with a speaker code and then the transcription for the segment.

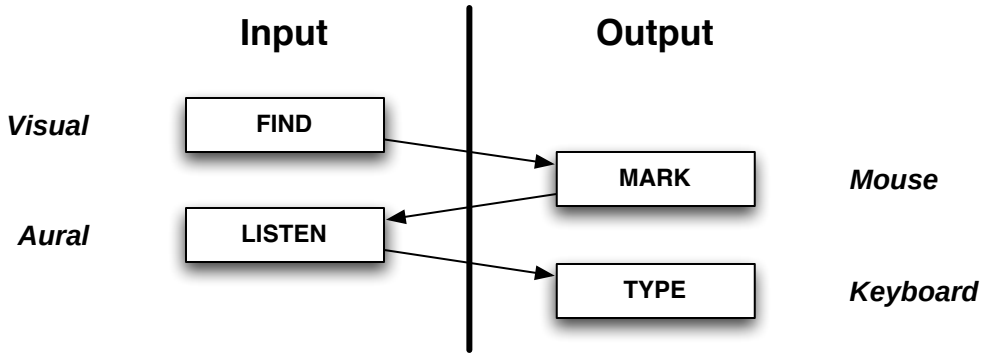


Figure 2-4: A functional decomposition of transcription, in terms of human input and output tasks and the modes of interaction in CLAN and Transcriber. The human transcriber must first find the speech by visually scanning a waveform. They then mark the speech segment using the mouse, listen to the segment, and type the transcription.

acceptable for a small corpus, consisting of tens or even hundreds of hours of audio. However, in HSP there are tens of *thousands* of hours of of audio. A typical day of recording gives rise to 14 simultaneous tracks, spanning roughly 10 hours, for over 140 hours of audio per day. Even listening to all this audio would take prohibitively long, let alone transcribing it. However, only about 2-5% of this audio contains speech, but even if all the speech could be isolated, at an optimistic factor of ten times real time it would still take a week of manual effort to transcribe a single day. A better solution is clearly needed.

Instead, if some of these functions could be automated, the transcription process could be streamlined. The approach taken here is to *eliminate* FIND and MARK by performing this with an automated system, and to build tools that are optimized for listening and typing. The next section introduces a semi-automated approach designed to streamline the transcription task.

2.5 Semi-Automatic Speech Transcription: The New Methodology

The system presented in this thesis is designed to maximize the speed and efficiency of speech transcription by combining human and machine capabilities in a harmonious way, based on a philosophy of human-machine collaboration [29]. The term "collaboration" is used here because the system leverages the complementary capabilities of both human and machine, and each helps the other perform their task better.

The approach automates as much of the FMLT task as possible, not only to reduce the total amount of work required for the human, but also to simplify the process and reduce cognitive load. Algorithms are used to FIND and MARK speech, allowing the human transcriber to focus only on the LISTEN and TYPE tasks. These last two stages still require manual transcription, largely because the speaking style in the HSP corpus is too challenging for automatic speech recognizers. The semi-automatic system presented in this section is at least 2.5 to 6 times faster than manual transcription alone.

2.5.1 Methodology Overview

The semi-automatic speech transcription system works as follows: Automatic channel selection and speech detection algorithms sift through the multi-track HSP audio corpus to find and mark speech segments. These speech segments are picked to be "easily transcribable", which means that they are in the loudest and clearest channel, and are the right length for human transcribers to store in short-term memory so they can quickly "listen and type." The speech segments found by the automatic system are then transcribed by a human annotator, who uses BlitZcribe (pronounced "blitz scribe"), a specialized tool designed for the listen and type process. BlitZcribe presents the detected speech segments in a list, and the human transcriber need only use the keyboard to play the audio and transcribe what they hear. They can also easily navigate the list and mark certain common errors using only the keyboard. There is no need to reach for the mouse, nor any need to examine

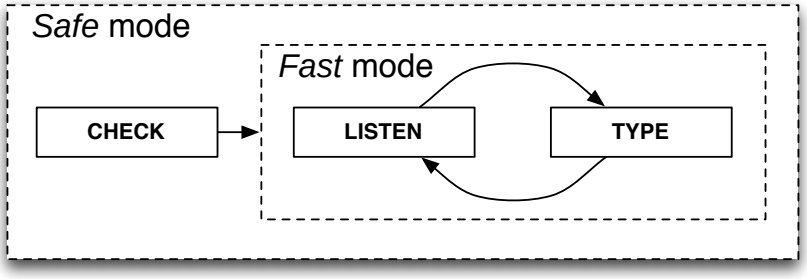


Figure 2-5: Semi-automatic transcription sequence, showing safe mode and fast mode. Safe mode begins by checking for missed speech, and then moves to fast transcription, which cycles between listening and typing.

spectrograms or search for speech. The simplified interface minimizes cognitive load, and focuses the human annotator's effort on exactly the steps of the FMLT process that cannot be performed automatically.

The introduction of the automatic component also introduces a source of potential errors. Missed speech and segments which do not actually contain speech are the two main error types. Fortunately, because the human annotator is an integral part of the system, human oversight can be added without creating significant additional work. Segments which do not contain speech are easily handled in BlitZcribe as part of the listen and type process. Missed speech requires a different tool to manually find and mark the segments. This tool, called TotalRecall, is a more general data browsing and annotation tool, and the user interacts with it in a manner similar to other manual transcription tools. Performing this step is optional and depends on whether the human transcriber feels satisfied with the automatic speech detector performance. Using TotalRecall to check for missed speech constitutes *safe* transcription mode, while skipping this step can be called *fast* transcription mode. In the FMLT paradigm, safe mode adds the CHECK step to the set of tasks. Figure 2-5 illustrates the steps for safe and fast transcription modes. Not only does human oversight monitor the automatic system performance, it also provides feedback the system can use to improve and adapt to changes in the acoustic environment. By explicitly considering all the human factors which arise in the semi-automatic approach, the automatic components and the user interfaces can be tuned to optimize the total system performance.

2.5.2 Terminology

The following sections refer to three basic data types used in the system: raw data, transform data, and metadata. *Raw data* is the original audio and video that is actually collected. *Transform data* is data that is a transformed representation of the raw data which may help a human annotator interpret the raw data. Spectrograms are a type of transform data. Both raw and transform data are stored on a filesystem. Finally, *metadata* are specific, semantically meaningful units of information about other data, usually raw data in this case. *Segments*, which are simply the combination of a channel and a time block, and *annotations*, which are labels for segments are the most common types of metadata in HSP. Metadata is stored in a central database.

2.5.3 Automatically Finding and Marking Speech

The first two steps of finding and marking speech in the audio stream are performed automatically. This process not only reduces the amount of audio that a human transcriber must listen to, it also simplifies the manual transcription task.

Finding speech segments is done using a statistical pattern recognition system that has been trained to distinguish between speech and non-speech. If a single stream of audio is input into the speech detection and segmentation system, a sequence of segments are returned which the detector believes contain speech.

However, since the audio data in HSP is actually recorded simultaneously from multiple overlapping channels, it is first necessary to identify the "best" channel. This must be done because audio channels overlap, in the sense that a sound occurring in one part of the house will register on multiple microphones. Choosing the loudest audio channel provides the best input signal not only to the speech detector, but also to the human transcriber who must eventually transcribe the segment. Raw, multichannel audio is sent to the *audio activity tracker*, which outputs a sequence of timeblocks and corresponding channels that represent the best channels over the course of a day. This can be thought of as a *virtual channel*,

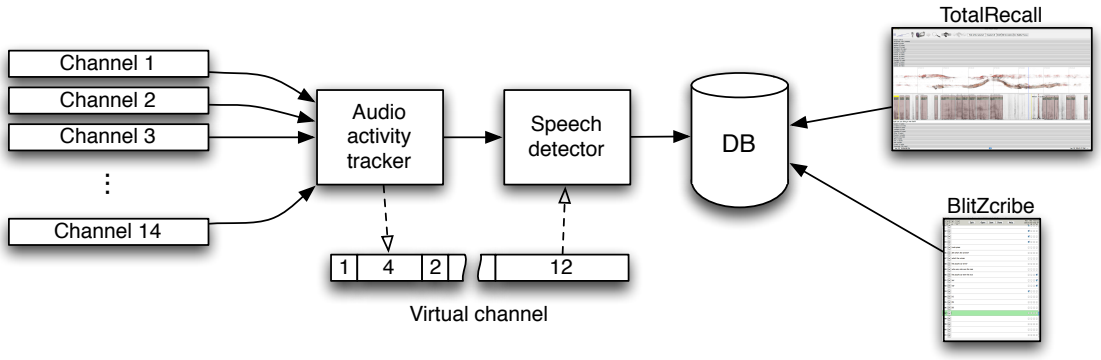


Figure 2-6: System diagram for the automatic speech detector. Multi-track HSP audio is processed by the audio activity tracker, which outputs a virtual channel of the loudest, clearest audio over time. The audio from the virtual channel is sent to the speech detector, which identifies the speech and produces segments that are "easily transcribable" by a human transcriber. The speech segments are stored in a central metadata database, which is used by BlitZcribe and TotalRecall.

which is like a channel that draws its source audio from various real channels over time. The speech detector then processes this virtual channel and output segments of speech. Figure 2-6 summarizes these automated steps. Given the rate of data collection in the Human Speechome Project, automatic speech detection runs on-site on a daily basis to keep up with the data flow, and status emails are automatically sent to administrators who monitor the system.

2.5.4 Tools

The metadata extraction described above is the first half of the system presented, but once speech segments are identified the human annotator must provide transcriptions for these segments. In addition, human oversight is needed to ensure the quality of the automatic processes. To accomplish both of these tasks, appropriate tools are necessary and are described in the following sections.

TotalRecall

The primary browsing and annotation tool is known as TotalRecall [28]. In addition to general browsing and annotation, it has been optimized specifically for transcription. At the simplest level it enables a user to view the data at multiple time scales. The interface presents a timeline view of all channels, and allows the user to zoom out to a time scale of years, or zoom in to a time scale of seconds. A playback head displays the user's current position in the data, and a selected audio or video channel can be played at that point. For video, a separate window shows all video channels playing simultaneously at a low resolution, and the selected channel playing at full resolution. In the case of audio, only the selected audio channel plays. The user can control the visibility of metadata, which is displayed in the appropriate channels and temporal locations in the interface. Transform data in each channel visualizes the corresponding raw data, which is essential for navigating and interpreting the data without necessarily playing it. Figures 2-7 and 2-8 show screenshots of TotalRecall and the video player, respectively. TotalRecall is the product of the efforts of multiple members of the Cognitive Machines group.

Data Visualization

In order to browse the data in TotalRecall without playing it back, visualizations are displayed. The user can select which channels to view – from a single channel, to a full view of all channels. Figure 2-7 shows four "open" channels – audio and video for both the kitchen and living room. The other horizontal bands are collapsed channels. All channels are synchronized in time, thus looking across channels for a given time can give a sense of what channels had activity, and often what sort of activity was taking place.

Audio visualizations are standard spectrogram images, which show the frequency content of the audio signal as a function of time. With a little practice, speech can be identified just by looking at the spectrogram. Figure 2-1(b) shows the spectrogram of speech between a caregiver and child. With spectrograms, a large amount of audio can be quickly scanned for potentially interesting segments. A custom spectrogram processor was written in C++

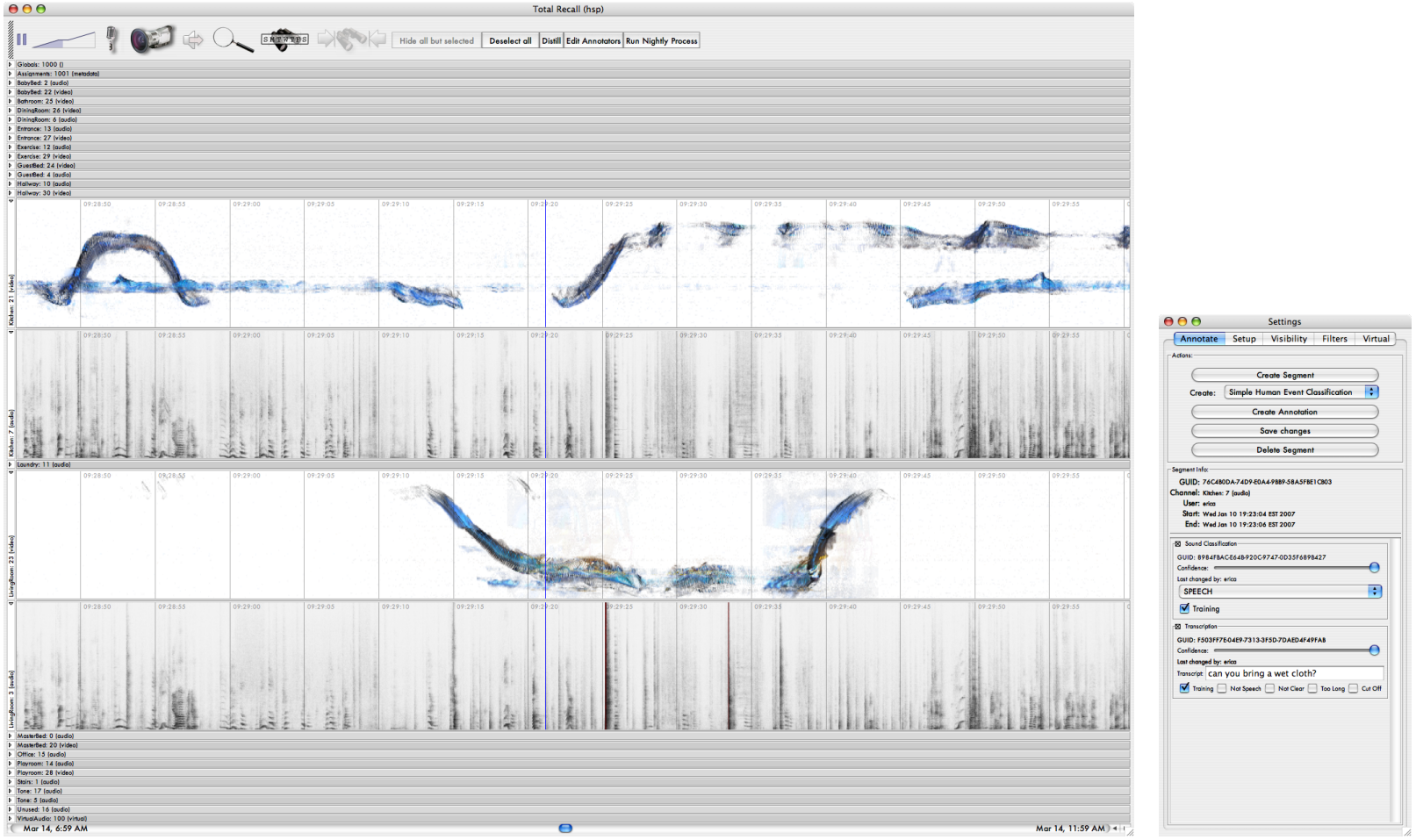


Figure 2-7: TotalRecall browser and settings windows. Each horizontal band represents a channel. Four channels are open in this screenshot - the top two channels are the video and audio for the kitchen, respectively, and the bottom two are the video and audio for the living room. The duration of time being displayed is about one minute. The video volume shows two people moving in the kitchen, with one briefly entering the living room. Both audio channels show speech activity, and the spectrograms look similar since the microphones in both rooms are picking up the same audio.

Figure 2-8: TotalRecall video player. On the left are low resolution videos from all cameras that are currently on. On the right is a full resolution (about 1000x1000 pixels) video from the kitchen, which is the currently selected channel.

using the FFTW library [19]. The primary requirements were to run as fast as possible, and to handle the custom HSP audio format. Figure 2-7 shows spectrograms as they appear in TotalRecall.

Video visualizations are *video volumes*, which roughly show the movement of people and objects in the video as static images. They are generated by calculating which pixels in individual video frames are in motion, and displaying these pixels, while making all non-moving pixels transparent. Successive frames are transformed in this way, and "slid" from left to right in a new image, with the amount of overlap between these transformed frames determined by the amount of time between each frame. Thus, a span of video produces an image with the height of a video frame, and the width proportional to the amount of time in the video clip. Figure 2-7 shows video volumes of two channels in TotalRecall. The video volumes are of two people in the kitchen, with one briefly entering the living room and then returning to the kitchen.

Both video volumes and spectrograms are types of transform data, described in section 2.5.2. They are simply channel specific images at various resolutions, representing one minute to one day of raw data for a channel. Using multiple resolutions is an optimization to help minimize the number of images that TotalRecall must load.

BlitZcribe

BlitZcribe is a tool specifically designed for rapid speech transcription in the semi-automatic framework. It is built with a simplified interface that focuses the transcriber's entire effort on listening to speech segments, and typing transcripts. The core interface element is a list of text boxes, with one for each segment. The user can play (or replay) a segment with a key press, type the transcript, and advance to the next segment using only the keyboard. Since the automatic segmenter is tuned to produce short segments that the human transcriber can easily remember, the need to replay segments is minimized. Common errors that do result from the speech detector can also be quickly marked with the keyboard as part of the normal workflow. By relying only on the keyboard, no extra time is needed to physically switch

between keyboard and mouse. In contrast to TotalRecall, which is a general browsing and annotation tool and is designed to present a comprehensive data view, BlitZcribe minimizes cognitive load by not presenting distracting or extraneous information. The key idea of BlitZcribe is to focus the transcriber's effort on the task of listening and typing short, easy to remember speech segments using a simple and streamlined mode of interaction.

In addition, because of the limited data access provided by BlitZcribe, it addresses some privacy concerns. Transcribers only hear segments of audio and cannot browse, which is important given the personal nature of the data and the need for transcribers who may not be closely connected to the project. In cases where a transcriber cannot completely understand what was said, some simple transcription conventions have been adopted. These segments can later be reviewed in TotalRecall by a privileged annotator with full access to the audio and video. A screenshot of BlitZcribe is shown in figure 2-9. The green text box (number 289) is the segment the transcriber is currently working on, and all previous segments have been annotated. BlitZcribe was written by Jethran Guinness, using components developed by various members of the Cognitive Machines group.

2.5.5 Process Summary

Now that the basic elements of the system have been introduced, the semi-automatic transcription process can be summarized. A detailed description is reserved for section 5.1. Automatic speech detection algorithms identify speech segments in the multi-track audio data, choosing segments to be an "easily transcribable" length and in the clearest audio channel. These speech segments are stored in a central metadata database, where they can be read by BlitZcribe. BlitZcribe is optimized for rapid transcription by minimizing the cognitive load on the transcriber. TotalRecall can be used for more general annotation or correcting errors and problem segments, since it allows all the audio and video to be played and displays transform data for efficient browsing. The human created annotations from BlitZcribe and TotalRecall are stored in the same metadata database, and an administrator can feed these human-created annotations back into the system to adapt and improve the

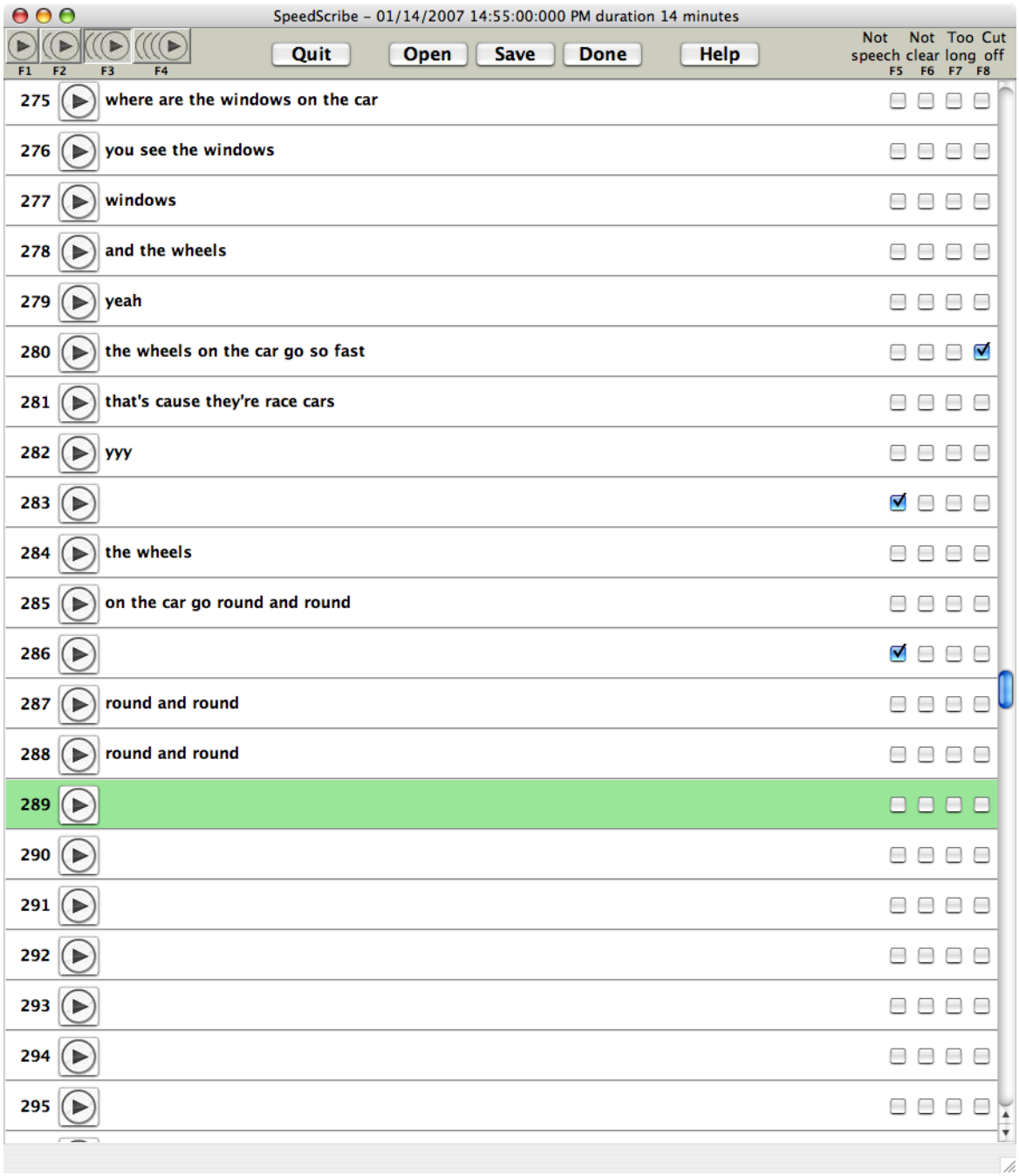


Figure 2-9: BlitZcribe. The green text box (number 289) is the currently active segment, which the transcriber is listening to. Transcribers can listen and type simultaneously, or mark common errors using only the keyboard. After finishing the segment, the transcriber hits return, and BlitZcribe will advance to the next segment.

automatic components. The total human-machine system is significantly faster – at least 2.5 to 6 times as fast – as purely manual approaches.

2.6 Fully Automatic Transcription

Although the previous sections have suggested that fully automatic transcription is not a viable option, it is an attractive idea and worth discussing in greater detail. Automatic speech recognition (ASR) systems process audio input and produce textual output. Though still an active area of research, such technology is finding its way into everyday applications, such as navigating telephone menu systems. For example, Gorin et al. present research for a spoken dialog system for call routing in [23]. Yet effective systems for unconstrained, conversational speech are still a long way off [15].

In an early pilot study, audio data collected for HSP was processed with a state of the art speech recognition system [11]. A large amount of manually transcribed speech was provided, along with the corresponding audio, to evaluate the recognition performance. One issue that came up was the need for a significant amount of transcribed audio in order to retrain the recognition system – a minimum of fifty hours was recommended. Since obtaining this amount of transcribed speech early in the project was a significant task in itself, a preliminary test applying the existing recognition system to ten hours of transcribed audio was performed. The audio data actually consisted of hundreds of short, extracted audio clips concatenated together, with an appropriate transcription file. Unfortunately, the initial results on this evaluation were quite poor, with a word error rate of about 95%. One reason for these results is simply that the speaking style is very informal, and significantly different from the type of speech that most ASR systems can handle. There is also background noise, variable acoustic conditions, and vocalizations from a child who at the time was not yet producing speech. ASR systems still have a long way to go before they can accurately recognize "speech in the wild."

2.7 Related Work

This thesis draws from prior work in speech transcription and annotation tool design, pattern recognition, and machine learning. Also of interest is work in language acquisition, user interface design and cognitive modeling of task performance. A selection of related work is presented here.

2.7.1 Annotation Tools

Transcriber and CLAN are the two annotation tools that are considered in depth, but other tools exist. CLAN and Transcriber are designed for speech transcription, but they also support other relevant annotations such as speaker identity. TransTool and SyncTool [34] are two complementary tools intended for multimodal spoken language corpora. Transcription is performed using TransTool, while SyncTool is used for aligning transcriptions with the corresponding audio (and video) recordings. The authors report creating these because of a lack of existing tools; those they could find were built for a specific purpose or standard, or were not available on multiple platforms.

The Annotation Graph Toolkit (AGTK) [31] is a toolkit for building applications which use the annotation graph formalism [4] for linguistic annotations of time series data. An annotation graph is a directed acyclic graph data structure capable of representing linguistic annotations, such as orthographic, phonetic, intonational, or other metadata.

Bird et al. describe a set of tools built upon the Annotation Graph Toolkit (AGTK) [5]. TableTrans is a spreadsheet based annotation tool, in which each row of the spreadsheet view corresponds to an annotation which is linked to a segment of audio, and each column a user definable annotation type. An extensible toolkit, AGTK uses third party software, such as WaveSurfer [50] for displaying audio waveforms, or Quicktime components for video. MultiTrans is another tool described in [5], designed for transcribing multi-channel audio signals. Its user interface is similar to Transcriber, but with a transcription panel for each audio channel. While the HSP audio is multi-channel, because the channels are overlapping

the multi-channel capabilities of MultiTrans do not provide any particular benefit.

WaveSurfer [50] and PRAAT [6] support powerful speech and signal analysis capabilities, and can also be used for transcription. Both tools can produce various visualizations of audio data, such as spectrograms, waveforms, pitch and power curves, as well as other transforms. However, the interface is not optimized for speech transcription, so would not be very useful for large-scale transcription.

MacVisSTA [44] is a system for multimodal analysis, which provides time synchronized access to multiple data sources, such as audio, video and other time-series data. For example, if sensors were used to track the hand position of a subject for gestural analysis, a hand position variable could be displayed with the corresponding audio and video. Along with audio, video, and time series data types, sequences of annotation metadata are supported. MacVisSTA also supports changing the temporal zoom level. By synchronizing annotations, audio and video, MacVisSTA is similar in spirit to TotalRecall.

A review and comparison of annotation tools is provided in [17]. Reidsma et al. discuss the challenges of annotating multimodal corpora in [42]. They summarize the annotation time factors for annotating various types of data, characterize the basic properties of different annotations tasks (for example, segmentation is a basic element of many annotation tasks), and discuss the implications for designing new annotation tools. See [43] for more on this work.

2.7.2 Audio Processing

The automatic speech detection algorithms draw from work in signal processing, pattern recognition, and machine learning. Many of these themes fall under the heading of "auditory scene analysis."

Auditory scene analysis refers to audio processing for extracting and interpreting information in the environment. Bregman gives a comprehensive treatment of this field in [7]. Computational auditory scene analysis (CASA) considers computational approaches to understanding the auditory environment, such as automatically separating speech from noise.

Bottom-up approaches to analysis, which integrate low level information and forward it to higher level processes corresponds to *primitive segregation* in Bregman's terms. The speech detector in this thesis uses a bottom-up approach. Top-down approaches, equivalent to Bregman's *schema-based segregation*, use information at higher level modules to bias lower level perception. Ellis [18] describes a system that uses a predictive model to interpret the auditory scene by resolving expected input with actual received input.

Speech detection may be viewed both as a recognition and segmentation problem. Segmenting an audio stream into classes may be performed without recognition, as in [55]. In this work, an audio stream is represented as a sequence of feature vectors, and the distance between successive features is computed. Changes in the audio signal, such as the start or end points of a speech event, should produce peaks in the distance function. Splitting the audio stream by choosing peaks results in a segmentation. Goodwin [22] takes a similar approach, but does use a training phase to find a distance function that reweights the features appropriately to focus on boundaries that distinguish acoustic classes. In addition, the peak picking procedure in [22] uses dynamic programming to find the best set of peaks to use as segment boundaries subject to a cost function. This work is interesting, but for this thesis we are only interested in detecting acoustic changes between speech and non-speech.

Computational auditory scene recognition focuses on recognizing the auditory context, or environment, rather than specific sounds [37]. Here, the authors are interested in recognizing environments like streets, restaurants, offices, family homes, and cars. The system in [8] recognizes auditory objects as part of recognizing the broader auditory context. Their interest is for mobile, wearable computing applications which are context-aware in order to minimize annoying or distracting interruptions. They point out the difficulty introduced by overlapping sounds, and the potential benefit of incorporating knowledge of human audition. Kapoor and Basu [27] explore auditory event and environment recognition using a representation they call an *audio epitome*, based on the idea of image epitomes developed in [26].

Speech segmentation in spontaneous speech is the subject of Yoshida [61], which used a statistical model of speech and silence based on frame level MFCCs, log frame energy,

and first and second derivatives. Segments were determined by smoothing frame label sequences with a four-state finite state machine that required a certain threshold of speech or non-speech in order to switch states. This work also considered the relative cost of false positives to false negatives, but did not consider the costs in terms of the effect on human performance. Yoshida also looked at higher level cues to improve speech utterance segmentation, an interesting adaptation that might also be explored in the speech segmenter in this thesis.

Other interesting work in this field tries to discover auditory objects from the data itself. Siegler et al. [49] used clustering to group utterances from broadcast news audio together, with the idea that clusters for individual speakers should form. Park and Glass [36] used an unsupervised clustering method on audio from academic lectures and were able to identify words and phrases. Smaragdis [51] used non-negative matrix factorization to identify acoustic objects in artificial as well as real data. In the real data, the acoustic objects were words, although in both [36] and [51], the acoustic environment is not nearly as complex as the audio collected in the Human Speechome Project.

Just as human annotators may review video in the HSP corpus to help resolve ambiguous words, machine algorithms may also benefit from multiple data modalities. Roy and Mukherjee present Fuse [45], which uses visual context to help speech recognition performance. Smaragdis and Casey [52] develop a method to determine meaningful audio and video features for events in a video stream, which they then use for segmentation. Making better use of the rich, multimodal data collected in the Human Speechome Project should not only improve speech detection and speaker identification, it should also enable extracting more complex and interesting meaning from the data.

Chapter 3

Automatic Speech Detection

The semi-automatic speech transcription system introduced in Chapter 2 tightly couples human effort with machine processing. The human works with the system using two primary tools – TotalRecall and BlitZcribe. With these tools, the results of the automatic processes are leveraged to enable the human to work faster and more efficiently. But it is important to recognize that although the automatic processes are intended to minimize human effort, they also require some maintenance. As a long term project, the overall health of the system relies on human oversight. In order to understand how the automatic component behaves, how to keep it running at its best, and how its performance characteristics affect the human task, the system must first be explained in more detail. This chapter focuses on the core elements of the automatic system.

3.1 Audio Activity Tracking

One reason collecting the HSP data is not overly burdensome to the participants is that it requires minimal attention to the recording process. While the participants have control of recording, it is designed to run continually and without intervention. As such, much of the recorded data doesn't contain any events of interest. In the case of audio, there may be stretches of silence or non-speech noise. When there is audio activity, only the microphone

closest to the sound source (ie. the loudest) need be processed, and the others can be ignored. The audio activity tracker does not distinguish between different sounds, its job is only to remove silence and track the best channel. Subsequent processing then focuses on speech, or potentially other sounds in the future. Functionally, the audio activity tracker processes the multichannel audio recordings and produces a sequence of segments containing audio activity. This sequence of segments may be thought of as a virtual channel.

3.1.1 Identifying the Loudest Channel

The audio activity tracker tracks the loudest channel over time, switching channels when there is sufficient evidence to support a channel switch. At equal intervals, a window is placed over the audio stream for a channel, and the power for that window is computed. For a window size of N samples, and audio samples x_i in the window, the power is defined as

$$p = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

Sliding the window results in a sequence of power values for each channel. This sequence is parameterized by the window size (N) and the window shift. The window shift is simply how many samples to shift the window for each power computation. In practice, a window size and window shift corresponding to 300 milliseconds (ms) has been chosen.

If the power p_j for a window at position j in the audio stream is below a minimum threshold α, the channel is determined to be silent. For the HSP audio, α has been chosen to be .00004. When all channels are silent, one can think of there being a currently active "silent channel". In this way it fits into the paradigm of choosing the "best" (or "active") channel.

3.1.2 Tracking the Best Channel

Simply choosing the loudest channel at each window results in a virtual channel that is fragmented. This can happen when two adjacent channels have roughly equal power due to a nearby sound source, and they toggle back and forth as the loudest channel, or a spurious

sound in another room occurs. Because of the high density of microphones in the house, multiple channels can often represent the source audio with similar fidelity. Therefore, it is not critical to always choose the loudest channel – other nearby channels can be used as well. Not only that, since segments are the basic element of annotations and are specific to a channel, a channel switch in the middle of an utterance will mean that the utterance cannot be represented by a single segment, which has implications for transcription later. So in the first case, it is better to simply pick one channel rather than switch back and forth. In the second case, a very short event in one channel should not cause a switch from a channel with sustained activity.

To reduce fragmentation, a smoothing step is performed to decide when to switch channels. This is accomplished using a dynamic programming [41] scheme that balances a cost for switching channels and a cost for choosing a channel that is not the current best channel. Following this smoothing, segments in the same channel separated by a short block of silence are merged. The dynamic programming algorithm used is the Viterbi algorithm, but it is used to minimize cost rather than maximize probability. However, the structure is identical. Algorithm 1 gives pseudocode for the process.

Notice that setting the channel switch cost to 0 essentially reduces to selecting the loudest channel at each window. Before the channel sequence is output, a pass over the channel segments merges any segments of the same channel which are separated by a short silence.

One property of this algorithm is that at each point in time, a single channel is designated to be active (if no channel is active, these gaps can be thought of as segments in an active "silence" channel). This means that if there are two distinct sound sources, only the loudest source will "win" and make its channel the active channel (subject to the smoothing conditions.) Experience with this process has encountered only two common problems that were easily solved without requiring a more sophisticated algorithm. The first was due to the sustained, loud noise in the laundry room from the washer and dryer, where speech almost never occurs. The laundry room channel was simply removed from the activity tracking. The second problem was due to the low-frequency rumbling of the garage door, or large trucks passing by outside. This was resolved by filtering out audio below about 200Hz, out

Algorithm 1 GlommiViterbi

N: number of sample windows of audio
K: number of channels
α: silence threshold

$\delta_0 \leftarrow \{0\}^K$ {Initialize the state sequence}
$\gamma_0 \leftarrow \{0\}^K$ {Initialize the cost vector}

for $i = 1$ to N **do** {For each sample window i}
  $o \leftarrow getActiveChannel(i, \alpha)$ {Get loudest, or silent, channel for window i}
  for each "to channel" $c \in$ {channels} **do**
    for each "from channel" $k \in$ {channels} **do**
      previous cost $\leftarrow \gamma_{i-1}(k)$
      transition cost $\leftarrow getTransitionCost(k, o, c)$
      costs[k] $\leftarrow$ previous cost + transition cost
    end for
    $\delta_i(c) \leftarrow \arg\min_j$ costs[j] {Store the minimum cost "from" channel to c}
    $\gamma_i(c) \leftarrow \min_j$ costs[j]
  end for
end for

$c \leftarrow \arg\min_j \gamma_N(j)$ {Get the minimum cost final state}
sequence $\leftarrow \{c\}$
for $t = N - 1$ to 1 **do** {Unroll the minimum cost sequence}
  $c \leftarrow \delta_t(c)$
  sequence $\leftarrow$ (c, sequence)
end for

segments $\leftarrow makeSegments$(sequence) {Convert to sequence of segments}
segments $\leftarrow removeSilentSegments$(segments) {Remove silence segments}
segments $\leftarrow glom$(segments) {Merge segments of same channel if close}

return segments

of the range of most sounds of interest.

However, to properly build a better audio activity detector, a method for determining whether the energy in a channel is due to the same sound source as a louder channel, or a different sound source should be applied. If the sound sources are different, then there should really be two virtual channels. The current algorithm assumes only a single sound source at a given time. The virtual channel that the audio activity tracker produces is stored as a file that is used by the speech detector.

3.2 Speech Detection

The speech detector is the component of the automatic process which processes an audio stream, and returns a set of speech segments. In practice, it takes the virtual channel produced by the audio activity tracker as an input. The speech detector is built from a combination of signal processing, pattern recognition, and smoothing algorithms.

3.2.1 Algorithm Overview

At a high level, the audio stream is first represented as a sequence of *feature vectors*. Feature vectors are computed from short windows, or frames, of audio. These feature vectors are then presented to a "frame level" classifier, which may output "silence", "speech", or "noise" for each frame. The sequence of frames is then smoothed and grouped into segments. The speech detector only outputs the speech segments, along with a confidence value for each segment indicating how sure it is that the segment is speech. Figure 3-1 shows the sequence of steps in the speech detection algorithm, and the following sections explain the algorithm in more detail.

3.2.2 Feature Extraction

The first step in speech detection is the feature extraction stage. A sequence of feature vectors are extracted from the audio stream, and summarize aspects useful for disambiguating

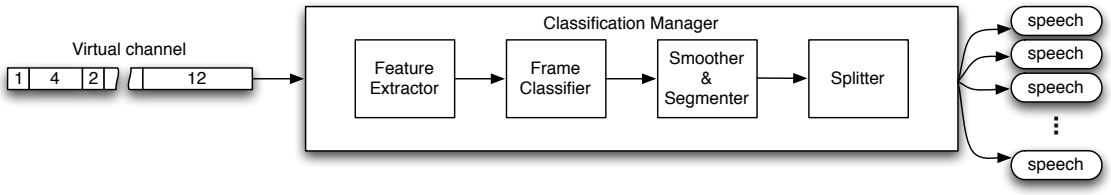


Figure 3-1: Speech detection block diagram. The speech detector processes a virtual channel, and outputs a set of speech segments. Internally, separate modules perform feature extraction, frame-level classification, smoothing and segmentation, and splitting of long segments to produce the final output.

sounds of different types. Each feature vector may be thought of as a point in the feature space. The choice of appropriate features is crucial for a successful pattern recognition application. A good feature representation groups examples of the same class together in the feature space, and separates examples of different classes. The pattern recognition algorithm uses these feature vectors to discriminate between examples of the different classes.

Since the rule for discriminating between classes by their feature vectors is usually unknown *a priori*, or is too complex to encode manually, it is the task of a machine learning training algorithm to find such a rule from a set of labeled examples. The training algorithm will be discussed in section 3.3.1.

Feature extraction begins by filtering and downsampling the audio from 48 KHz to 8 KHz. This is done both for speed purposes (by reducing the number of samples to process) and because it removes higher frequencies that are outside of the main speech frequency range. Using this narrower band produced better empirical results. After downsampling, the audio is partitioned into a sequence of 30 ms frames, shifted by 15 ms. For each frame a feature vector is extracted.

3.2.3 Feature Representation

The feature extraction stage processes the audio stream into a sequence of feature vectors, where each feature vector is composed of different feature types, described below. The features represent both temporal and frequency properties of the audio.

MFCC features

MFCC features are frequency domain features, which are transformations of the magnitudes of the frequencies present in a frame. Instead of considering the many raw frequencies present in the frame, a more compact representation is obtained by using knowledge of speech production and perception [13].

Each MFCC (or "Mel Frequency Cepstral Coefficient") vector is a function of a single Fourier transform output vector. MFCCs are computed by first producing a small set of weighted averages of different frequencies. This is the Mel filtering stage. A triangle function, which is defined by a center frequency and a bandwidth, is placed over the FFT vector at the center frequency, and the average of the magnitudes of the contained frequencies is computed, weighted by the triangle function value at each frequency. In other words, for a triangle function $T_{c,w}[j]$ with center frequency c and width w, and frequency vector $f[k]$, the weighted average

$$w = \sum_{k=0}^{N} T_{c,w}[k] f[k]$$

is computed. M triangle functions are chosen with centers $\{c_i\}$ and widths $\{w_i\}$ based on perceptual information. The discrete cosine transform of the Mel-filtered spectrum is then taken, resulting in an MFCC vector.

Give a sequence of sample frames, the MFCC extraction procedure produces a sequence of MFCC vectors. MFCCs are computed using customized code based on the CMU Sphinx package [57]. For more information on MFCCs, see [13].

Other features

In addition to MFCCs, other features are used to represent the audio stream.

- Zero crossings

  The zero crossing feature is the count of how many times the signal crosses the zero

point. This is the number of times the signal amplitude transitions from a positive value to a negative value, or vice versa, in the frame.

- Max amplitude

  The max amplitude feature is the maximum observed amplitude in the frame.

- Power

  The power feature is the total power of the signal in the frame. Two power features are computed – one in the short 30 ms frame, and a second over a wider 330 ms window.

- Spectrum entropy

  The spectrum entropy calculates the entropy of the frame's frequency magnitude vector, if it is viewed as a probability distribution. Entropy is a quantity in information theory that roughly measures how "peaked" a probability distribution is [12]. In this case, the idea is that common noise in the HSP audio, which has roughly equal power in all frequency bands (approximately uniform) has a higher spectrum entropy than speech, which has peaks at specific frequencies.

- Relative power

  The relative power is the ratio of the signal power in one frequency band against that of another frequency band. In speech, most of the power is in the frequency band 100 Hz to 4000 Hz. Thus the ratio of power in this band against the entire frequency band (0 - 24000 Hz, for the raw HSP audio) will be close to 1. However, in the case of white noise, there is power in the entire frequency range, so the amount of power in the range 100 Hz to 4000 Hz will be only small fraction of the total power in the signal.

3.2.4 Frame Classifier

The result of feature extraction is a sequence of feature vectors, which are then used for classification. While the goal of the the speech detection algorithm is to produce speech

segments, internally each feature vector is first classified by a frame-level classifier. This classifier is built using the Weka machine learning library [60], and is written in Java.

The classifier used is a boosted decision tree. Decision trees [39] are classifiers that perform a sequence of decisions on an input feature vector to obtain a classification. The classifier can be viewed as a tree, where at each node the feature vector is evaluated based on a criterion specific to that node. The outcome of the evaluation determines which child node should next evaluate the feature vector. Each leaf node in the tree corresponds to a classification – when a leaf node is reached the classification is returned.

The performance of decision trees can be improved through a technique known as *boosting* [47] [40] [16]. Boosting is a method for generating an improved classifier from an ensemble of classifiers. Roughly, boosting works by building a classifier, re-weighting the training data to emphasize misclassified training examples, building a new classifier, and so on, to produce an ensemble of classifiers. The re-weighting step is intended to produce a classifier that performs better on the misclassified examples. Classification is performed by combining the outputs of the component classifiers in the ensemble.

In practice, this is done using the MultiBoostAB algorithm in the Weka library [59]. The extracted features are passed to the frame level classifier, which applies the boosted decision tree classifier to the sequence of feature vectors, and returns a sequence of frame classifications and associated confidence values.

3.2.5 Speech Segmentation

The frame classifications produced by the frame level classifier are only an intermediate result. The final speech segments are obtained by grouping these frame classifications into actual speech segments. A simple approach to obtaining speech segments is to identify contiguous blocks of frames that have the same classification, and return each block as a segment. However, as with audio activity tracking, this would produce too many short segments, due to spurious classifications. A desirable segment, from a transcriber's perspective,

is one which is long enough to contain a word, but short enough that the transcriber can remember what was said.

While the frame level classifier has a myopic view of the audio stream, and treats each frame independently, the transcriber's notion of a speech segment is at a broader timescale. Even if an individual frame appears to be speech, if the surrounding frames aren't speech then it is likely a spurious classification. Similarly, speech segments often contain speech sounds which are indistinguishable from noise, or silences when there are pauses. Thus, the relationship between adjacent frames must be considered in order to identify good speech segments. One way to view the problem is that there is a true underlying transition point in the audio stream at the boundary of speech and non-speech, and the task of the segmenter is to identify these transitions from the observed frame classifications.

This is accomplished in the system using a smoother similar to that developed for the audio activity tracker. The smoother takes a sequence of frame classifications as an input, and returns a smoothed sequence of frame classifications, where some frames have been relabeled by the smoother. The smoothing algorithm assumes that there is a true hidden state, which may either be speech or non-speech. There is a cost for switching states, and a separate cost for being in a state which differs from the frame classification. A dynamic programming algorithm [41] then runs to find the minimum cost state sequence, subject to the costs and the frame classification sequence. This state sequence represents a segmentation of the audio stream.

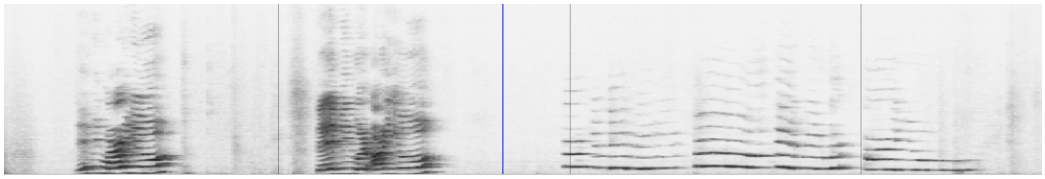
However, smoothing may still produce segments which are too short or too long. Segments which are shorter than a minimum length threshold α are discarded. Segments which are longer than a maximum length threshold of β are split into multiple shorter segments, with duration between α and β. In practice, $\alpha = 350$ ms, and $\beta = 5000$ ms. This choice of β has been made so that the human transcriber will never need to transcribe a speech segment that is too long to store in short-term memory.

For long segments, finding the optimal split points proceeds by looking for local minima in the *confidence function*, and looking for silence points. The confidence function is a

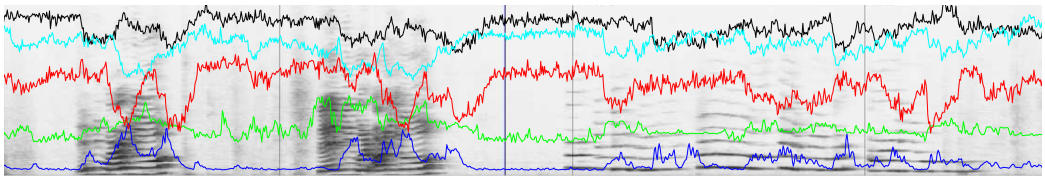
function over the confidence values for the frame classifications. The confidence values are first smoothed using an averaging scheme. If there are regions in the frame classification sequence where the classifier was less confident of the speech label (for example, the frame classifier was less sure or the smoothing actually relabeled the frame from non-speech to speech), then the minimum point is a good split candidate. In addition, a point of minimum energy may also be a good split point, since it could correspond to a silence between words. Functionally, the splitter takes a single speech segment as an input, and returns a set of speech segments within the target duration range. Each speech segment has an associated confidence value, which is the average of all the frame level confidence values in the segment.

3.2.6 Summary

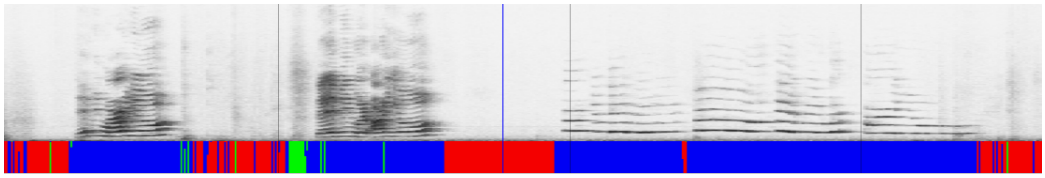
The final result of feature extraction, frame classification, smoothing, segmenting and splitting is a set of speech segments, which are stored as metadata in a central database. These will be the segments presented to a human annotator for transcription in BlitZcribe. An example will help make the speech detection algorithm more concrete. Figure 3-2 shows a visualization of the speech detection processing stages. Figure 3-2(a) shows a spectrogram of audio. Features are extracted, several of which are shown in figure 3-2(b) (the features have been scaled and positioned over the spectrogram for clarity.) The set of features are processed by the frame-level classifier, which produces a classification for each frame, visualized in 3-2(c). Each frame classification should be read as a column from bottom to top: the output classification color starts at the bottom and goes up to a height proportional to its confidence, then the next most confident label color is displayed, and so on. The next picture shows the result of smoothing the frame classifications. Since the audio was fairly clean, the starting frames were actually well segmented to begin with, but the output of the smoother defines the segments. However, the rightmost block of speech is longer than the maximum allowable segment length, so the splitter has broken it into three segments, attempting to find the best split points.



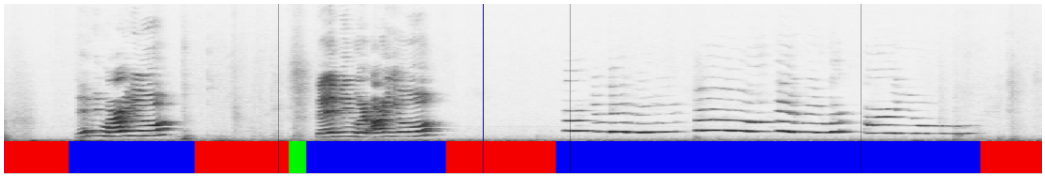
(a) Audio spectrogram



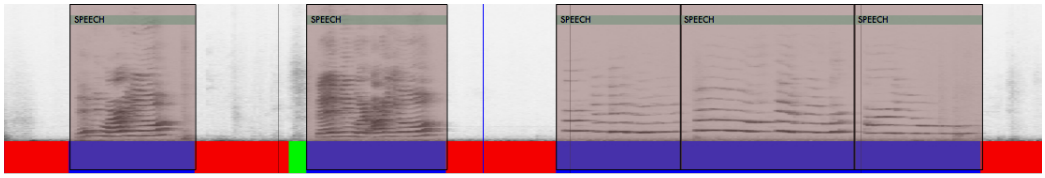
(b) Spectrogram with selected features plotted. From bottom to top: short window energy, zero crossings, mfcc_2, mfcc_1, mfcc_3



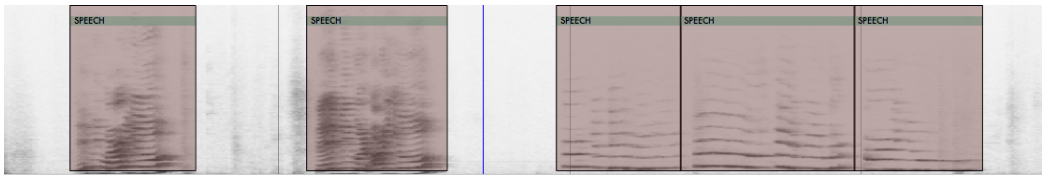
(c) Spectrogram with frame-level classifications displayed



(d) Smoothed frame-level classifications. Note the relabeled frames



(e) Spectrogram with segments. Long speech block has been split into three segments



(f) Final speech segments

Figure 3-2: Processing sequence in speech detection. The colored band in figures 3-2(c) and 3-2(d) are visualizations of the individual frame classifications. Frames labeled as speech are blue, silence is red, and noise is green. After smoothing, segments are defined by the boundaries between different acoustic classes. However, the speech block on the right is too long, and is split by the splitter algorithm into three segments.

3.3 Building the Speech Detector

The speech detection algorithm has a boosted decision tree frame-level classifier at its core. This section describes how the frame classifier is built, and how other speech detector parameters are set.

3.3.1 Training the Frame Classifier

The training algorithm starts with a hand labeled *training set*, which is a set of examples drawn from the different classes to be differentiated. In this case, speech, silence, and noise segments are hand labeled. Rather than grouping silence and noise together into a "non-speech" class, they are kept separate, in part with an eye toward future development.

Training proceeds as follows. First, all examples of speech, silence and noise are collected from the database, and feature extraction is performed. The same feature extractor used for classification is used for training. For each training segment, all the extracted feature vectors for that segment are labeled with the segment label. These labeled feature vectors are then input to the Weka boosted decision tree training algorithm. For details on training, see [39] and [59]. The result of training is the core Weka classifier model, which is saved as a file, and a frame-level performance summary.

3.3.2 Frame Classifier Performance

The frame classifier which results from the training procedure may be characterized in terms of its error rate, and the types of errors that it makes. Since the label of each frame is known, a set of training data can be held out from the training process, and used for testing. The training data is actually partitioned into five sets, and five-way crossvalidation performed. As part of the training procedure, it is useful to output performance characteristics of the model along with the model. Before the training procedure returns, the new model is used to classify each frame of the test set. A classification error occurs whenever the classifier

label disagrees with the human label for a particular frame. A simple measure of error is just the total number of frame errors relative to the total number of frames. However, a more detailed summary of the classifier performance can be obtained using a *confusion matrix*, and calculating the *precision* and *recall* for each class.

Measures of Classifier Performance

Given k classes, a $k \times k$ matrix can be populated which contains all possible error types. Each row corresponds to an actual, human label, and each column corresponds to a "guessed", classifier label. Entry i, j in the confusion matrix corresponds to the number of times an example labeled by the human as i was classified by the classifier as j. Any entries on the diagonal correspond to correct classifications, and any off-diagonal entries are errors. If the classifier is allowed to return "no classification", an additional column can be added to the confusion matrix.

A confusion matrix immediately shows whether two classes are being confused with one another, if there are large counts in a particular off-diagonal entry. *Recall* represents the fraction of examples with a particular human label that are also correctly guessed by the classifier. *Precision* represents the fraction of examples that the classifier labels with a particular label which are actually correct. More compactly, given a confusion matrix $C_{i,j}$ and a label l,

$$\text{recall}_l = \frac{C_{l,l}}{\sum_j C_{l,j}}$$

$$\text{precision}_l = \frac{C_{l,l}}{\sum_i C_{i,l}}$$

While one would prefer a classifier with perfect precision and recall, in practice this rarely occurs, and in fact there is often an inherent tradeoff between the two. For example, consider a classifier that labels *all* examples with a particular label. It will have perfect recall for that class, but poor precision. On the other hand, if the classifier classifies only a single example with a particular label and is correct, it will have perfect precision, but poor recall.

The result of classifier training is a classifier model and the frame-level performance characteristics. While the frame-level performance obviously affects the final speech detector, it does not characterize the complete system performance as observed by a human annotator. System level performance issues are briefly described in section 3.3.4 and will be further discussed in Chapter 6.

3.3.3 Improving the Frame Classifier

The various components of the speech detector all contribute to its final performance. The feature extractor must provide useful features for discriminating between sounds of different types. The addition of a good feature can significantly improve performance. The frame level classifier must be built using a good core classifier algorithm, but also depends critically on the training set. Finally, the smoothing and segmentation parameters must be selected to produce high quality speech segments.

In this architecture, if the performance of the frame-level classifier is poor, the resulting segmentations will also suffer. Assuming the architecture is fixed, one way of improving the classifier is to continually improve the training set. If the confusion matrix reveals a frequent confusion between two classes, simply adding more training data can help. Classification errors, when corrected by a human, can be added to the training set to help prevent such errors in the future. In addition, alternatives to boosted decision trees can be used simply by changing a configuration file. In building the system, experimenting with feature extraction has had significant impact, but these experiments are not the focus of this work and are omitted here.

3.3.4 Speech Detector Performance Tradeoffs

The end product of automatic speech detection is a set of segments which the transcriber works with. Measuring the speech detector performance is important since it is directly relevant to the human transcriber.

False Positives and False Negatives

As with the frame classifier, precision and recall are useful for characterizing the speech detector performance, as are other measures. Consider a particular class, such as "speech". As a binary labeling problem, all speech segments are positive examples with respect to the label "speech", while all non-speech segments are negative examples. If a speech segment is missed by the system, it is falsely labeled as negative – in other words, it is a *false negative*. Likewise, a speech segment which is found that is not actually speech is a *false positive*. *True positives* and *true negatives* are segments which have been correctly labeled (ie. skipped, for negatives). A confusion matrix can be collapsed into a table of such counts for a given class.

False positive and false negative errors are subject to the same tradeoff as precision and recall, mentioned above. Internally, many classifiers use a confidence threshold to decide when to output a particular classification. Varying this threshold will affect the false positive and false negative rates. A lower threshold for outputting a speech classification would result in a more "aggressive" speech detector, which outputs more speech but also produces more false positives, while a more conservative classifier will miss more speech but produce fewer false positives. Other parameters can also affect how aggressive or conservative a classifier is.

Tuning the Speech Detector

For the speech detector, the smoother and segmenter determines how the sequence of frame classifications is mapped to speech segments. Adjusting its parameters tunes the final speech detector performance. These can be adjusted using a parameter search process. The primary smoother parameters are the cost of switching states, and the cost of being in a state which disagrees with the current frame classification. Finding a good setting for these parameters is accomplished by a grid search over the parameter space, and picking the parameters that produce the best resulting segmentation relative to a human specified segmentation.

The best segmenter for semi-automatic transcription is not simply the one with the minimum total error. This is because each error by the speech detector directly affects the workload for the human annotator, and different errors may require different amounts of effort. The best speech detector is the one which minimizes human effort. In a very concrete sense, there is a cost for both false positives and false negatives – that is the monetary cost for additional transcriber time.

By understanding the relationship between the automatic system performance and the human transcriber performance, the speech detector can be optimized in a principled way. The next chapter discusses evaluations of the speech detector, and chapter 6 gives an in-depth analysis of the connection between the human and machine performance.

Chapter 4

Evaluating the Speech Detector

This chapter evaluates the speech detector as a standalone system. Several types of error may result from the speech detector, but the two primary errors discussed are false positives and false negatives. The automatic system is complex, consisting of multiple modules as illustrated in figure 3-1, and errors can propagate. These modules are further investigated here.

4.1 Performance Metrics

Functionally, the task of the speech detector is to break an audio segment into easily transcribable chunks of speech. The main requirements are that segments labeled as speech contain speech, and segments labeled as non-speech do not contain speech. In addition, segments containing speech should be a reasonable length – they should not be so long that a transcriber will have difficulty remembering what was said. It is also desirable that a speech segment does not clip the speech at the ends, or contain excessive amounts of non-speech.

These last two aspects – clipping and excessive non-speech within a segment – may be considered secondary. Such speech segments are still in need of transcription, but the

transcriber may spend more time than necessary listening to the audio, or have trouble distinguishing words at the boundaries if there is clipping. Ideally the speech segments are all tightly end-pointed at word boundaries. Keeping the segments short enough is a matter of breaking up long speech segments at appropriate points. The issue of false positives and false negatives, however, still holds primary importance.

4.1.1 Comparing Segmentations

Evaluating the output of the automatic processes relies on appropriate similarity and error metrics. While the quality of the output is ultimately reflected in the human task performance times, this is an indirect measurement, and obtaining these measurements is a time consuming process. Instead, a mechanism for evaluating segmentations directly would be useful. This approach relies on comparing the machine generated segmentation against a human generated baseline. However, unlike some other labeling tasks, there may not be any "true" segmentation – two human annotators may produce different segmentations of the same audio stream, and yet both be completely happy with either segmentation. Still, two human segmentations should be quite similar, since both should at least agree which portions of the audio are speech or non-speech.

Definitions

Before proceeding, it is useful to define some of the relevant concepts more precisely. A *segmentation* is a set S of segments (t_s, t_e, l), where t_s is the start time of the segment, t_e is the end time, and l is the label. For simplicity, consider only segmentations where for any pair of segments a, b, a does not intersect b. Two segments intersect if they share a common block of time. Furthermore, consider only contiguous segmentations in which the union of all segments covers the entire block of audio. The result of these simplifications is that for any time t in the audio stream, there is exactly one segment containing t, (and thus one label for each point in time). This constrained form of segmentation is a function $f : T \rightarrow L$ for all t in the time range T of the audio, and l in the set of labels L. The *speech*

density of a segmentation is a very useful quantity, which can be defined as the total amount of time in speech segments relative to the total audio time. Speech density characterizes how much speech is in an audio stream, and is a good predictor of transcription task time. Alternatively, speech density can be defined as the number of speech segments per unit of audio time, which will also be used in later sections.

4.1.2 Similarity

Perhaps the simplest way of comparing two different segmentations of a single audio block is to accumulate the amount of time the segmentations are in agreement. Using the simplified notion of a segmentation described above, a simple similarity function can be computed by scanning the audio from beginning to end, and at each point t evaluating the label l for both segmentations. The cumulative amount of time both labels are in agreement relative to the total amount of time represents a similarity between two different segmentations. More compactly, if $L = \{-1, 1\}$, let the *overlap* be

$$\text{overlap}(S_a, S_b) = \sum_{t=1}^{N} \frac{f_a[t] f_b[t] + 1}{2}$$

This simply counts the number of agreements between S_a and S_b. For an audio stream of length N, the similarity is then

$$\text{sim}(S_a, S_b) = \frac{\text{overlap}(S_a, S_b)}{N}$$

This has the nice property that $\text{sim}(S_a, S_a) = 1$ and the similarity of a segmentation S and its complement $\overline{S}$ is $\text{sim}(S, \overline{S}) = 0$.

This simple similarity function could also be restated using a different definition of segmentation. If the constraint of contiguity is relaxed, and all intervening non-speech segments are removed, the overlap is the cumulative amount of time in the intersection between segments in S_a and S_b.

4.1.3 Error

If one segmentation is assumed to be the "true" segmentation, then an error can be computed for other segmentations relative to the provided true segmentation. Assuming the simplified segmentation described above, false positive and false negative rates can be computed, as well as the related measures of precision and recall.

Let A be the "actual" segmentation, and G be the "guessed" segmentation. Then for the corresponding segmentation functions, for each time t in the audio stream we have $a[t] \rightarrow l_a$ and $g[t] \rightarrow l_g$. For our purposes, each l is either "speech" or "non-speech". Considering speech as the "positive" class, then the following table results:

	l_g = speech	l_g = non-speech
l_a = speech	True positive	False negative
l_a = non-speech	False positive	True negative

Table 4.1: False positives and false negatives

In the similarity metric, the sum was over all agreements relative to the total amount of time. In this case, the four types of agreements and disagreements are binned appropriately and accumulated. For each time index t, $a[t]$ and $g[t]$ are computed, and the appropriate cell of the table is incremented.

Sections 3.3.2 and 3.3.4 discuss false positives, false negatives, as well as precision and recall. Precision and recall were previously defined in terms of a confusion matrix. Precision and recall can also be computed from this table as follows:

$$\text{Precision} = \frac{tp}{tp + fp}$$

$$\text{Recall} = \frac{tp}{tp + fn}$$

Precision measures the fraction of the classifier's guesses that are actually correct, while recall measures the fraction of the actuals which are correctly guessed. The true positive

and false positive rates can also be computed. They are defined as

$$TPr = \frac{tp}{tp + fn}$$

$$FPr = \frac{fp}{fp + tn}$$

Another useful quantity to remember is the false negative rate, which is $FNr = 1 - TPr$. From the pool of all negative examples (non-speech, in our case), the fraction of times the classifier incorrectly labeled such examples as positive (ie., speech) determines the false positive rate. A hypothetical classifier which labels everything as speech will have an FPr of 1, since all non-speech is labeled as speech. On the other hand, the true positive rate looks at the fraction of positive examples that are actually correctly labeled as positive by the classifier. In fact, the TPr is the same as the recall.

4.1.4 ROC curves

Many classifier algorithms produce, for any particular input, a "degree of belief" or *confidence* over possible output classifications. For a particular label, the classifier can be thought of as a binary classifier, where all classifications with a particular label are positives and all others are negatives. The choice of whether or not to output the label usually comes down to thresholding the confidence. By varying this threshold, the classifier will be more or less likely to produce that label as an output, and thus varying this threshold impacts the TPr and FPr. A curve which relates TPr and FPr for different threshold values is known as an *ROC curve*. The ROC curve effectively summarizes the quality of the whole family of classifiers which are parameterized by the threshold.

ROC curves typically depict the false positive rate on the x-axis and the true positive rate on the y-axis. An informal way to look at the process of generating an ROC curve is as follows. All examples are sorted according to the classifier's confidence that they have a particular label. A confidence threshold divides this list into two halves – everything in one half is labeled as a negative, everything in the other half is labeled as a positive. The

group of positives consists of true positives and false positives, so as the number of false positives increases, the number of true positives cannot decrease, and vice versa. Since the denominators of both the TPr and the FPr are constants, then the resulting ROC curve will be monotonically increasing. Generally, a classifier with a steeper curve is a better classifier. Figure 4-1 shows the ROC curve for the frame classifier.

4.2 Frame Classifier Performance

The frame classifier is the first module in figure 3-1 where errors can occur relative to a human specified ground truth. Errors at this stage will degrade the performance of the segmenter, so reliable frame classifier error metrics are needed to monitor frame classifier performance. Section 3.3.2 describes how the training procedure determines frame errors, using five-way crossvalidation. Precision, recall, and FPr are used to summarize the performance of the frame classifier. In addition, an ROC curve shows how the classifier performs for a particular class as a function of a varying confidence threshold.

The performance of the frame classifier used in this thesis are as follows:

Precision	Recall	FPr
.890	0.943	0.060

Table 4.2: Frame Classifier performance

The ROC curve in figure 4-1 also summarizes the performance for the "speech" class, assuming a confidence threshold is applied to the frame classifications. The four different curves represent four different training runs, in which various aspects of the training process had been varied. The best frame classifier, represented by the steepest, solid blue ROC curve, is the one used for all evaluations. One point about the frame classifier is that since it uses a decision tree as its core classifier, it does not necessarily provide good class membership probabilities, or confidences. Therefore, there is not a broad distribution of confidence values, and as the threshold is varied to produce the ROC curve, most of the actual observed (FPr, TPr) pairs are in the top left of the plot. Techniques for modifying decision trees to produce better probability estimates are explored in [38], [1] and [54].

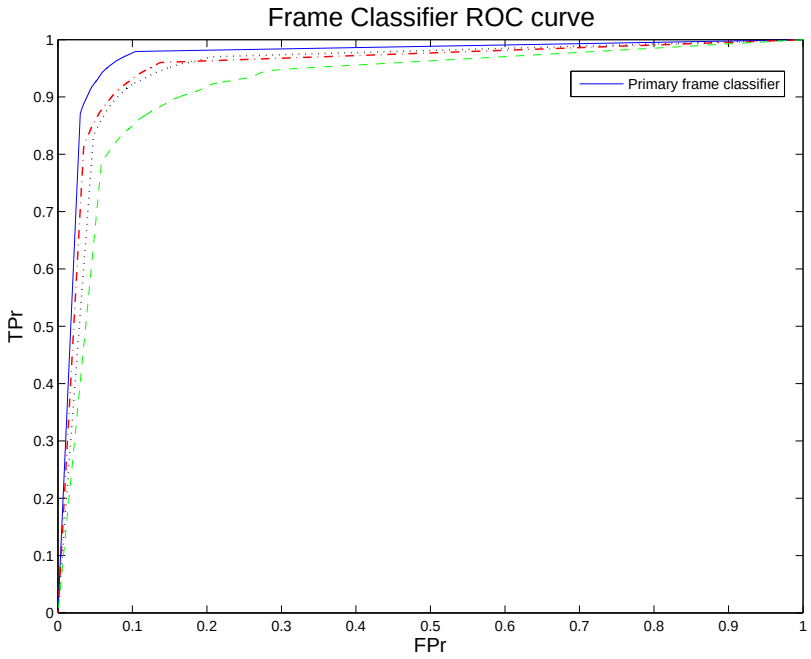


Figure 4-1: Frame classifier ROC curves, for four different frame classifiers. The solid blue line is the ROC curve for the frame classifier used in these evaluations.

A number of modifications were evaluated to improve the frame level performance, such as changing the feature extraction algorithms, modifying the training procedure, and using different core classifier algorithms. These details, while interesting, are not essential to the thesis and are omitted.

4.3 Speech Detector Performance

The speech detector performance measures the quality of the actual output that a human annotator will work with. One way to evaluate the speech detector is to apply it to audio, and have a human look at the segmentation and identify errors. However, this is time consuming and does not permit automatic performance optimization where parameters are adjusted and performance re-evaluated. Alternatively, the speech detector's segmentation can be compared to a human generated, ground truth segmentation. The performance measures described in section 4.1 are used here to evaluate segmentations.

Trial	Tool	Duration	Segment density **a1**	**a2**	Similarity
1	CLAN	5	.483	.527	.939
2	Transcriber	5	.395	.403	.922
3	CLAN	5	.525	.560	.922
4	Transcriber	5	.421	.451	.929
				Average	.928

Table 4.3: Similarity between human segmentations of the same audio stream. This gives a sense of a "good" similarity, and a target range for machine segmentation algorithms.

4.3.1 Human Segmentation Comparison

Before computing the automatic speech detector performance with the similarity and error metrics, what sort of numbers should be expected? Assuming two human annotators segmented the same audio, how similar are their segmentations? Four manually segmented blocks of audio from two transcribers were compared. These were segmentations generated using CLAN and Transcriber. Table 4.3 summarizes the comparison.

One way to interpret a similarity of ".939", as in trial 1, is to assume that two human segmentations should be roughly correct, and thus .939 is an example of a "good" similarity. However, the meaning of this number is still somewhat vague. Another way to look at it is to consider a random segmenter, which for each frame assigns the label "speech" with probability p, independent of the audio. To be fair, p should be chosen so that the resulting *expected* speech density matches the density obtained by the human. In this case, p should be chosen to be the observed speech density. Then, two such segmenters with parameters p_1 and p_2 will produce an expected similarity of $p_1p_2 + (1-p_1)(1-p_2)$, which for trial 1 is about .52.

Now consider a model which *does* use the actual source audio. Conditioned on the true frame label, each segmenter has a probability of producing that frame label or an error. More compactly, if a frame is truly speech, then segmenter 1 produces "speech" with probability $\Pr_1(s|s)$ and "non-speech" with probability $1 - \Pr_1(s|s)$. Similarly, if the frame is truly not speech, segmenter 1 may return "non-speech" with probability $\Pr_1(\bar{s}|\bar{s})$ or an error with $1 - \Pr_1(\bar{s}|\bar{s})$. Assuming the segmenters are independent given the observed frame,

the similarity can be thought of as the chance that the segmenters agree, subject to their respective probabilities of being correct.

A similarity of .939 means that the two segmentations agree on about 94% of all frames, and disagree on about 6%. For a 5 minute block of audio, they disagree for a total of about 18.3 seconds. This disagreement could be due to missing segments, or differing boundaries. For an average speech segment length (in trial 1) of 1210 ms, this corresponds to 15 missing speech segments out of the roughly 125 speech segments found by both annotators. Assuming no missed speech segments and only boundary differences, this corresponds to boundaries being off an average of 144 ms for each speech segment, which is really more like 72 ms for each boundary (since all segments except the first and last have two adjustable boundaries). It's not so unreasonable to assume that a human annotator could pad 72 ms to the beginning and end of each speech segment, though it seems less likely that they will clip a speech segment.

Rather than focus only on the similarity measure, how can the error measures be evaluated given the human segmentations? One simple way is to assume one human segmentation is the "true" segmentation, and evaluate the other in terms of it. That should give an idea of the appropriate range of numbers for a "good" segmenter. The advantage of looking at the error rather than the similarity is that it indicates the performance of the speech class, which is what is most important for this application. Tables 4.5, 4.6 and 4.7 show the error performance of each human segmentation relative to the other. Note the precision and recall are simply swapped in each table.

Returning to trial 1, an 89.3% recall for **a1** relative to **a2** implies that,

$$\text{FN time} = \text{duration} \times \text{density} \times (1 - \text{recall}) \tag{4.1}$$

minutes of speech was missed by **a1** relative to **a2**, which is about 16.9 seconds in this case. An FPr of .011 implies that

$$\text{FP time} = \text{duration} \times (1 - \text{density}) \times FPr \tag{4.2}$$

minutes of non-speech, according to **a2**, was labeled as speech by **a1**, which is roughly 1.6 seconds. This is a breakdown of the 18.3 seconds of disagreement between segmentations. This analysis of the similarity and error metrics links the performance measures back to actual audio times and the transcription task, which is dependent on false positives and false negatives.

4.3.2 Machine Segmentation Comparison

The same audio blocks were used as input to the automatic speech detector in order to evaluate the similarity, as well as error, with respect to the human segmentations. The "M vs H" column of tables 4.4, 4.5, 4.6 and 4.7 show the average pairwise performance of the machine segmenter against the human segmentations. Perhaps more interesting than the performance measures are the total false positive and false negative times, as compared to the same times for human segmentations. This is a conversion using equations 4.1 and 4.2 from performance measures to actual seconds of false positive or false negative audio.

In the case of false negative times, the numbers are quite close to the human performance numbers. This indicates that the machine segmenter misses about as much speech as another human would. On the other hand, the false positive times are higher, indicating that the segmenter is fairly aggressive, and human transcribers working with these segmentations will encounter more non-speech than if they were working with a manually created segmentation.

4.3.3 Extensions to Similarity and Error Metrics

One problem with the above similarity and error metrics is that they consider the segmentation agreement at a granularity that is not necessarily important to a human transcriber. Most human annotators would consider two different segmentations equally acceptable even if their boundaries didn't precisely line up. In fact, they need not even have the same number of boundaries – an utterance containing multiple words might just as easily be represented as a single segment, split into phrases, or possibly even segmented into individual words.

Trial	Similarity	
	H vs H	M vs H
1	.939	.861
2	.922	.852
3	.922	.907
4	.929	.765
Average	.928	.846

Table 4.4: Similarity comparison between two human segmentations, and the average similarity between the automatic segmenter and the human segmentations.

Trial	Precision		
	a1 vs **a2**	**a2** vs **a1**	M vs H
1	.989	.893	.806
2	.912	.893	.767
3	.956	.891	.893
4	.952	.888	.676
Average	.952	.891	.786

Table 4.5: Precision comparison between **a1** and **a2**, and the average precision of the automatic segmenter relative to both human annotators.

Trial	Recall			FN time		
	a1 vs **a2**	**a2** vs **a1**	M vs H	**a1** vs **a2**	**a2** vs **a1**	M vs H
1	.893	.989	.950	16.92	1.59	7.58
2	.893	.912	.904	12.94	10.43	11.49
3	.891	.956	.926	18.31	6.93	12.04
4	.888	.952	.885	15.15	6.06	15.04
Average	.891	.952	.916	**15.83**	**6.25**	**11.54**

Table 4.6: Recall comparison between **a1** and **a2**, and the average recall of the automatic segmenter relative to both human annotators.

Trial	FPr			FP time		
	a1 vs **a2**	**a2** vs **a1**	M vs H	**a1** vs **a2**	**a2** vs **a1**	M vs H
1	.011	.107	.227	1.56	16.6	33.71
2	.058	.071	.182	10.39	12.89	32.81
3	.044	.110	.112	5.81	15.68	15.37
4	.037	.087	.327	6.09	15.11	55.33
Average	.038	.094	.212	**5.96**	**15.07**	**34.31**

Table 4.7: FPr comparison between **a1** and **a2**, and the average FPr of the automatic segmenter relative to both human annotators.

Without over-complicating the metric, adding the ability to ignore precise boundary differences could be a useful feature. Yet while more elaborate error metrics may have some advantages, the error metrics used here have the benefit of being simple and interpretable. Further development of the error metric may be a good candidate for future work.

4.3.4 Summary

To summarize, the performance metrics developed in this chapter were applied to evaluate the automatic speech detector. In order to understand the meaning of the metrics, they were applied to human segmentations, and then to automatic segmentations of the same source audio. This demonstrated that the automatic speech detector was comparable to a human in terms of recall, but had more than twice the false positive rate than a human. The performance numbers were also converted back to concrete false positive and false negative times, reflecting the actual amount of audio for each error type. As expected, the results show that the automatic segmenter is quite comparable to a human in terms of false negatives, but it is more aggressive than a human in terms of producing false positives. Out of four 5-minute audio segmentations, the automatic segmenter missed an average of about 12 seconds of speech audio, which is between the 6 seconds of speech audio missed by one human segmenter and the 16 seconds missed by another. On the other hand, it produced an average of about 34 seconds of false positive audio, compared with about 6 seconds and 15 seconds produced by human annotators, relative to each other. Whether this tradeoff of producing more false positives than false negatives is desirable in terms of the total annotation time will be answered in Chapter 6.

Chapter 5

Semi-automatic Transcription

5.1 Transcription Process

The automatic processes are only successful to the extent that they provide conditions for the human transcriber to work more effectively. The tools used by the human transcriber, primarily TotalRecall and BlitZcribe, leverage the automatically generated metadata in order to simplify the annotation task. They are designed to be complementary, though both are critical parts of the process. Because the automatic speech detector is error prone, it is important to incorporate human oversight without making the task overly burdensome. The following sections detail the transcription process of the automatically identified speech segments.

5.1.1 Assignments

Transcribing the speech in the HSP data will require significant human effort, and will be performed by multiple annotators. In order to manage the work, assignments are created and assigned to individual annotators. An assignment is simply a block of time to be transcribed by a particular annotator. Assignments are created in TotalRecall, and can be managed by an administrator. This helps an administrator track the progress of the

transcription task as a whole, and the progress of individual transcribers. A simple approach to assignments is to assign all speech segments in the timeblock for transcription, but one problem with this is that not all speech is necessarily relevant to the child language acquisition study. For example, if the child were asleep at the time, or in a room where they could not hear the speech, it need not be transcribed. Such speech might also be of a private nature between adults, an additional reason not to transcribe it.

5.1.2 Identifying Relevant Data

In order to identify relevant audio, the location and whether the child is awake or asleep is manually annotated. This is is carried out in TotalRecall using video volumes and the video player. Experience with this task indicates that a day of data can be manually annotated in about two hours. By knowing the child's location throughout the day, a set of potentially relevant audio channels is inferred. The relation between a video channel and a set of potentially relevant audio channels is fixed ahead of time, based on the house layout and knowing which microphones could pick up sounds occurring in different video zones. A set of speech segments for transcription is defined by combining the annotated location of the child with the assignment.

5.1.3 Transcription

When a transcriber is ready to transcribe, they start BlitZcribe and are presented with a list of open assignments. Choosing an assignment loads all relevant speech segments and presents them in a list view. BlitZcribe is designed with speed in mind, and to minimize the cognitive load on the user. It does this by only providing functionality for playing a segment, typing a transcription, or indicating various types of errors, such as whether the segment is not speech (ie. is a false positive) or is clipped. The segments are also short enough to remember, which reduces the number of times the segment is replayed. All of this helps optimize the LISTEN and TYPE steps of transcription.

Transcription is straightforward to explain, but doing it well requires diligence and good judgment. Familiarity with the data is extremely helpful, and transcriber skill definitely improves with practice. Perhaps the main difficulty with transcription for the HSP data is the informal nature of much of the speech. Child speech can be hard to understand, or the child may be babbling, and adult speech may not be well articulated. Furthermore, in the HSP context, multiple people may be speaking at the same time or there may be background noise.

Adopting simple conventions can help mitigate the difficulties of transcription. Table 5.1 summarizes the conventions used.

yyy	Use for a word (or phrase) which cannot be understood
[?]	Use when a transcription is entered, but the transcriber is unsure if it is correct
[v]	Use for a non-speech vocalization, for example, a yawn or cough
[b]	Use to indicate baby babble
;	Use to separate utterances by different speakers

Table 5.1: Transcription conventions

It was originally proposed to mark all baby babble by a "[v]", but this was unsatisfying to transcribers who preferred their transcription to be as meaningful as possible if it didn't require extra effort. So "[b]" was chosen for baby babble. When transcribers have worked with the data and tools enough it is important to trust their judgment, and any special sound may be transcribed as long as it occurs in square brackets. For example, a yawn could be indicated by "[yawn]". If an utterance contains multiple speakers, ";" helps to group utterances together by speaker, and it also serves as an indicator that the segment may need further processing.

In addition to transcription conventions, hot-keys for marking segment errors, such as "not speech" (ie. a false positive) or "cut off speech" are provided. Figure 2-9 shows some of these transcription conventions and segment error flags in use.

Transcription proceeds by opening the assignment, and for each segment in the transcription

list, listening, typing, and marking errors. Hitting "return" advances to the next segment, "tab" replays the segment, and the up and down arrows can be used to navigate (and automatically play) the segments. If a segment is left blank, hitting "return" marks it as non-speech before advancing. In this way, the user need not move their fingers from the keyboard.

Transcribers can save their work before it is complete, which stores it in the metadata database. When they have transcribed all segments (and marked errors appropriately) they can mark the assignment as "complete" so it will not appear the next time they view open assignments.

5.1.4 Finding Missed Speech

In BlitZcribe, the transcriber only hears the audio for speech segments that have been previously created. If the speech detector misses speech, then the transcriber will never transcribe it in BlitZcribe. Safe mode adds a CHECK step to the FMLT process, in order to monitor and correct errors that result from the automatic process. This is important for ensuring the quality and completeness of the transcribed data, as well as for improving the automatic speech detector by improving the training set.

Missed speech can occur for a variety of reasons, but candidate speech segments can often be quickly identified by visually scanning a spectrogram, and verifying whether they are speech by listening to the audio. TotalRecall is the appropriate tool for browsing audio and viewing spectrograms, however, the usual mode of browsing data in TotalRecall can present an information overload. For HSP, fourteen audio channels, eleven video channels, potentially hundreds of automatically detected speech segments, and people moving between channels can obscure any missed speech. Therefore, a simplified view is provided for this purpose. This view displays the virtual channel of *all* potentially child relevant audio (not just speech) for an assignment. Playback in a virtual audio channel plays the appropriate source channel, automatically switching channels as necessary. The spectrograms for a virtual audio channel are composites of the source channel spectrograms. All

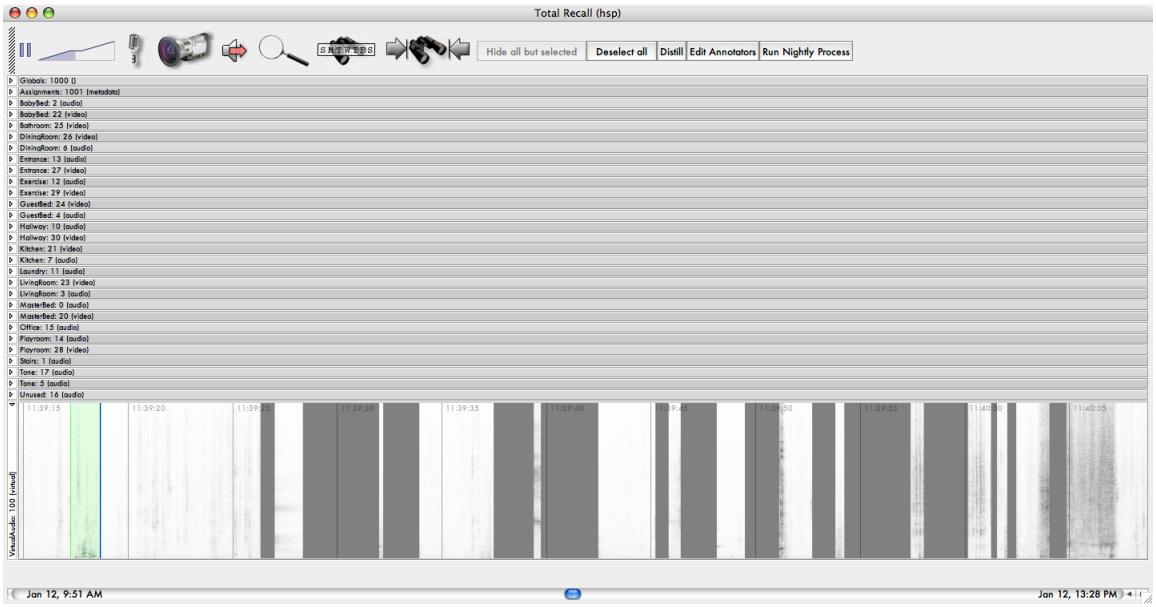


Figure 5-1: TotalRecall displaying the virtual channel. Note the missed speech that has been highlighted, and the grayed-out regions where speech had been detected.

speech segments that have been previously identified (and will thus appear in BlitZcribe) are "grayed out" of the spectrogram. In this way, the spectrogram that remains shows only the audio that *could* contain missed speech. Roughly, this view collapses information across both channels and time.

When annotators wish to scan an assignment for missed speech, they open TotalRecall, load the virtual channel for their assignment, and create segments for any missed speech they find. These segments are created in the virtual channel, and then automatically migrated to the source audio channel. Figure 5-1 shows a screenshot of TotalRecall being used to find missed speech.

5.2 Performance Factors

Performance on a transcription task can be measured in terms of speed and accuracy. Since there is very little one can say about accuracy without a human verified baseline, it is of more immediate interest to have as much human annotated and reviewed data as possible.

The operating assumption is that significant speed increases can be achieved by increasing efficiency. Thus, the primary goal is to make the process faster through human-machine collaboration.

5.2.1 Modeling Transcription Time

There are a number of issues which occupy the attention of a human transcriber. These factors affect transcription time, and are explicitly considered here.

One way to model the time demands of a given transcription task is as follows. There is an ideal "minimum" amount of time required to transcribe the speech in an assignment. Assuming a "perfect" segmentation, and no errors, the time taken depends only on how much speech there is, and the inherent complexity of the speech. For example, an assignment in which an adult is reading a story to a sleepy (and quiet) child might constitute a lot of speech and yet be fairly simple to transcribe. Multiple speakers overlapping with background noise could actually result in a shorter transcript and yet be more difficult to transcribe.

Additional time is required of the human transcriber due to errors and imperfections in the segmentation. A perfect segmentation is not a well-defined notion (at least at the present time), rather it is used as a placeholder for allowing that while many different, error-free segmentations may exist, there may be an optimal segmentation for a transcriber to work with. In other words, if it were possible to measure the time taken for the same transcriber to transcribe different segmentations of the same audio, the perfect segmentation would be that which results in a minimum time required. The significance of different error-free segmentations on transcription time is currently unknown, although the heuristic of choosing segments which are not too short or too long is employed.

While the perfect segmentation is an abstraction, segmentation errors are not. The two primary error types discussed above are *false positives* and *false negatives*. These are also known as type I and II errors, respectively. In addition, clipped speech and excessively long

speech segments constitute errors. This work focuses only on the two primary error types, and their associated costs.

5.2.2 Cost of Errors

The cost of false positives and false negatives can be considered from two viewpoints. The first viewpoint considers what impact each error type has on the *scientific* goal of the Human Speechome Project. Any false negative which is not corrected results in speech that will not be analyzed. However, one must weigh this against the natural amount of missed speech. Over the course of three years, vacations will be taken, the child will be outside, the recording system may be off briefly – this all results in naturally missed speech, or speech that is never collected in the first place. Given this, a small percentage of false negatives does not significantly damage the quality of the HSP corpus.

The second viewpoint for considering errors is that of the added cost of transcription due to the additional human effort required. Handling false negatives requires scanning the virtual channel spectrograms as described in section 5.1.4. Handling false positives is accomplished in BlitZcribe. How much time is added to the transcription task to handle each of these error types? Correcting a false negative error requires scanning the virtual channel spectrogram, potentially listening to the audio, and creating a segment and annotation. A false positive requires listening to the complete audio clip plus the time required to mark it. Adding spectrogram images to BlitZcribe could be helpful for identifying false positives, although from a design perspective complicating the user interface may cause more harm than good.

So far, the hypothesis has been that false negatives are more costly both in terms of the project goals as well as the time required to handle them. This is simply because identifying a false positive depends only on the short, bounded duration of audio in the candidate speech segment. The BlitZcribe interface has been designed to make it easy to handle these false positive errors. On the other hand, a false negative can be anywhere in the variable amount of audio left over in the virtual channel for the assignment. On the assumption that false negatives are more costly than false positives, in terms of human transcriber effort, the

system has been built to err on the side of producing more false positives. Studying and quantifying the relationship between the different error types and human performance is the subject of the next chapter.

Chapter 6

Semi-automatic System Evaluation

This chapter evaluates the semi-automatic transcription approach. It begins by evaluating transcription times using CLAN, Transcriber and the new system. The direct comparison shows that the new system is at least 2.5 to 6 times faster than existing approaches. However, there are some new factors in the semi-automatic system that must be considered. Specifically, false positive and false negative errors caused by the automatic speech detector increase total transcription time. Experiments show that false negative errors require a factor of about eighteen times more effort to correct than false positive errors, which provides a principled scheme for tuning the automatic speech detector. The chapter concludes with a summarization and discussion of the results.

6.1 Evaluating Manual Transcription

As a starting point to evaluating the semi-automatic system proposed in this thesis, it is useful to determine the baseline performance using purely manual methods. How long does transcription take using a purely manual system? For this evaluation, CLAN and Transcriber (see section 2.3) were selected, as well as TotalRecall. While both CLAN and Transcriber fall short in that they do not handle multi-track audio, this was ignored because

it was assumed that a separate (manual or automatic) procedure could identify the relevant audio channels.

CLAN was used in "sonic mode," in which the waveform is displayed at the bottom of a text editor. The user transcribes by highlighting and playing a block of audio, typing a transcription, and then binding the transcription to the audio segment using a key combination. In Transcriber, key combinations can be used to play or pause the audio. A segment break is created whenever the user hits the "enter" key. The segment break ends the previous segment, and begins a new one. Each segment corresponds to a line in the transcription window. By typing, a transcription is entered which is bound to the current segment. One approach to transcription in Transcriber is to listen to the audio and mark speech boundaries roughly, then go back and adjust the boundaries and transcribe the individual segments. Alternatively, one may transcribe and segment simultaneously.

The basic experimental procedure was as follows. First, several short blocks of audio containing speech were identified in the HSP data for transcription. Each annotator would separately transcribe the same blocks of audio using the same tool, and use a stopwatch program to time the how long the task took. Six audio clips were exported from the HSP data, five minutes each, from different times of the same day in two separate rooms. These blocks of time were selected from a single channel that contained heavy speech activity. Before beginning for the first time with a tool, a shorter 1 minute block of audio was transcribed as a warm up. This was not timed, as it was only intended to familiarize the user with the tool.

Transcription for this experiment used the same conventions as used for BlitZcribe. However, in CLAN, identifying the speaker is part of the syntax. Each transcription line begins with a speaker identity code. In Transcriber, it requires more effort to annotate speaker turn taking, and in BlitZcribe, checking or correcting automatically identified speaker labels is not yet implemented. To compare these three systems fairly, speaker identification was skipped in Transcriber. It was performed in CLAN, because it is part of the syntax and it should take about as much effort to type the true speaker identity code as a fixed code for all speakers. This does result in a richer transcription when using CLAN, so separate

Trial	Tool	Speech duration		Annotator time		Avg time factor	
		a1	**a2**	**a1**	**a2**	Actual	Speech
1	CLAN	2.42	2.64	50	50	10	19.8
3	CLAN	2.63	2.80	44	45	8.9	16.4
20	Transcriber	2.78	2.52	34	32	6.5	12.3
30	Transcriber	2.01	1.94	27	26	5.3	13.5
16	TotalRecall	3.24		22		4.4	6.79
17	TotalRecall	3.17		21		4.2	6.64
2*	Transcriber	1.98	2.02	46	49	9.5	23.8
4*	Transcriber	2.11	2.26	47	41	8.8	20.2

Table 6.1: Transcription time in minutes. Each trial is on a separate five-minute block of HSP audio from a single channel, which contained heavy speech activity. The speech duration indicates the total duration of speech segments found by each annotator. Audio is from either the living room or kitchen only, on Jan 10 and Jan 15, 2007. For trials 2* and 4*, speaker identity was also annotated, which makes Transcriber significantly slower.

trials with speaker annotation were performed in Transcriber, to compare it to CLAN and to check against results reported in the literature.

Finally, an informal test was conducted using TotalRecall for manual transcription. Hotkeys can be defined in TotalRecall for performing different tasks, and in this case one hotkey was defined for creating segments and another for creating a transcription annotation. As soon as a transcription annotation is created, the transcription field is activated and the user need only type. Alternatively, an annotator might segment the audio first, and then transcribe in a batch. If done in this manner, as soon as a transcription is entered, TotalRecall advances to and plays the next segment so the annotator can listen and type, without worrying about navigation. This task was performed by annotator **a1**, who in practice did both segmentation and transcription at the same time.

Table 6.1 collates the results of manual transcription using these tools. The first thing to notice is the consistency in the time per transcriber for each trial. The average transcriber performance is not far from the observed performances. The second thing to look at is the speech duration. Each trial consisted of five minutes of actual audio time, or "house time" from the HSP corpus, and about half of the actual audio time was speech. So the trials were fairly similar in terms of the amount of transcription needed. On the whole, the conditions for comparing these tools are favorable. The results from this evaluation

show that CLAN is the slowest tool, requiring a factor of about nine to ten times actual time for transcription, or about eighteen times speech time. Transcriber requires about six times actual time, or thirteen times speech time. Interestingly, with speaker annotation, Transcriber is the slowest, requiring about nine times actual time, or twenty-two times speech time. TotalRecall is the fastest tool, requiring just over four times actual time for transcription. Conversations about the user experience indicated that the spectrograms were significantly more useful than the waveform in the other programs. Scrolling through audio is also very smooth, hot-keys permit quick segmentation and annotation, and the responsiveness of the interface all contribute to TotalRecall being about 25% faster than Transcriber and 50% faster than CLAN.

These manual transcription times are at the low end of the range reported in [53], and much faster than the factor of fifty reported in [3]. This could be for several reasons. First, transcribing five minutes of audio can be done in one sitting, but after about an hour of transcription most annotators would need to take a break. The transcription time factor may actually increase for longer transcription assignments. Transcription detail was also fairly basic, without speaker identification (except in CLAN) or other annotations. More detailed annotation or finer segmentation would certainly require more time.

6.2 Evaluating Semi-automatic Transcription

This section evaluates the new semi-automatic transcription system. It answers the basic question of how long transcription takes using the semi-automatic system, in order to compare against manual approaches. Blocks of data were assigned to annotators, who used a stopwatch program to time how long the task took in either TotalRecall or BlitZcribe (for BlitZcribe, the stopwatch was added to the program itself.) Most blocks were assigned to multiple annotators, although some trials were performed by just one annotator. Each annotator used a Macintosh computer and sound-muffling headphones. Time for launching the applications was not included.

In the semi-automatic transcription methodology (section 2.5), transcription can be per-

Trial	Audio duration		Annotator time			Avg time factor	
	Actual	Speech	**a1**	**a2**	**a3**	Actual	Speech
5	5	3.13	10	13		2.43	3.88
6	15	7.96	27			1.8	3.39
7	15	8.75	26			1.73	2.97
8	5	1.92	5			1	2.6
9	5	2.52	8	9.35	11	1.89	3.76
10	5	0.82	1.5	2	2	.37	2.22
11	10	5.95	18			1.8	3.03
14	10	4.23	11.5		14 .1	1.28	3.03
					Average	1.54	3.11
					Std Dev	.64	.56

Table 6.2: Transcription times using the semi-automatic system in fast mode.

formed in either safe or fast modes. In safe mode, TotalRecall is used to check for and mark any false negative speech segments. In fast mode, only the segments found by the speech detector are transcribed. This evaluation looks at both of these modes, beginning with fast mode.

6.2.1 Fast Mode

Three transcribers used BlitZcribe to transcribe audio from the HSP corpus, and used the timer or a separate program to time, in minutes, how long transcription took. Table 6.2 shows transcription time, and calculates the time factors in order to compare with manual transcription.

The time factors for speech transcription with the semi-automatic system in fast mode are about 1.5 times actual audio time, and 3 times the total speech time [1]. This is significantly faster than manual transcription using any tool. It is almost four times faster than Transcriber, and more than six times faster than CLAN relative to the actual audio time to transcribe. Similar factors hold for the speech time.

Four additional transcription trials were performed under different speech detection con-

[1]For these experiments, the total speech time actually includes manually identified false negatives. So this time factor really reflects the time to transcribe *all* speech in the audio, and not just the audio found by the speech detector. This causes a slight over-estimate of the average speech time factor for fast mode.

(a) Manual segmentation

Trial	Audio duration		Annotator time			Avg time factor	
	Actual	Speech	**a1**	**a2**	**a3**	Actual	Speech
12-	10	4.13	10.5			1.05	2.54
13-	10	5.93	15.5			1.55	2.61
					Average	1.3	2.58
					Std dev	.35	.05

(b) Automatic segmentation, with random segments added

Trial	Audio duration		Annotator time			Avg time factor	
	Actual	Speech	**a1**	**a2**	**a3**	Actual	Speech
18+	5	4.98	10.5	15		2.55	2.56
19+	5	5.17	12	18	17.33	3.16	3.05
					Average	2.85	2.81
					Std dev	.43	.35

Table 6.3: Semi-automatic transcription in fast mode, under special conditions. Table 6.2(a) shows the transcription times in BlitZcribe when the speech was manually segmented. Thus, there will be no false positives for the transcriber to work with. In table 6.2(b), the non-speech audio blocks, according to the automatic speech detector, were *randomly* segmented and relabeled as speech. Thus there would be many false positives in BlitZcribe.

ditions. These times are presented in table 6.3 The "-" indicates that a purely manual segmentation was performed in TotalRecall on the data, and thus no false positives should occur in BlitZcribe. The "+" indicates that the automatic speech detector was run, and then all "non-speech" segments were *randomly* segmented (with the same minimum and maximum length constraints) and also labeled as speech. Thus, the annotator would encounter significantly more false positives in BlitZcribe, but no speech would be missed. These results illustrate a few points. First, the manual segmentation results in faster transcription times as compared against the time factors in table 6.2. This is expected, since there will be no false positives, and the speech segmentation may be better. On the other hand, the automatic segmentation with random segments has a slower actual time factor, since there are many false positives, but a faster speech time factor. The faster speech time factor is likely due to the fact that it is faster to annotate a false positive in BlitZcribe than to produce a transcription. So the proportion of true speech relative to the total speech time is lower, implying less transcription is needed.

Figure 6-1 incorporates all the data in tables 6.2 and 6.3, and shows a consistent linear

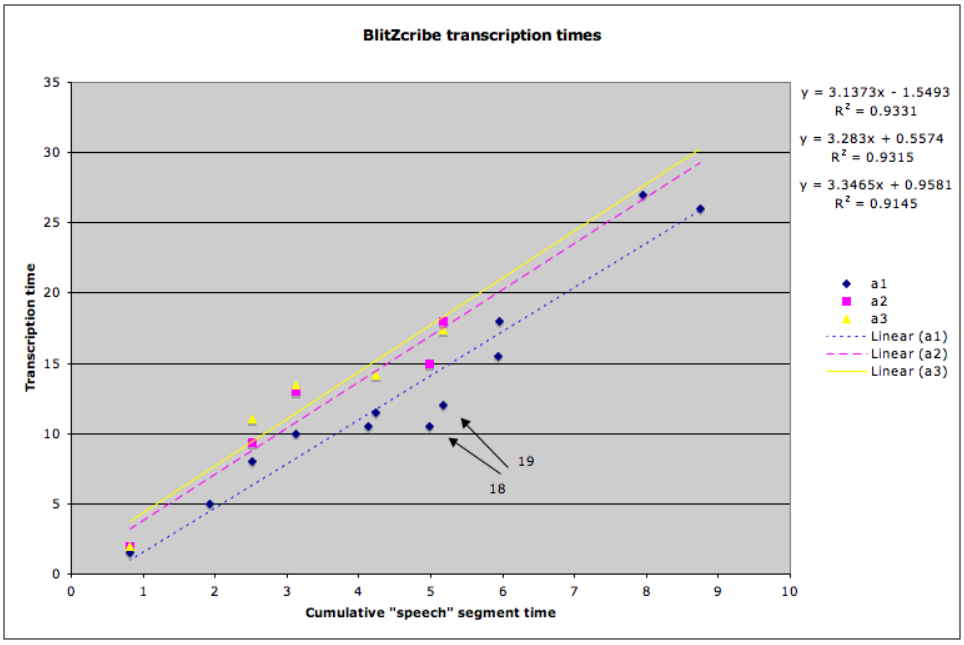


Figure 6-1: Transcription time vs. cumulative "speech" time in BlitZcribe

relationship between the speech time and the transcription time, for each of the three annotators. This makes intuitive sense – in order to transcribe an assignment, one must listen to all the audio segments. As explained above, trials 18 and 19, which were the trials in which all non-speech was randomly segmented and relabeled as speech, take less time than expected because annotating false positives is faster than transcribing speech. The amount of effort introduced by false positives is considered in more detail later in this chapter. While this linear model for transcription time is simple, it is significant and extremely useful as a predictive model for estimating the amount of work that will be required to transcribe speech in the HSP corpus.

6.2.2 Safe Mode

Safe mode transcription is the same as fast mode, but in addition the transcriber searches for missed speech, and manually creates segments for any speech found so that it can be transcribed in BlitZcribe. This step helps monitor the automatic speech detector performance, identifies errors that can be used to improve the system, and helps maintain the

Trial	Audio duration		Segments found		Time		Avg time factor		
	Actual	Remaining	**a1**	**a2**	**a1**	**a2**	Actual	Remaining	Segment
5	5	1.93	6	4	3	3	.6	1.55	.63
6	15	7.1	6		3.5		.23	.49	.58
7	15	6.64		33	4		.27	.60	n/a
9	5	2.65	14	16	5.2	5.15	1.04	1.95	.35
10	5	4.56	23	21	6.5	4.5	1.1	1.21	.25
11	10	4.26	16	18	7	8.33	.77	1.80	.45
14	10	6.77	86		14.75		1.48	2.18	.17
21	10	4.31	7		5.5		.55	1.28	.79
22	15	7.52	19		7.63		.51	1.01	.40
23	5	3.5	8		3.75		.75	1.07	.47
24	10	3.92	14		5		.5	1.28	.36
						Average	.71	1.31	.44
						Std dev	.37	.53	.18

Table 6.4: Time spent identifying missed speech in TotalRecall for safe mode transcription. The time factors are the ratios of annotator time to the amount of actual audio time, the amount of remaining audio, and the number of segments found.

quality of the HSP dataset. TotalRecall is used in a mode that displays a virtual channel, which shows only the portions of spectrograms that could contain missed speech.

Transcribers used TotalRecall and a separate stopwatch program to time how long it took to find missed speech. Table 6.4 shows these results. The column "remaining time" refers to the amount of non-speech time, and "segments found" to the number of missed speech segments that were found. This table shows that finding missed speech takes a factor of about .75 times actual time, and 1.3 times remaining time. While these factors may over-simplify the model for the time to find missed speech, they still provide a useful estimate of the additional time factor required to transcribe in safe mode. These additional factors mean the total transcription time in safe mode for the semi-automatic system should be about $1.54 + .71 = 2.25$ times actual time. The total transcription time relative to the speech time is slightly more complicated, since remaining time is defined in terms of speech time. It is $1.3 \times (\text{actual time} - \text{speech time}) + 3 \times \text{speech time}$. The average speech density reflected by the times for the trials in tables 6.2 and 6.4 is about .46. Making a very rough assumption of a typical speech density of .5 results in total transcription time factor of about 4.3 times speech time. Interestingly, the assumption that speech time is about half

Tool	Time factor	
	Actual	Speech
CLAN	9.5	18.1
Transcriber	5.9	12.9
TotalRecall	4.3	6.7
Semi-automatic (safe)	2.25	4.3[2]
Semi-automatic (fast)	1.54	3.11

Table 6.5: Time comparison between all tools

of actual time would mean that 4.3 times speech time is about 2.15 times actual time, close to the direct estimate of the actual time factor. The task time per segment found is also shown in the "segment" average time factor column. Figures 6-2(a) and 6-2(b) show the task time for finding missed speech in safe mode, relative to the remaining audio time and the number of segments found, respectively.

6.2.3 Comparison to Manual Transcription

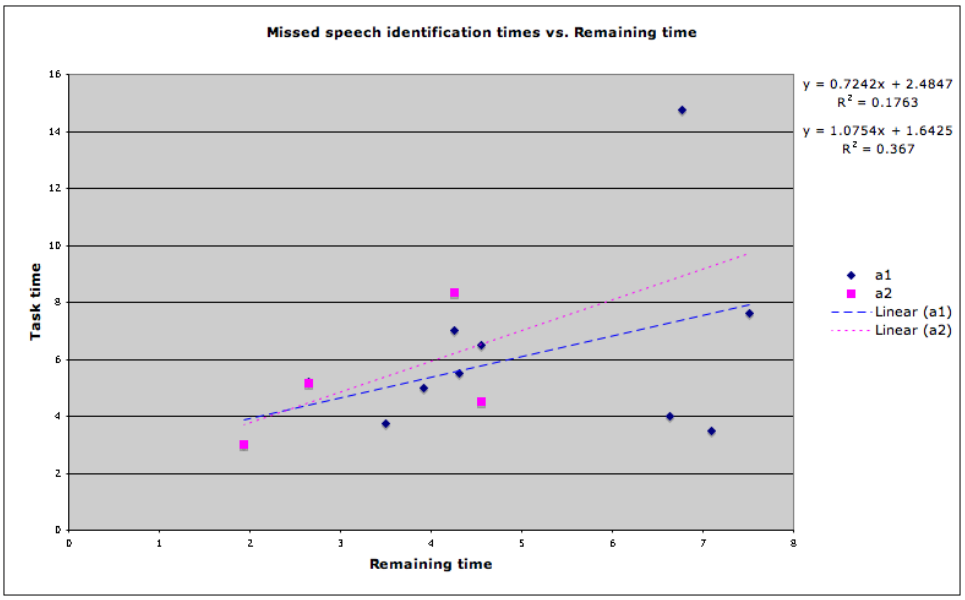
In summary, the evaluation of manual transcription against semi-automatic transcription shows the semi-automatic system to be significantly faster. Table 6.5 compares the time factors for all transcription methods.

Depending on the mode of semi-automatic transcription, it is in the range of 2.6 to 6.1 times faster than CLAN or Transcriber. This is without even considering the limitations of CLAN and Transcriber on a massive corpus of multi-track audio, where isolating the audio for transcription is itself a challenge.

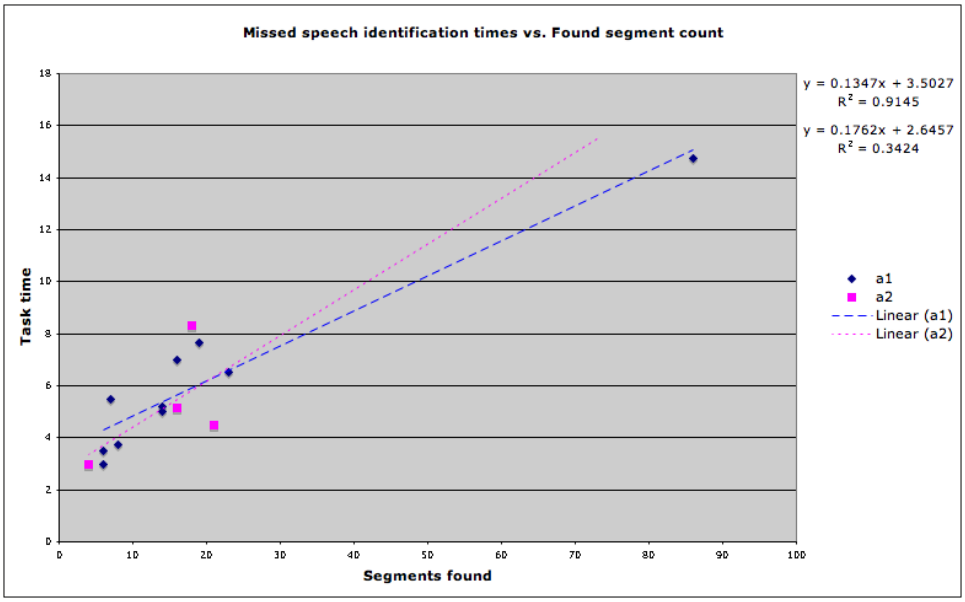
6.3 The Cost of Speech Detection Errors

In manual transcription, the FMLT paradigm means that a human essentially processes all the audio, even audio which does not require transcription. On the other hand, the semi-automatic approach dispenses with the need to have a human FIND and MARK speech,

[2]The estimate of this time factor is based on rough assumptions of speech density, and is explained at the end of section 6.2.2.



(a) Task time vs remaining time



(b) Task time vs found segment count

Figure 6-2: Task time identifying false negatives

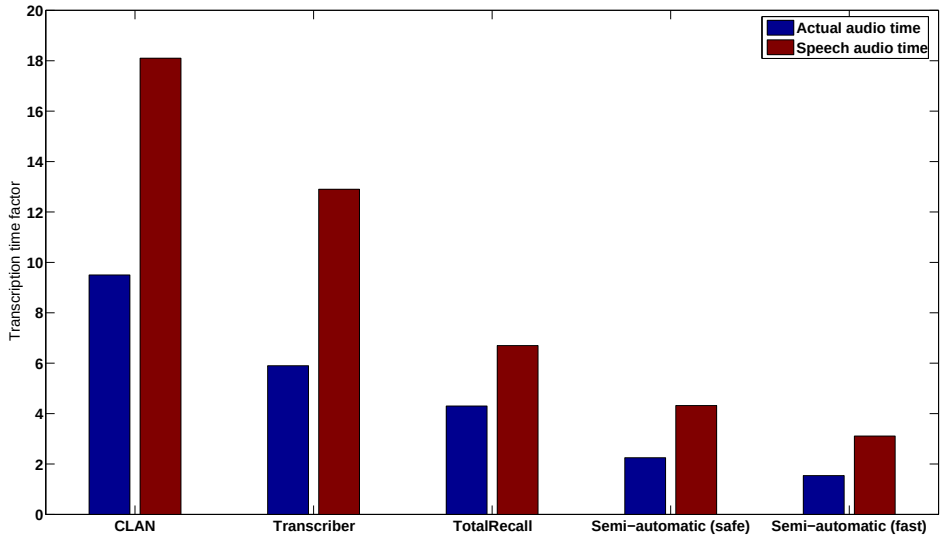


Figure 6-3: Time factor comparison

resulting in significant speedups. However, it also means that errors in these two steps may result from the automatic speech detector. Identifying and correcting these errors adds a time factor which is not present in manual transcription, which is the time required to handle errors finding speech (roughly, false negatives) and errors marking speech (roughly, false positives). This section analyzes the cost, in terms of additional task time, incurred by these errors.

6.3.1 Time Identifying False Positives

Non-speech segments encountered in BlitZcribe, or false positives, are expected to require less transcriber time than actual speech segments. This is because the transcriber need only listen to the audio segment, and if it is not speech, mark it as such and move on to the next segment. Marking false positives in BlitZcribe is done quickly, by hitting "return" without typing a transcription, or by using a hot-key.

In order to understand how much time false positives contribute to the transcription task, the timer module in BlitZcribe accumulates the total time spent per segment. When the

annotator has finished transcribing, a time log is saved for analysis. Annotators were told of this functionality, and were able to see the timer display as well as pause and resume it if necessary, so that a segment would not accumulate time if the annotator stopped working. However, annotators were asked to perform this task at a time when there would be few interruptions.

Figure 6-4 shows the relationship between actual cumulative false positive time and the time spent on these false positives. A simple model of the transcription time required by a false positive is to break it into the time spent *listening* to the segment, and the time spent *marking* the segment. In BlitZcribe, it is reasonable to assume that the action of marking a segment is roughly constant – the annotator uses the same keystroke for a false positive regardless of how long it is. Under this assumption, the time to handle a false positive should depend on the length of the segment. Figure 6-4 supports this model, showing a linear relationship between the cumulative false positive duration and the time spent annotating false positives. Interpreting the slope of each trend line as the factor required to handle a false positive, we obtain roughly 1.25 - 1.75 times real time. Recalculating the trend lines to require a $y-$intercept of 0 results in a range of about 1.5 - 2 times real time. Assuming a $y-$intercept of 0 simply means that if there are no false positives identified, then no time was spent identifying them. However, the positive $y-$intercept may actually be capturing the time required to *mark* the segment in the limit, as actual false positive time goes to zero.

Since the average false positive time observed is about 800 ms, the average amount of annotator time spent per false positive should be in the 1000 - 2000 ms range.

6.3.2 Time Identifying False Negatives

The evaluation of the automatic speech detector showed a false negative rate comparable to a human annotator. This implies that spending time searching for false negatives in TotalRecall may not yield many missed segments. Even so, transcribing in safe mode helps ensure the quality of the system and can help improve performance.

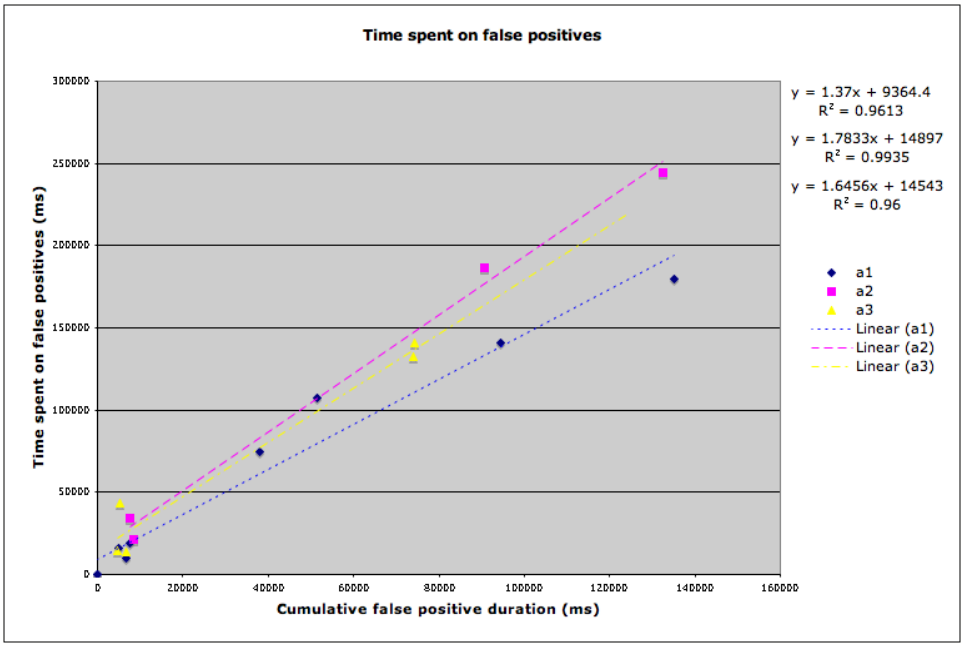


Figure 6-4: Time required to mark false positives relative to the actual duration of false positives, in BlitZcribe

The process for finding missed speech is described in section 5.1.4. The transcriber uses TotalRecall to scan spectrograms which only visualize audio that could contain false negatives. If a speech segment is found, the annotator creates a segment and speech annotation. Figure 5-1 shows TotalRecall being used for this purpose. In cases where speech was missed due to being clipped in an existing segment, it can usually be ignored, since it will later be identified as "clipped" in BlitZcribe. This process goes fairly quickly, generally taking about .75 times the actual audio time. The time to find missed speech correlates positively with the amount of remaining audio and the number of segments found, illustrated by figure 6-2. The expected "cost" of a false negative, in terms of human annotator time, can be estimated as the average time spent per false negative. From table 6.4, the average time per false negative is $.44 \times 60 \approx 26$ seconds, with a standard deviation of about 11 seconds. Rather than attempt a rigorous statistical analysis, we'll wait until the implications of different ranges of false negative times is more meaningful.

6.3.3 Modeling the Time to Identify False Negatives

Unlike the time required to mark false positives, the task time to find missed speech does not correlate as strongly with any single variable, but instead seems to have a more complex relationship. This section develops a model of the time required to find false negatives.

Finding Missed Speech: Subtasks

Procedurally, there are essentially two functions annotators must perform in TotalRecall to find false negatives. They must scan the audio (both visually and by listening to portions of it) and they must create segments and speech annotations. The first requires navigating the audio stream. The amount of time spanned in TotalRecall is still the actual assignment time, but interspersed within this span are visible spectrogram portions. The more spectrogram there is to look at, the more time must be spent visually scanning, listening, and scrolling. Secondly, whenever a missed speech segment *is* found, it takes time to highlight the correct audio block, create a segment, and associate a speech annotation. One would suspect it takes roughly as long to create a 500 ms segment as a 1500 ms segment, since most of the overhead is in highlighting the region and using the hot-keys that create a segment and associate a speech annotation. So, assuming a roughly constant amount of time for each speech segment, the more segments are identified, the more time would be required.

Assuming a roughly constant amount of time to create a segment, the variability in time required per segment should be due to the seek time to *find* each segment. Thus, if the missed speech segments are thinly spread out over the remaining audio time, each segment will require more annotator time to locate. On the other hand, if they are packed close together, more of the annotator time will be spent creating rather than seeking segments, *per segment.* The segment count density quantifies how many segments are packed into the audio span. Using segment count density, the claim is that the annotator labor time per segment will actually decrease as the density of false negative segments increases. This is not saying that the *total* annotator time decreases – it is expected to increase as the number

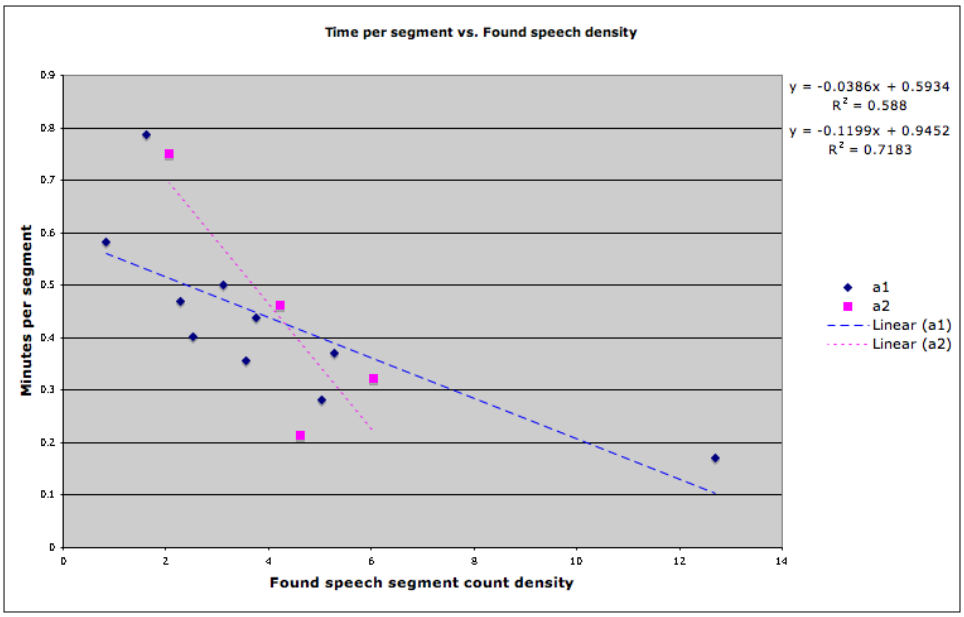


Figure 6-5: Annotator time per false negative, as a function of the density of false negatives

of false negatives increases – but the time per segment should decrease. Figure 6-5 provides some evidence for this claim.

Power Law Model

If this model is roughly correct, we might expect to find a power law relationship between the false negative density and the time per segment, rather than the linear trend shown in figure 6-5. As the density of false negatives increases, the time required per segment should asymptotically approach the assumed "constant" time for creating a segment, since seek time should decrease toward zero. On the other hand, as the segment density decreases, the time required per segment should be primarily due to "seek" time. As the false negative density approaches zero, the time required per segment should tend towards infinity. This would be consistent with a simple power law model of the form $f(x) = ax^{-k}$, $k > 0$. Of course, with fewer segments the total task time should actually complete fairly quickly.

Figures 6-6(a) and 6-6(b) illustrate a power law relationship for the data from annotator **a1**, and show that it better fits the data than the linear trends in figure 6-5. More on power

laws can be found in [48] and [32].

Modeling Manual Segmentation Times in TotalRecall

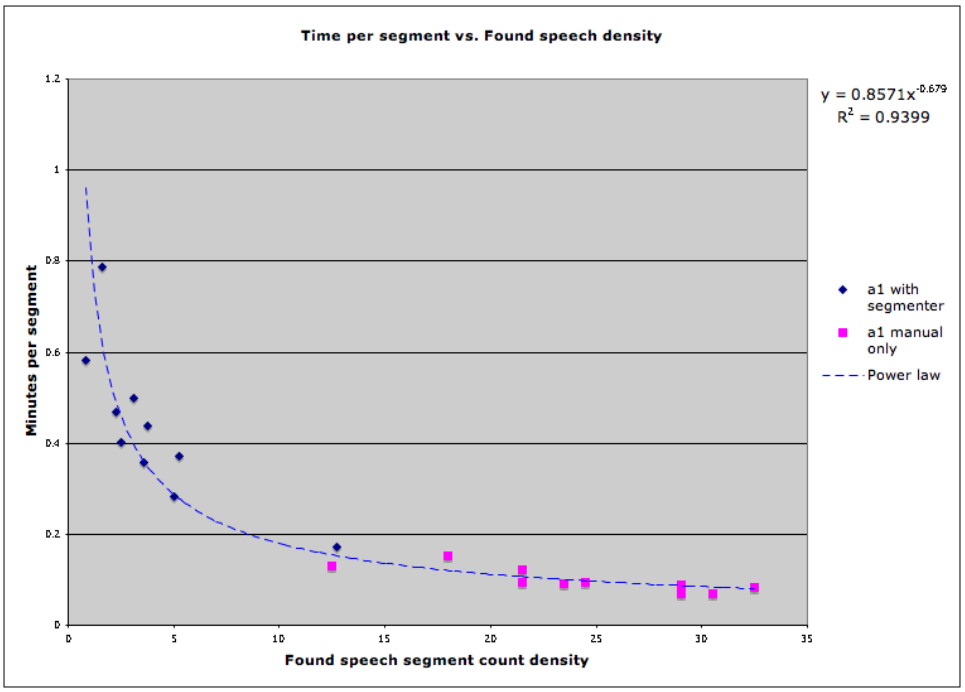
The power law model for identifying false negatives considers the count density of false negatives with respect to the amount of audio the human annotator must consider. If the model is correct, it should also work for a purely manual segmentation, in which the amount of remaining audio is actually the full audio duration.

One way to view manual segmentation is that the speech detector simply missed all speech in an assignment, and the human annotator is identifying false negatives. However, this is not exactly correct, because if the speech detector does not find speech in an assignment, it implies that there is very little speech and the expected speech density is low. On the other hand, if the speech detector is never applied to an assignment, the assignment speech density is completely unknown. In any case, the proposed power law model depends only on the duration of audio to consider, and the number of false negatives. If the model is robust, it should also model the manual segmentation task time.

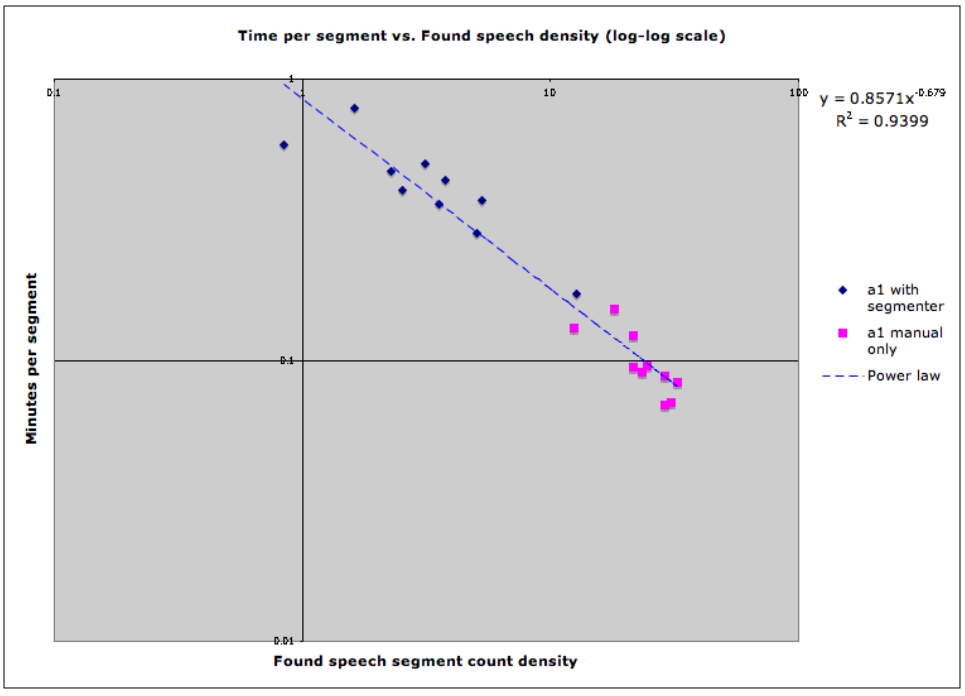
Manual segmentation was performed by annotator **a1** on several single channel audio blocks that contained speech, and the time spent for each block was recorded. The time per segment is plotted aginst the false negative count density, and is shown in figure 6-6(a). The same data is plotted on a log-log plot in figure 6-6(b). The pink square data points show the data for this task, plotted along with the blue diamond data from the previous section for annotator **a1**. Note the higher count density for the manual segmentation task – this is because the automatic segmenter had not been applied and thus there was a significant amount of speech to be processed. The power law model fits the data for both the semi-automatic task and the purely manual task.

6.3.4 Relative Error Costs

The previous sections analyzed the human annotation task in detail to help understand the transcription effort for the Human Speechome Project. A significant result of this analysis



(a) Annotator time per segment relative to found segments per minute of audio time



(b) Log-log plot of the same data

Figure 6-6: Annotator time per false negative, as a function of the density of false negatives. Density is measured as the number of found segments per minute of remaining audio time. This includes segmentations that were done purely manually in TotalRecall. A power law relationship fits both manual segmentation and partially automated segmentation. Data is from annotator **a1**.

is that the system can be tuned to maximize the productivity of the human annotator. In particular, by quantifying the cost of false positive and false negative errors, the automatic speech detector can be tuned to minimize the human task time.

The annotation time required to handle false positive errors in an assignment correlates directly with the cumulative duration of false positives in an assignment. Figure 6-4 illustrates this relationship. Assuming a somewhat conservative factor of 1.75, and an empirical average false positive duration of about 800 ms, each false positive is expected to contribute about 1400 ms of annotator time to the transcription task.

While the model of the time required for false negative errors is more complex, simply using the average time spent per false negative segment is a good starting point. This is about 26 seconds, for both **a1** and **a2**. This implies that false negatives require about $26/1.4 \approx 18$ times more effort than false positives.

Predicting the Task Times

The linear model of the time introduced by false positives, and the power law model for false negatives opens the possibility of predicting the additional transcription time introduced by speech detection errors. However, neither model is immediately predictive because the relationships depend on the speech density and the number of false positives or false negatives, all of which are determined only after human review. Figure 6-2(a) does provide some predictive power for false negatives, because it depends on the amount of remaining audio after speech detection.

Still, it may be possible to estimate the *expected* amount false positive speech and missed speech by combining the the segmenter output and the false positive and false negative rates of the speech detector. From this, the expected false positive duration can be computed to predict the amount of time the annotator must spend marking false positives. Likewise, the expected number of false negatives can be guessed by dividing the expected false negative time by the average false negative duration. This would allow the power law model to be

applied. In fact, applying the power law model in this way does yield similar time factors for annotator time per false negative.

Discussion

The goal of analyzing the additional task time introduced by errors is not to develop predictive models, but to understand the tradeoff between error types. Relative to identifying false positives, finding missed speech takes significantly more time – about 18 times more annotator time than false positives. Given that the algorithm may be tuned to be more aggressive and produce fewer false negatives at the expense of additional false positives, knowing the relative amount of effort between error types is important.

The time spent searching for missed speech relative to the actual number of found segments also does not seem favorable in many cases (eg. trials 5, 6 and 21 in table 6.4). However, there are cases such as trial 14 where there were a significant number of missed segments, so safe mode transcription is important to ensure that the classifier is performing well and to catch cases like trial 14 when they do occur.

However, the time spent searching for missed segments may be better spent transcribing, and perhaps random spot checks for missed speech by an administrator may be sufficient. Alternatively, an interface designed to optimize missed speech detection could be developed that would work on larger chunks of audio, perhaps all audio in a day, rather than only on the audio for the transcriber's current assignment. This would have the added benefit of minimizing time spent switching between applications. Appropriately organizing the workflow should not only reduce the total task time, but also simplify the management of the task as more annotators start working with the data.

6.4 Optimizing the Total System Performance

Chapter 4 introduced performance metrics for evaluating the automatic speech detector. These metrics demonstrated that the automatic speech detector has a low false negative

rate, but a much higher false positive rate. However, it was unknown whether this was desirable, because the impact of false positives and false negatives on the human annotator's workload was not understood. Now that the relative error costs have been quantified, the total system performance can be optimized.

6.4.1 Performance Predictions and Observed Performance

If the performance metrics from Chapter 4 are to be of any use, they should provide some prediction of the speech detector system performance in practice. This justifies optimizing the performance metrics in order to optimize the actual transcription task. The practical system performance is determined by the feedback provided by the human annotator both in TotalRecall and BlitZcribe. Segments marked as non-speech in BlitZcribe and missed speech found in safe mode are used to calculate the observed system performance. The observed system performance for various trials is summarized in table 6.6, and correlates with the expected error determined by the error metrics. The final row shows the expected performance drawn from the human-machine performance comparisons of tables 4.5, 4.6 and 4.7. A separate average observed performance without trial 10 is given, since trial 10 seems to be an outlier with poor precision and recall. Interestingly, this trial had very low speech density. An investigation of the source audio revealed that the child was playing with a noisy toy – and much of the speech was the child making quiet vocalizations. Figure 6-7 shows the source audio, and the speech segments. Segments 2 and 4 were identified automatically, while 1, 3 and 5 were identified by hand. Even for a human annotator, it is difficult to tell there is speech from the spectrogram.

Table 6.6 demonstrates that the error metrics do correlate with the actual observed performance. However, there are a few points worth mentioning about both the observational error and the performance metrics. The observed performance only considers errors at the segment level, so if a segment is too long, but does contain some speech, it will not contribute any false positives. On the other hand, if a segment contains speech, but is clipped at the boundaries, it does not contribute any false negatives. This simple error metric only looks at false negatives which occur as a result of completely missed segments, which are

	Average		
Trial	Precision	Recall	FPr
5	.598	.965	.397
7	.897	.951	.121
8	.932	.956	.040
9	.954	.929	.043
10	.727	.464	.028
11	.890	.968	.135
14	.975	.758	.014
21	.927	.983	.090
22	.930	.961	.068
23	.818	.902	.075
24	.880	.939	.170
Average	.866	.889	.107
Average w/o trial 10	.880	.931	.115
Expected performance	.786	.916	.212

Table 6.6: Observed performance vs expected performance. The observed performance is determined by the number of false positive and false negative segments found by a human annotator. A separate average without trial 10 is also provided since it was an outlier condition. The expected performance is culled from the average machine performance tables 4.5, 4.6 and 4.7. This table shows that the performance metrics provide a good prediction of the observed segmenter performance, as reported by human annotators.

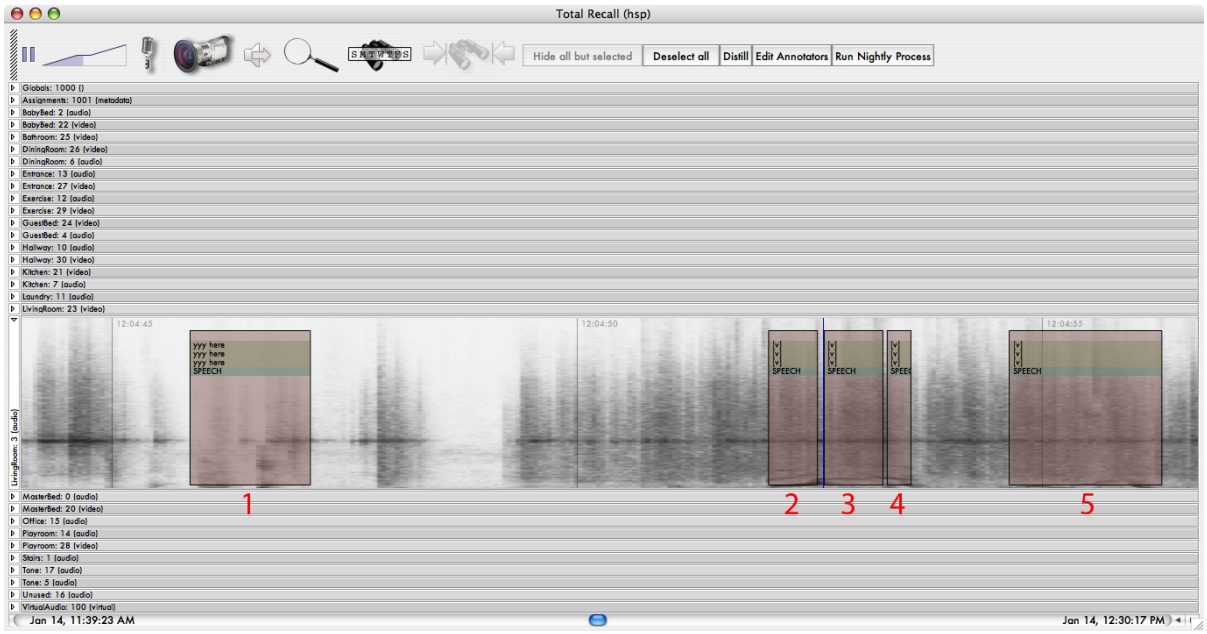


Figure 6-7: Trial 10 spectrogram. Segments 2 and 4 were automatically identified, the rest were identified manually.

segments that the human annotator created in TotalRecall. Like the performance metrics, the observational error rates are calculated based on the time spans of segments, and not the segment counts. Because the observed error does not treat clipping, it should overestimate the recall. By not treating segments which are too long, it should underestimate the FPr. This is reflected by the data in the table.

6.4.2 Segmenter Parameter Selection

The smoother and segmenter component operates on the output of the frame classifier. A good frame classifier is critical to the final segmenter performance, but the smoother and segmenter can be tuned to produce a tradeoff between error types.

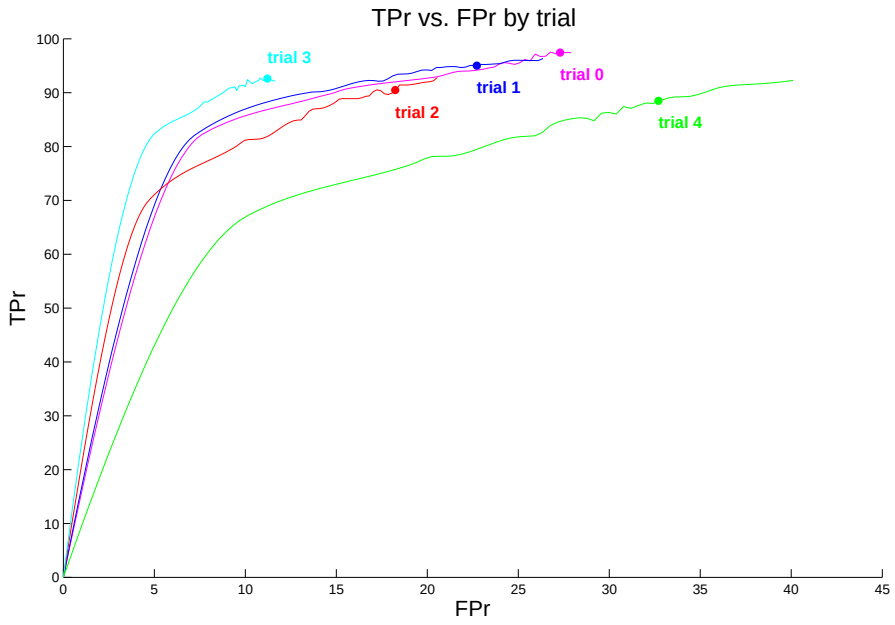
One approach to tuning parameters is a simple search over parameter space. For each setting of parameters, the performance is evaluated against a set of human segmentations. These are the four human segmentation trials from the original machine performance evaluations in section 4.3.2, as well as an additional segmentation trial (trial 0). The result of these

evaluations are speech detection performance curves, both for each trial individually and for the average performance. Figures 6-8 and 6-9 show performance plots for TPr relative to FPr and precision relative to recall, illustrating the performance tradeoffs that can be obtained with different parameter settings. These smooth curves interpolate between the actual performance points at each parameter setting; the actual points are left out for clarity. In each of these curves, the solid dot represents the performance for the parameter setting used by the speech detector for this thesis.

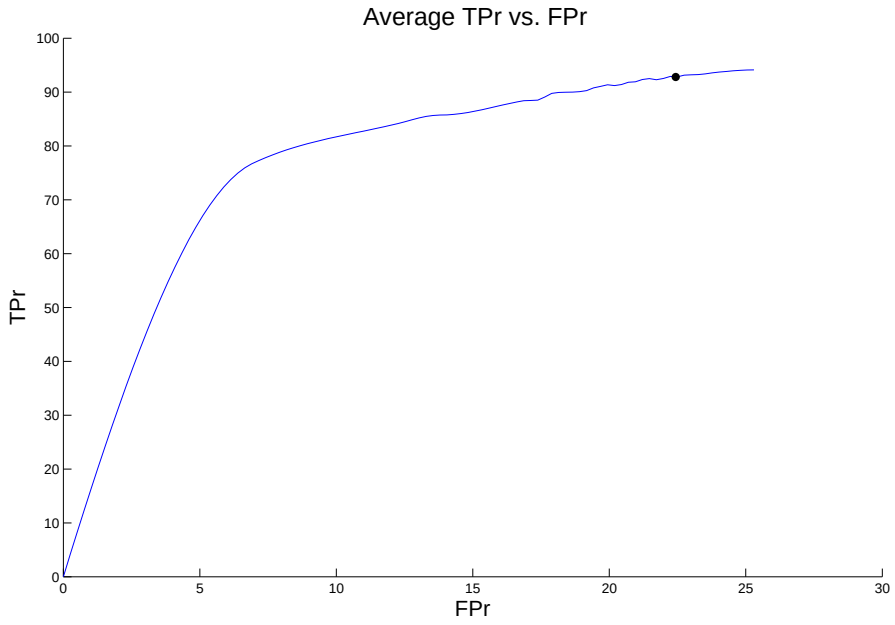
Section 6.3.4 determined a correspondence, in terms of human effort, between false positives and false negatives. False negatives were found to be a factor of 18 times more labor intensive than false positives. That means that marking 18 false positives should take about as much human annotator time as marking a single false negative. For a given FPr and FNr, the human effort expended on *errors* can be quantified by

$$\text{effort} = FPr + k\ FNr$$

where k is the cost ratio of false negatives to false positives, estimated to be 18. This equation can be used to tune the segmenter to produce segmentations that minimize the human effort required for transcription. Of course, the basic transcription difficulty remains, but the additional work of identifying and marking errors can be minimized. Equation 6.4.2 can be plotted at various settings of FPr and FNr, for various choices of k. Taking the speech detector performance curves in figure 6-8, and computing the expected effort at each segmenter parameter setting and five different settings of k, yields the plots in figures 6-10, 6-11, and 6-12. The actual effort curves plotted have been squashed with a square root, and scaled to fit on the plot, so only the shape and minimum points should be considered. In fact, the actual effort required for performance points on the left side of the plots would be much more than what is displayed. Each effort curve shows the point of minimum effort as a filled-in black circle. The six curves in figures 6-10, 6-11 and 6-12 are for the five unique trials (trials 0-4) and a sixth average curve. The vertical dotted line in each plot is at the FPr of the actual speech detector used, to show where it intersects the five effort curves and the actual performance curve. Notice that, in general, the minimum expected effort for

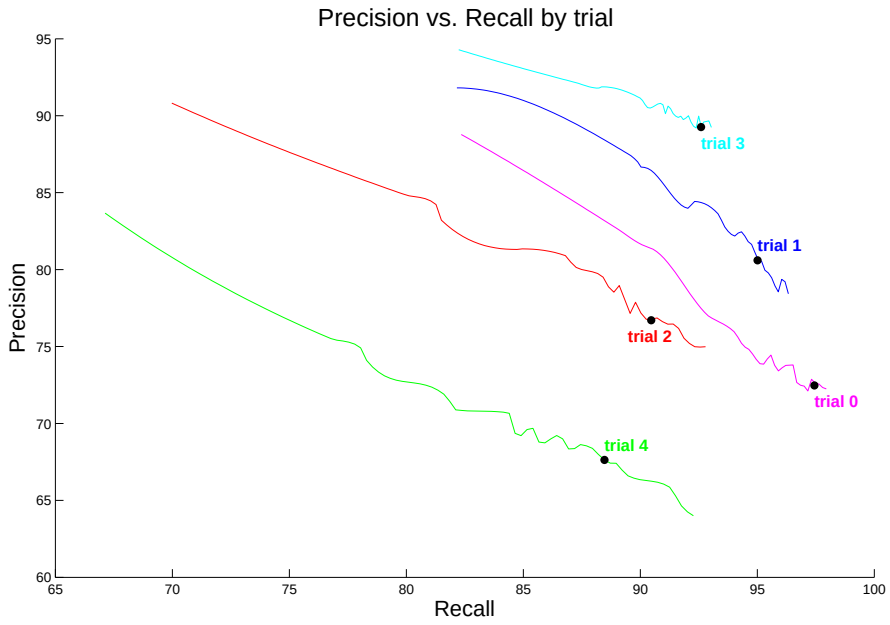


(a) TPr vs. FPr by trial.

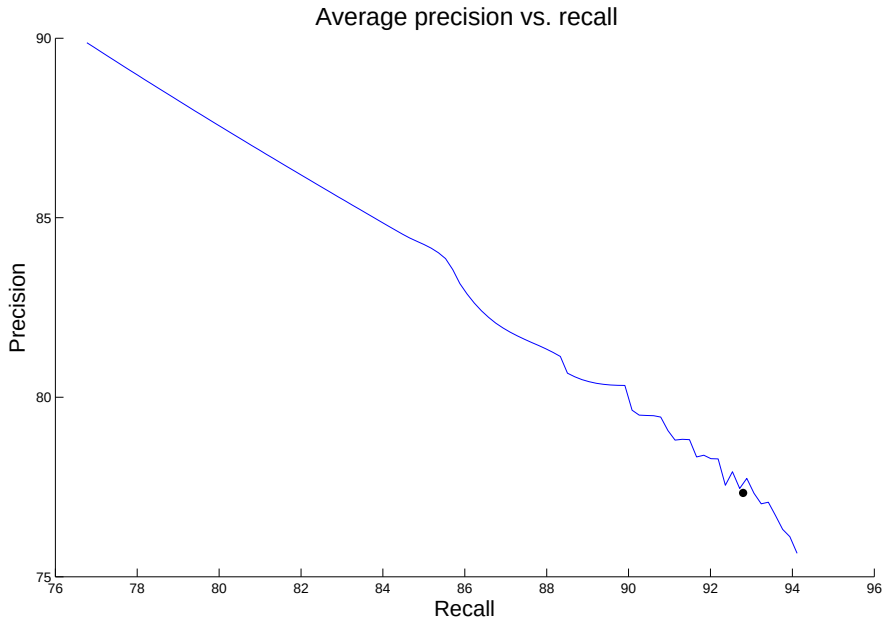


(b) Average TPr vs. FPr

Figure 6-8: Segmenter TPr vs. FPr. In each, the filled black dot indicates the parameter setting used by the speech detector in this thesis. Moving to the right on the performance curve increases the false positive rate, simultaneously increasing the true positive rate.



(a) Precision vs. recall by trial



(b) Average precision vs. recall

Figure 6-9: Segmenter precision vs. recall. In each, the filled black dot indicates the parameter setting used by the classifier. Parameter settings which increase recall tend to reduce precision.

the maximum $k = 16$ is at the rightmost point on the performance curve. For trial 3, it is very close to the rightmost point though not quite at the edge. The actual speech detector used, indicated by the vertical dotted lines, may be close to the right tradeoff between false positive and false negative rates, but is still too conservative and could be improved by picking parameters with a higher FPr. Choosing more aggressive parameters is done by finding the parameter settings for a particular performance point.

The segmenter algorithm uses the classifications and confidences returned by the frame-level classifier. However, since the frame-level classifier is based on a decision tree, the confidence values do not vary smoothly and are less meaningful than for other classifiers. The effect of this is that varying the segmenter parameters, which pay attention to both classifications and confidence values, does not smoothly change the segmenter behavior, but instead produces more discrete "jumps" in performance. This accounts for the limited performance range of the segmenter. A segmenter with a wider, more tunable performance range might be obtained by modifying the frame-level classifier to produce better confidence estimates. Section 4.2 pointed out some of these issues as well as references on ways to address this problem.

To repeat a point made earlier, finding false negatives may not be the best use of a human transcriber's time. Obviously, if the transcription task does not include finding false negatives, then the factor k decreases toward zero. However, one must also remember the scientific goal, which would suffer due to missed speech. Thus, false negatives may still be considered a more expensive error than false positives.

6.5 Discussion

This chapter presented a number of evaluations of the system. First, human performance was evaluated on the transcription task using various tools, including the new tools for the system. This helped validate the approach, and demonstrated that it is at least 2.5 to 6 times faster than other systems. The evaluation then focused on specific aspects of the semi-automatic transcription process, in order to better understand the human performance

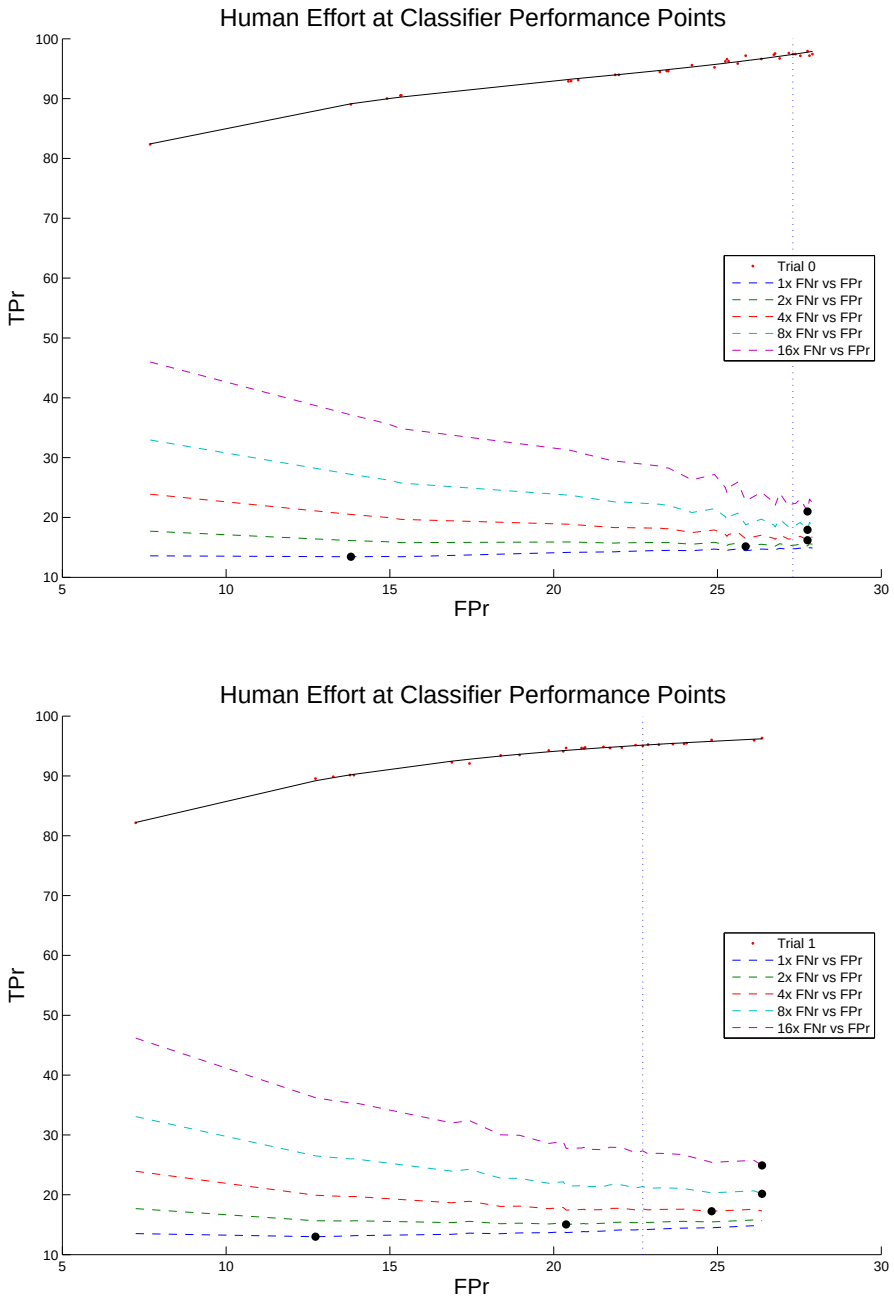


Figure 6-10: Expected human effort at various segmenter performance points. The performance curve is shown, along with five curves representing the expected human effort assuming false negatives require 1x, 2x, 4x, 8x, and 16x more effort than false positives. The filled in black circle indicates the point on the curve of minimum effort. The vertical dotted line is at the FPr of the actual speech detector used in this thesis.

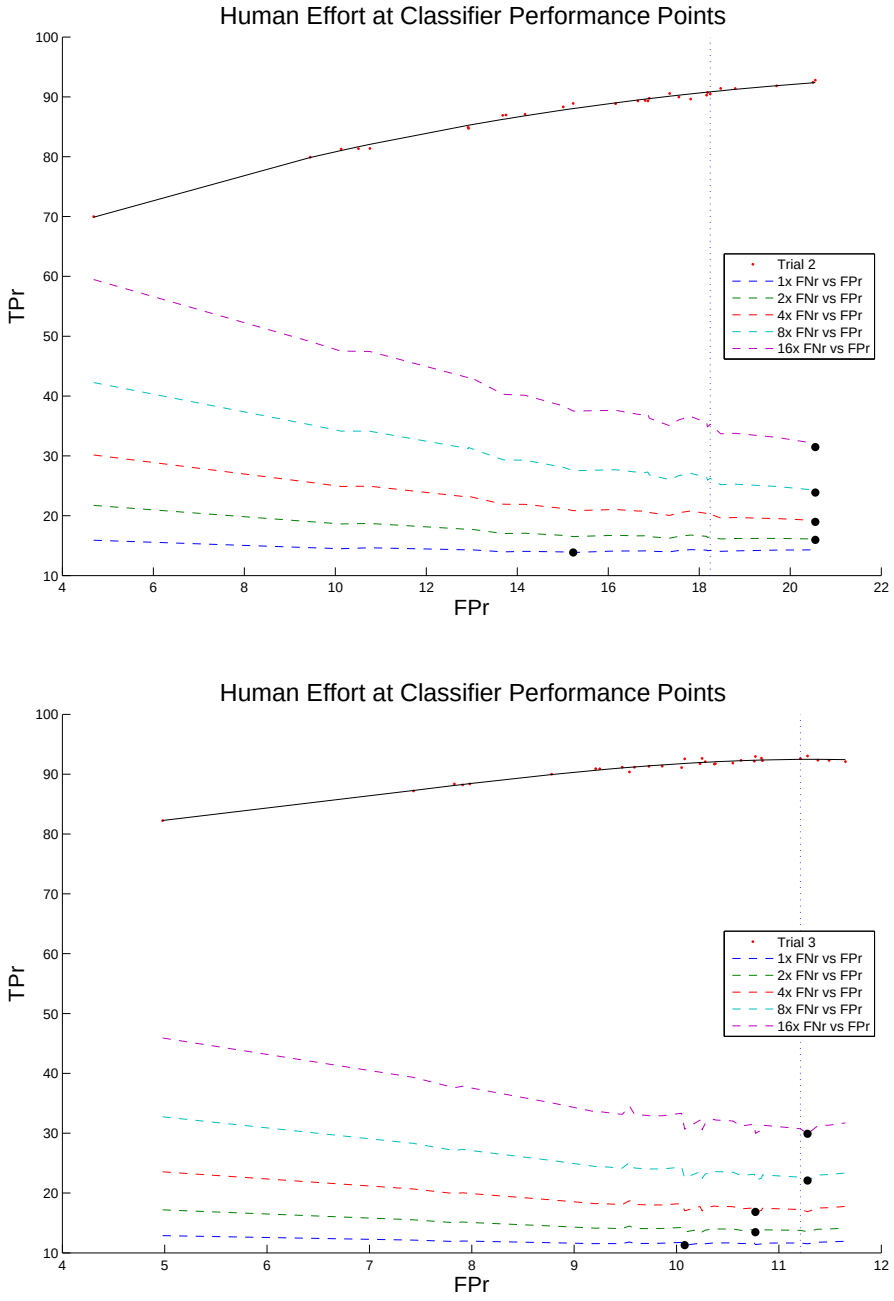


Figure 6-11: Expected human effort at various segmenter performance points, part 2.

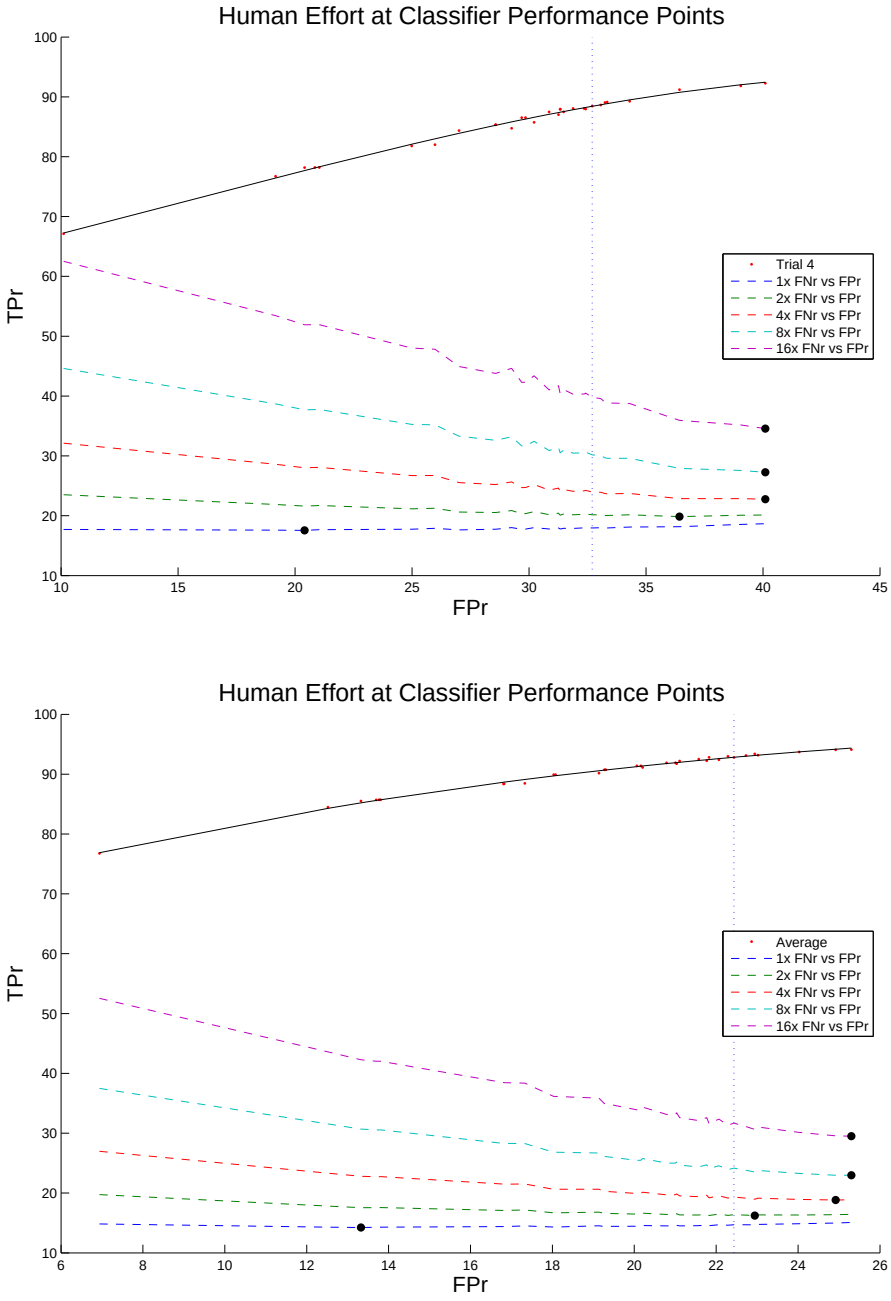


Figure 6-12: Expected human effort at various segmenter performance points, part 3.

factors. First, a simple linear model was developed that related the transcription time to the amount of detected speech from the automatic processes. This provides a way of estimating how much annotator effort will be needed to transcribe the speech for the Human Speechome Project. The analysis then focused on two primary error types resulting from the automatic speech detector – false positives and false negatives. A model for the time required to handle each error type was developed, in order to estimate the relative costs of each. Processing false negatives seemed to exhibit an interesting power law, while false positives followed a linear model. While the purpose of this analysis was not to find a predictive model, the task time could potentially be roughly predicted if certain properties of the speech segmenter were known, such as the false positive and false negative rate. The relative cost of false positives to false negatives was used to tune the automatic segmenter. The automatic segmenter was evaluated using the performance metrics developed in Chapter 4. In order to be sure the performance metrics actually reflected the performance experienced by a human annotator, the "predicted" performance was compared to the observed performance. By viewing the amount of human effort in terms of the automatic speech detector performance, the two halves of the system become tightly linked. Putting all the pieces of the analysis together, the automatic system can be tuned in a principled way, and informed decisions for future work can be made.

Chapter 7

Conclusions and Future Work

This thesis presented a new system for speech transcription that combines human annotation with machine processing. The system enables rapid transcription of a massive, multitrack audio/video corpus collected for a longitudinal study of human language acquisition. Speech transcription is not only relevant to the language acquisition community, it is also critical to the entertainment and medical transcription industries. Speech transcripts are also proving to be extremely useful for video search applications. In general, massive multimedia datasets depend on comprehensive, useful annotations such as speech transcripts to enable efficient browsing, search and analysis. These are the semantic units that a human can use for expressing a search query, or for interpreting the information in a dataset more easily. However, many annotation tasks still require human intervention or oversight to produce. Thus, as datasets grow due to advances in collection and storage technologies, so does the amount of human labor needed to work with them.

7.1 Thesis Summary

The system described in this thesis is a semi-automatic system for speech transcription, built primarily as part of the Human Speechome Project. The Human Speechome Project is a new study of human language acquisition, obtaining the most comprehensive ever

record of all auditory and visual events in the life of a child, from birth to age three. These dense recordings capture a significant portion of the child's experience, including not only linguistic but also contextual information. Speech transcripts are needed as a first step to analysis, which can later be combined with other information for more elaborate studies. For example, each occurrence of a word in the corpus may be aligned with the corresponding video to study word usage in context.

While tools for speech transcription do exist, it would take a prohibitive amount of human annotator time and effort to transcribe the thousands of hours of speech in the HSP corpus. Instead, the system first applies automatic methods to find and segment speech in the multi-track audio data, and then presents these speech segments to a human transcriber for transcription. By using automatic processes to perform the tedious task of locating and segmenting speech, the human annotator can focus on transcription, a task currently out of reach of even the best speech recognizers. In addition, human judgment and understanding are needed to transcribe much of the speech in the HSP corpus, since a significant portion includes baby babble and early word usage. Not only does the automatic processing reduce the total amount of data a human must work with, it also simplifies the annotation tools. This reduces the cognitive load on the annotator and makes transcription more efficient.

Because the system is prone to errors, it is important to understand the impact these errors have on transcription. By analyzing and modeling the transcription task, appropriate design choices can be made. In particular, the automatic component can be tuned to maximize the productivity of the human annotator. The motivating philosophy is that, through a harmonious collaboration of human and machine, the resulting system will be more effective than either by itself.

7.2 Future Work

There are many exciting directions to pursue as part of this work, too many to discuss in detail, but a summary will give a sense of some possibilities. A project which is actually

quite far along, though not treated here, is that of speaker identification. As part of automatically detecting and segmenting speech, determining the speaker identity can be done automatically. Gish and Schmidt [21] provide an overview of speaker identification, which has many similarities to speech detection. Once speaker identification is fully implemented, it may be desirable to further split speech segments by speaker. Currently, TotalRecall allows an annotator to view and change the annotation produced by the speaker identifier, but this has not been integrated into BlitZcribe. Incorporating speaker identity into BlitZcribe will likely slow the transcription process somewhat, but the benefit will be richer metadata. As BlitZcribe has demonstrated, if the additional functionality is incorporated in a streamlined, simple way, it may not require much additional annotator effort to mark and correct speaker identification errors.

Another component which has been developed, but not yet integrated into BlitZcribe, is a method for playing audio more rapidly without distorting the pitch. Applying the SOLAFS algorithm [56] [24] to speed up the audio by a reasonable factor would allow the annotator to hear the speech in a shorter amount of time, which should translate to faster transcription times. Of course, if sped up too much, it may be harder to understand, so finding the right factor is important. It is even conceivable that *slowing* the audio could improve transcription times, since it may reduce the number of times the transcriber needs to repeat a segment. Another interesting suggestion for optimizing the audio for the annotation task is to use stereo playback. This may help spatially locate different speakers, and should be possible since the locations of the microphones in the house are known.

There are some interesting machine learning and signal processing challenges specific to auditory processing raised by the Human Speechome Project. First, the speech detection algorithm could potentially be improved by taking better account of the inter-frame dynamics. As a temporal process, there is a strong dependence between a feature vector at time t and time $t+1$. A spectrogram clearly reveals the structure of different sounds, including speech, in both the frequency and time domains. Better modeling of this structure may lead to better sound classifiers. Another interesting property of sound is the fact that it is additive – two sounds occurring at the same time mix together. Humans are adept at separating

a sound back into its components, or paying attention to a single sound of interest, yet the current discriminative classifiers are not designed to handle these mixtures. Techniques for handling sound mixtures [18] and auditory source separation [25] may help build a better speech detector and other sound classifiers. One observation about the training process that has led to better frame classifier models is that the training data is not completely trustworthy – indeed, there are many "label errors", when the audio is considered at the frame level. Label errors occur when a segment of speech in the training set contains audio frames that are identical to silence or noise. For example, there may be silences in the middle, or at the ends of a speech segment. In training, all frames will be labeled as speech, which makes the training task more difficult. Initial experiments identifying and relabeling these frames yielded promising improvements in the frame-level classifier. Mislabeled training data is further discussed in [9] and [2].

Another exciting direction for research is how the audio and video modalities, as well as other contextual information, such as the time of day, room location, etc., might combine to identify activities, objects, and events of interest. Current work on the video data is focused on person tracking and determining gaze direction. This, combined with speech detection, might form a basis for associating words with objects, or even discovering the words themselves.

7.3 Implications and Contributions

At the time of this writing, more than two years of data has been collected for the Human Speechome Project. Recording from fourteen microphones for an average of ten hours per day, for about 350 days per year for two years amounts to an estimated 100,000 hours of recorded audio. At an average of four to five hours of speech per day, the data will contain about 3000 hours of speech to transcribe.

To get a sense for the size of this transcription task, consider the following. If all the multitrack audio collected over the course of two years could be collapsed into a single audio track, there would be about seven thousand hours of audio to transcribe. Using

CLAN and Transcriber, this would take anywhere between 42,000 and 67,000 hours of human transcriber effort. At the current transcriber hourly rate of about $10 per hour, this approaches or exceeds one half of a million dollars. If the task is split among ten transcribers, this should take around three years to complete, optimistically assuming transcribers work at average efficiency for an eight hour day. It is not even worth considering directly applying CLAN or Transcriber to the 100,000 hours of audio collected in the two year period.

Unlike CLAN and Transcriber, the semi-automatic system is designed to easily support dense, multitrack audio. Using this system in safe mode, it should take about 16,000 hours of labor, and cost about $160,000. This should take ten transcribers less than a year to complete. If the system is used in fast mode, the numbers decrease further, requiring just under 10,000 hours for 3000 hours of speech. This means the cost of transcription may be brought down to under $100,000, and will take ten transcribers about half a year. With the new system, transcription will complete faster and at a fraction of the cost than with either of two popular, existing tools.

While fast mode transcription is the fastest, is it the right solution for the Human Speechome Project? Because some speech may be missed by the automatic speech detector, fast mode will reduce the coverage of the speech transcriptions, as compared with safe mode or manual approaches. However, there is already a certain amount of child relevant speech that is naturally missed as part of collecting the HSP corpus. For example, the corpus does not capture speech activity when the child is outside, or the family is on vacation. A current estimate is that about 70% of all child relevant activity is recorded. So while this is the most comprehensive child language acquisition corpus to date, it won't capture everything. Therefore, the current speech detector, with a false negative rate of about 7%, will reduce the total speech coverage by about 5%. For the cost and time savings between safe mode and fast mode, this amount of additional missed speech may be tolerated. The nature of such dense, longitudinal studies is that by capturing a comprehensive dataset, one may be able to afford not annotating some of the data in the corpus.

The speech transcription system described here has been motivated by the needs of the Human Speechome Project. However, the tools and methodology should also be useful to

others transcribing speech. Because manual transcription is so time consuming, both smaller and larger projects will benefit from faster transcription. Looking ahead, we hope that this system will encourage other large-scale studies akin to the Human Speechome Project. Whether one wants to study how humans learn from and interact with their environment, or build machines that mimic these abilities, collecting and analyzing rich datasets will play an important role. In concert with new data collection and storage technologies, better tools for annotating and analyzing large scale datasets will surely create exciting new research opportunities.

Bibliography

[1] Isabelle Alvarez, Stephan Bernard, and Guillaume Deffuant. Keep the decision tree and estimate the class probabilities using its decision boundary. In Manuela M. Veloso, editor, *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 654–659, 2007.

[2] Dana Angluin and Philip Laird. Learning from noisy examples. *Machine Learning*, 2:343, 1987.

[3] Claude Barras, Edouard Geoffrois, Zhibiao Wu, and Mark Liberman. Transcriber: development and use of a tool for assisting speech corpora production. *Speech Communication*, 33(1-2):5–22, January 2001.

[4] Steven Bird and Mark Liberman. A formal framework for linguistic annotation. *Speech Communication*, 33(1-2):23–60, 2001.

[5] Steven Bird, Kazuaki Maeda, Xiaoyi Ma, Haejoong Lee, Beth Randall, and Salim Zayat. Tabletrans, multitrans, intertrans and treetrans: Diverse tools built on the annotation graph toolkit. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, April 2002.

[6] Paul Boersma. Praat, a system for doing phonetics by computer. *Glot International*, 5(9/10):341–345, 2001.

[7] Albert S. Bregman. *Auditory Scene Analysis: The Perceptual Organization of Sound*. The MIT Press, September 1994.

[8] C. Brian, N. Sawhney, and A. Pentland. Auditory context awareness via wearable computing, 1998.

[9] C. E. Brodley and M. A. Friedl. Identifying mislabeled training data. *Journal of Artificial Intelligence Research*, 11:131–167, 1999.

[10] Jerome Bruner. *Child's Talk: Learning to Use Language*. W.W. Norton, 1983.

[11] Y. Chow, M. Dunham, O. Kimball, M. Krasner, G. Kubala, J. Makhoul, P. Price, S. Roucos, and R. Schwartz. Byblos: The bbn continuous speech recognition system. In *Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP '87.*, volume 12, pages 89–92, 1987.

[12] Thomas M. Cover and Joy A. Thomas. *Elements of information theory*. Wiley-Interscience, New York, NY, USA, 1991.

[13] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *Acoustics, Speech, and Signal Processing [see also IEEE Transactions on Signal Processing], IEEE Transactions on*, 28(4):357–366, 1980.

[14] Philip DeCamp. Using head pose estimation to efficiently annotate large video corpora. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, August 2007.

[15] Li Deng and Xuedong Huang. Challenges in adopting speech recognition. *Commun. ACM*, 47(1):69–75, 2004.

[16] Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

[17] L. Dybkjær and N. O. Bernsen. Towards general-purpose annotation tools—how far are we today? In *Proceedings of the Fourth International Conference on Language Resources and Evaluation LREC'2004*, volume I, pages 197–200, Lisbon, Portugal, May 2004.

[18] Daniel P. W. Ellis. Prediction driven computational auditory scene analysis for dense sound mixtures. In *ESCA workshop on the Auditory Basis of Speech Perception*, Keele, UK, 1996.

[19] Matteo Frigo and Steven G. Johnson. The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231, 2005. special issue on "Program Generation, Optimization, and Platform Adaptation".

[20] Jim Gemmell, Gordon Bell, and Roger Lueder. Mylifebits: a personal database for everything. *Commun. ACM*, 49(1):88–95, 2006.

[21] Herbert Gish and Michael Schmidt. Text-independent speaker identification. *IEEE Signal Processing Magazine*, October 1994.

[22] Michael M. Goodwin and Jean Laroche. Audio segmentation by feature-space clustering using linear discriminant analysis and dynamic programming. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, October 2003.

[23] A. L. Gorin, G. Riccardi, and J. H. Wright. How may i help you? *Speech Communication*, 23(1/2):113–127, 1997.

[24] D. Henja and B.R. Musicus. The solafs time-scale modification algorithm. Technical report, BBN, July 1991.

[25] A. Hyvärinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13(4-5):411–430, 2000.

[26] N. Jojic, B. J. Frey, and A. Kannan. Epitomic analysis of appearance and shape. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 34–41 vol.1, 2003.

[27] A. Kapoor and S. Basu. The audio epitome: a new representation for modeling and classifying auditory phenomena. In *Acoustics, Speech, and Signal Processing, 2005. Proceedings. (ICASSP '05). IEEE International Conference on*, volume 5, pages v/189–v/192 Vol. 5, 2005.

[28] Rony Kubat, Philip DeCamp, Brandon Roy, and Deb Roy. Totalrecall: Visualization and semi-automatic annotation of very large audio-visual corpora. In *ICMI*, 2007.

[29] J. C. R. Licklider. Man-computer symbiosis. *IRE Transactions on Human Factors in Electronics*, HFE-1:4–11, March 1960.

[30] Brian MacWhinney. *The CHILDES Project: Tools for Analyzing Talk*. Lawrence Erlbaum Associates, Mahwah, NJ, 3^{rd} edition, 2000.

[31] Kazuaki Maeda, Steven Bird, Xiaoyi Ma, and Haejoong Lee. Creating annotation tools with the annotation graph toolkit. In *Proceedings of the Third International Conference on Language Resources and Evaluation*, Apr 2002.

[32] M. Mitzenmacher. A brief history of generative models for power law and lognormal distributions. *Internet Mathematics*, 1(2):226–251, 2004.

[33] S. Monsell. Task switching. *Trends in Cognitive Science*, 7(3):134–140, March 2003.

[34] Joakim Nivre, Jens Allwood, Jenny Holm, Dario Lopez-Ksten, Kristina Tullgren, Elisabeth Ahlsen, Leif Gronqvist, and Sylvana Sofkova. Towards multimodal spoken language corpora: Transtool and synctool. In *Proceedings of the Workshop on Partially Automated Techniques for Transcribing Naturally Occurring Speech*, Montreal, Canada, August 1998. COLING-ACL.

[35] Sharon Oviatt. Human-centered design meets cognitive load theory: designing interfaces that help people think. In *MULTIMEDIA '06: Proceedings of the 14th annual ACM international conference on Multimedia*, pages 871–880, New York, NY, USA, 2006. ACM Press.

[36] Alex Park and James Glass. Unsupervised word acquisition from speech using pattern discovery. In *International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France, 2006.

[37] Vesa Peltonen, Juha Tuomi, Anssi Klapuri, Jyri Huopaniemi, and Timo Sorsa. Computational auditory scene recognition. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 2, pages 1941–1944, May 2002.

[38] Foster Provost and Pedro Domingos. Tree induction for probability-based ranking. *Machine Learning*, 52(3):199–215, 2003.

[39] J. Ross Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[40] J. Ross Quinlan. Bagging, boosting, and c4.5. In *AAAI/IAAI, Vol. 1*, pages 725–730, 1996.

[41] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[42] Dennis Reidsma, D. H. W. Hofs, and N. Jovanovic. Designing focused and efficient annotation tools. In L. P. J. J. Noldus, F. Grieco, L. W. S. Loijens, and Patrick H. Zimmerman, editors, *Symp. on Annotating and measuring meeting behavior, MB2005*, pages 149–152, Wageningen, NL, September 2005.

[43] Dennis Reidsma, Natasa Jovanović, and Dennis Hofs. Designing annotation tools based on properties of annotation problems. In *Measuring Behavior 2005, 5th International Conference on Methods and Techniques in Behavioral Research*, 30 August - 2 September 2005 2005.

[44] R. Travis Rose, Francis Quek, and Yang Shi. Macvissta: a system for multimodal analysis. In *ICMI '04: Proceedings of the 6th international conference on Multimodal interfaces*, pages 259–264, New York, NY, USA, 2004. ACM Press.

[45] Deb Roy and Niloy Mukherjee. Towards situated speech understanding: visual context priming of language models. *Computer Speech & Language*, 19(2):227–248, April 2005.

[46] Deb Roy, Rupal Patel, Philip DeCamp, Rony Kubat, Michael Fleischman, Brandon Roy, Nikolaos Mavridis, Stefanie Tellex, Alexia Salata, Jethran Guinness, Michael Levit, and Peter Gorniak. The human speechome project. In *Proceedings of the 28th Annual Cognitive Science Conference.*, pages 2059–2064, 2006.

[47] Robert E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[48] Manfred Schroeder. *Fractals, Chaos, Power Laws: Minutes from an Infinite Paradise*. W. H. Freeman, July 1992.

[49] M. A. Siegler, U. Jain, B. Raj, and R. M. Stern. Automatic segmentation, classification and clustering of broadcast news audio. In *Proceedings of the Ninth Spoken Language Systems Technology Workshop*, Harriman, NY, 1996.

[50] K. Sjölander and J. Beskow. Wavesurfer - an open source speech tool. In *Proc. of ICSLP*, volume 4, pages 464–467, Beijing, Oct. 16-20 2000.

[51] Paris Smaragdis. Discovering auditory objects through non-negativity constraints. In *Statistical and Perceptual Audio Processing (SAPA)*, October 2004.

[52] Paris Smaragdis and Michael Casey. Audio/visual independent components. In *4th International Symposium on Independent Component Analysis and Blind Signal Separation*, Nara, Japan, April 2003.

[53] Michael Tomasello and Daniel Stahl. Sampling children's spontaneous speech: how much is enough? *Journal of Child Language*, 31(1):101–121, February 2004.

[54] Norbert Tóth and Béla Pataki. On classification confidence and ranking using decision trees. In *Proceedings of the 11th International Conference on Intelligent Engineering Systems (INES)*, Budapest, Hungary, 2007.

[55] George Tzanetakis and Perry Cook. Multifeature audio segmentation for browsing and annotation. In *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, October 1999.

[56] Sunil Vemuri, Philip DeCamp, Walter Bender, and Chris Schmandt. Improving speech playback using time-compression and speech recognition. In *CHI '04: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 295–302, New York, NY, USA, 2004. ACM Press.

[57] Willie Walker, Paul Lamere, Philip Kwok, Bhiksha Raj, Rita Singh, Evandro Gouvea, Peter Wolf, and Joe Woelfel. Sphinx-4: A flexible open source framework for speech recognition. Technical Report 139, Sun Microsystems, November 2004.

[58] Chip Walter. Kryder's law. *Scientific American*, August 2005.

[59] Geoffrey I. Webb. Multiboosting: A technique for combining boosting and wagging. *Machine Learning*, Vol.40(No.2), 2000.

[60] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems).* Morgan Kaufmann, June 2005.

[61] Norimasa Yoshida. Automatic utterance segmentation in spontaneous speech. Master's thesis, Massachusetts Institute of Technology, Septmber 2002.