2023

Use of Speech Recognition Tools to Predict Mistakes in Audio Book Recordings

Micah Tietz

Use of Speech Recognition Tools to Predict Mistakes in Audio Book Recordings

by

Micah Tietz

A Thesis Submitted in Partial Fulfillment of the

Requirements for the Degree of

Bachelor of Science

in

Cognitive Science

with an emphasis in Computer Science

Minnesota State University, Mankato

Mankato, Minnesota

July 5, 2023

This thesis paper has been examined and approved.

Examining Committee:

Dr. Rebecca Bates, Chairperson

Dr. Rushit Dave

Dr. Richard Liebendorfer

Abstract

This thesis documents the feasibility of creating an automated system to identify mistakes and errors in audiobook performances, with the intent of helping to speed up and streamline the otherwise-lengthy process of audiobook production and editing. The primary method being explored is to compare the intended script with the audio recording of the performance, as evaluated and transcribed by an automatic speech recognition (ASR) system, and to search for discrepancies between the two. Other topics examined include the background and history of audiobooks and ASR systems, the difficulties of creating a bespoke machine-learning system for a specific performer, the strengths and limitations of freely-available tools such as Kaldi and Aeneas, as well as the possibility of using statistical and mathematical techniques to try to identify the patterns of certain unwanted artifacts within the performance. The results of the experiments conducted were promising, but fell short of full success in several ways. Most importantly, the Kaldi-based system did not achieve error recognition with a sufficiently high degree of confidence or consistency when using a speaker-independent model. Additionally, the designed purpose of many ASR systems are at odds with this project's intent, in that for most applications, they are built to accommodate errors and ignore background noise. This work demonstrates that there is potential to improve the performance and accuracy of existing ASR tools for error detection.

Table of Contents

Table of Figures

List of Tables

Chapter 1

Introduction

In the audiobook industry, the need to manually review and edit each recorded audiobook before publishing has been a consistently and unavoidably time-consuming part of the production process. This is due to the reality that until very recently, the focused attention of a human listener has been the only way to meaningfully detect and correct any mistakes.

The core question of this thesis is to determine whether or not it is viable to create an error-detection system for audiobooks and other speech performances using freely-available open-source automatic speech recognition (ASR) tools. If possible, such a system could have benefits and potential applications in several industries, as an automated system is not subject to the same limitations of a human listener, such as the gradual lapse of attention and focus when editing a long audio recording. While the benefits of a software approach to analyzing the accuracy of audio performances may be applied to many possible fields, such as voice acting or training for public speaking, this thesis is primarily concerned with the applicability of these techniques to audiobook production.

Speech recognition is a rapidly-maturing field, both as a technology and methodology. Applications for ASR systems range from automated processing of speech samples for the purpose of data mining and analysis to hands-free interfaces between humans and computers that are more reminiscent of a casual conversation than of operating a machine through traditional interfaces such as graphical or command-line input. Whether in automatic call systems, forensic speech analysis, or asking a device for directions in an unfamiliar city, speech processing technologies have found many practical applications, and are constantly growing in sophistication and scale.

One potential use of speech processing that has remained relatively unexplored is in the evaluation of audio performances. Voice acting and narration require a significant time investment not only to create a raw recording, but that recording must then be listened to by a human being to discover any errors so they can be corrected. These may be things such as pronunciation mistakes, unwanted noises, gaps that are too long or too short, and sections in which the performer's voice sounds significantly different from the rest of the performance, just to name a few.

This project explores the use of speech recognition to examine and critique audio performances in the specific context of audiobook narration, and whether or not there exists a reasonable possibility of using currently-existing and freely-available tools to reduce the amount of time and effort required to properly edit a raw audiobook recording for publication and distribution. Audiobooks as a medium are particularly suitable for testing speech recognition, due to the implicit presence of a precise written script of what is being performed, and the strong incentive to keep the performance accurate to the source. This allows a labeled dataset to be created with less difficulty than many alternative data sources, which may need to be transcribed by hand.

Also of interest to this project are how speech recognition tools interact with existing toolsets already used during the editing process. Unlike the current state of speech recognition, which relies heavily on machine learning and statistical analysis, most legacy editing tools are purely algorithmic in nature. Some common tools include high- and low-pass filters, unwanted noise removal through filtering, de-essing, and amplitude normalization.

In addition to examining whether speech recognition can reliably locate unwanted discrepancies in audiobook narration, it was also tested to what extent the traditional audio processing techniques used for audiobooks influenced the accuracy and efficacy of speech recognition processes.

To shortly summarize the outcome, there were some promising developments in applying free, open-source speech recognition to audiobook samples. However, in the absence of sufficiently large datasets to generate custom speaker-dependent speech models, the use of generic speech models is not particularly accurate and produces numerous false-positives. The use of filters and pre-processing, however, was found not to have a statistically significant impact on how well the system was able to comprehend the samples it was provided with.

Further details about the subject of audiobook recording and speech transcription can be found in subsequent sections of this thesis. Also described are the tools, methods, and results of the project, as well as the conclusions based on those results.

Chapter 2

Background

In order to fully understand the tools, process and subject of this project, it is helpful to have some familiarity with audiobooks, audio recording technology, and speech recognition. This section explains and summarizes information about the history of audiobooks, the development and components of speech recognition, and utilities such as PRAAT [4], Kaldi [13], and Aeneas[11].

2.1 History of Audiobooks

Audiobooks themselves date back to the 1930s, as the capacity of early audio-recording technology had improved to a point that it became practical to record at least significant portions of a book on a close-grooved record. These early “talking books” were originally produced by the American Foundation for the Blind and the Library of Congress as a way to make materials more accessible to blind and reading-impaired audiences, to name just two of the many organizations that became involved providing in such services [15].

For the first decades of their use, talking books were produced to meet a very utilitarian need. Limited by the relatively low capacity, inconvenience and non-portability of the media upon which they were recorded, the development and distribution of book recordings was largely a non-profit undertaking aimed directly at people for whom reading a physical book was not an option. Audiobooks were more likely to be found in libraries and other public institutions than in private ownership.

The blind and reading-impaired community remained the only substantial market for recorded books until the establishment of the first commercial audiobook seller, Caedmon Records, in 1952. Since then, the market has continued to grow and benefit from changes in technology and available media, including cassette tapes and compact discs. The emergence of digital media players and internet-based audiobook services in the late 1990s and early 2000s were a particular boon for the industry.

In addition to audio storage, developments in audio engineering have also benefited the production side of the industry. Inexpensive recording equipment and software tools have lowered the barrier to entry for individuals to start producing recordings themselves, allowing for studio-quality results at far less cost. The result of all of these developments have resulted in substantial growth of the audiobook publishing sector. As of 2020, audiobooks have become a billion-dollar industry in the United States alone, and will likely continue to grow for the foreseeable future. [17]

2.2 History of Speech Recognition

Contemporary with the founding of the first commercial audiobook producer, the first speech recognition system was demonstrated in 1952. At Bell Labs, Stephen Balashek, R. Biddulph, and K. H. Davis developed ‘Audrey’, a system that could identify digits uttered by a speaker. This system used direct analysis of the sound to detect the formants in the speaker’s utterances and was speaker-dependent. It could work fairly consistently when spoken to by the system's creator and other designated speakers, but simply did not function at all for any voice or dialect that was not a close match to the patterns it was built to detect.

Much early work in speech recognition took place during the 1950s and 1960s, alongside development in artificial intelligence (AI). Unsurprisingly, one of the most popular ambitions in both fields was to be able to create an interface that allowed a person to interact with a computer in the same way they would hold a conversation with another person. Ideally, the machine would understand what it was told, process the request, and provide corresponding output, either as a spoken reply, or text on a display. Of all of these goals, it turned out to be that speech synthesis, or the production of an artificial voice given a textual source, was the only goal that could be reasonably accomplished within the technological constraints of the time. Even then, the quality of voice synthesis was quite poor and obviously unnatural.

The very first speech recognition systems were fundamentally rule-based and relied upon a hard-coded dictionary of speech formants, referencing from internal lists in order to identify individual sounds and words from an audio signal. By their very nature, systems of this sort tend to function only for a small group of speakers, as variations in vocal tract shape and size, vocal pitch, speech rate, dialect, accent, and other factors make comprehension unreliable outside of the system's intended users. Much like AI research of the time, these initiatives demonstrated some initially promising results, but were unable to develop into the sort of robust systems that their financers, such as DARPA, were actually interested in. [16]

One of the most persistent barriers to effective speech recognition was the sheer amount of computing power required to analyze an audio stream in real-time. The hardware available simply was not fast enough to make meaningful experimental progress, or to build any marketable, commercially-available systems until the 1990s. One of the first commercial systems to respond to human speech was Dragon Dictate, released for Microsoft DOS in 1990. [12]

Such systems are now used in a variety of applications. The first significant application of speech recognition was dictation, in which the words a speaker says are automatically transcribed into digital text, the specific applications of which might be composing documents in a word processor, automatically recording minutes from a meeting, or digitizing the proceedings of courts and government bodies. More recently, however, novel applications of this technology have used voice input to facilitate interaction with a program, such as virtual assistants, automatic call systems, or networked smart devices. In these situations and others, automatic speech recognition systems have begun to be used in capacities that previously would have required the conscious attention and communication of a human being. [9]

2.3 Audiobook Production Process and Error Correction

Creating an audiobook is a multi-stage process. The first and most straightforward of these steps is the recording process, which is ideally carried out with a decent-quality microphone in a quiet room with minimal reverberation, or echo. Reverberation reduction can be achieved in multiple ways, from recording in a closet filled with hanging clothes to using a dedicated studio where one or more walls are padded with

Table 2.1: Audiobook Error Types and Examples

Error Type	Examples
Non-speech sounds	background noise, breaths, coughs, sneezes, lip-smacks.
Performance mistakes	mispronounced words, incorrect words, superfluous words.
Pauses and silence	gaps between words, paragraphs, or titles either too long or too short.
Volume discrepancies	sections that are uncomfortably loud or too quiet to hear clearly.
Tonal discrepancies	emotional inflection not fitting the subject, or simply inconsistent.

sound-dampening foam panels. In this environment, the narrator sits and reads the text either from paper or a device with a screen, typically for no more than a few hours at a time in order to preserve the quality and consistency of their voice.

The ideal is to record at a high level of fidelity, and in a way that minimizes how much editing is required. Often, this involves the narrator simply stopping to correct mistakes immediately after they are made, but even in the best case, at least some manual adjustment is necessary. These may involve the narrator re-recording certain segments, or they may be able to be adjusted through the use of audio engineering tools.

One application of a static algorithm used in audio processing is noise removal, which is typically performed by using Fourier Analysis. In this process, a sample is taken to identify specific frequencies which relate to background noise such as an electronic hum or a fan blowing. The rest of the audio stream is then analyzed, and anywhere these noise frequencies occur, the intensity of these frequencies is reduced.

Table 2.2: Corrective techniques and application examples

Corrective Technique	Applications
Normalization	volume discrepancies, variance between multiple recording sessions.
Noise Filtering	non-speech sounds, environmental noise.
Pop-Filtering	breath sounds, plosives
De-Essing	hissing when pronouncing s-sounds
Cutting/silencing	lip smacks, breath sound, background noises, excess words
Re-recording	mispronunciation, substitution of wrong words, incorrect inflection

Once the recording is complete, it is usually necessary to edit the resulting audio file to meet publisher standards. Common issues and problems that must be addressed when editing raw recordings include those listed in Table 2.1.

Modern audio editing software incorporates many tools to modify the characteristics of an audio signal. Some of these techniques include those listed in Table 2.2. Tools and filters like these are very useful within very specific contexts, but are limited to problem cases where the problem can be reduced to a simple algorithm and seamlessly corrected with the rest of the performance. Overall, a minority of errors and issues can be corrected entirely through automated filters and tools like these, and the rest must be manually reviewed.

Finally, the recording must be reviewed in its entirety, listened through from beginning to end. In this process, cognitive filling can become an issue. It is a function of human hearing to unconsciously fill in gaps and alter what is heard in a way that may more closely match what the brain expects to hear. This can lead to certain errors being overlooked, particularly if the listener has worked for several hours and may be unable to maintain focus. Further, if the listener is also the narrator, listening to their own work, with which they are already familiar, may make it even more likely to overlook mistakes.

In this capacity, while machine-driven analysis would have its own issues, inattention due to fatigue and repetition are problems unique to a human listener, and would not be a concern for an automated system.

2.4 Functional Overview of Automatic Speech Recognition Systems

An automatic speech recognition (ASR) system accepts speech in the form of an audio signal and converts that signal into text, as shown in Figure 2.1. A standard ASR system contains several components that operate together to convert an audio signal into readable text. These components include the acoustic model, the language model, and the lexicon, or dictionary.

Before any work can be done, it is necessary to first convert the raw audio input into a processed signal. Rather than operate on a basic waveform, most ASR systems interpret sequences of vectors that identify changes in the strength of frequency bands over a particular window in time, such as can be seen illustrated in the spectrographs in Figure 2.2. These typically divide the audio signal into time frames calculated across overlapping 'windows' of between 20 and 40 milliseconds in length, spaced every 10 milliseconds or so. Vectors encode the strength of the signal within a set of frequency bands, for which those frequencies are often mapped on a logarithmic, rather than linear scale. In this fashion, the audio signal is divided into blocks of fixed length, measuring the change in intensities from one vector to another. These differences can be analyzed and identified as components of a specific tone or speech sound, making interpretation easier. The minimum length of each speech sound is typically three frames.

An ASR system uses an acoustic model to identify which speech sounds, or 'phones', are present within the audio signal. Under ideal circumstances, the audio signal itself is clear, and contains little or no non-speech sound. Within many ASR applications, however, this will not be the case, and a speaker may be communicating into a device in the presence of background noise or even other speakers in the vicinity. In these cases, the system may not only have to account for ordinary noise, but also to filter out competing vocal sounds which may otherwise have been valid input. Some ASR systems also include noise models specifically designed to compensate for unwanted clutter in an audio signal, such as breaths, clicks, or background noise. Rather than provide one specific interpretation of the speech signal, a typical acoustic model will assess a set of probabilities for what is being said, which can then be interpreted using the lexicon and the language model.

Identifying phones also involves classifying them as specific parts that are combined to form words, as defined by the lexicon that the ASR system is using, which contains a list of valid words as well as the pronunciation of those words. These phones are labeled using a key derived from a subset of the International Phonetic Alphabet (IPA). Used in the English language, for example, 'cat' might be rendered as ‘k æ t’, beginning with a hard-k sound, leading into the ‘æ’ (‘ash’) sound, and finishing with the the ‘t’ sound. Regional accents within languages lead can to significant variation in pronunciation, and there are several other ways that ‘cat’ can be described in IPA units, including ‘k a t’ and ‘k^{h} æ t’. Gaps of silence between words are also recognized as their own distinct unit according to acoustic models. As an example, a portion of the lexicon used in this work is shown in Table 2.3.

The language model of an ASR system both relies on and informs the acoustic model and lexicon. Where the acoustic model analyzes what sounds are present in the signal, and the lexicon determines what words those sounds might be mapped to, the language model evaluates what combinations of words and sounds are likely to occur in the language being spoken. The likely probabilities of phones and silences may not always agree with results that are analyzed in the context of language. However, a lexicon and language model cannot account for all cases, such as names of people, titles of places, and proper nouns which may not exist in the lexicon, or phrases or figures of speech which may be improvised on the spot, as well as cases of nonstandard pronunciation or dialect. To account for such edge-cases, most ASR systems account for the possibility of output that is not classified in the model.

The language model takes word frequency into account. For example, many English language models would account for the fact that ‘the’ almost always appears before nouns, or adjective-noun groups, and is rarely used adjacent to another ‘the’. Grammatical structure, which is incorporated in the sequential modeling approaches used for language models, is used to disambiguate cases as homophones like there, their, and they’re, or Mary, merry, and marry.

The acoustic model may also have further influence here, as robust acoustic models

Table 2.3: Lexicon and Pronunciation Examples

ARRIVAL	ER AY V AH L	CELL	S EH L
ARRIVALS	ER AY V AH L Z	CELL'S	S EH L Z
ARRIVE	ER AY V	CELLA	S
ARRIVED	ER AY V D	CELLAR	S EH L ER
ARRIVES	ER AY V Z	CELLARS	S EH L ER Z
ARRIVING	ER AY V IH NG	CELLED	S EH L D
ARROGANCE	EH R AH G AH N S	CELLI	CH EH L IY
ARROGANT	EH R AH G AH N T	CELLINI	CH EH L IY N IY
ARROGANTLY	EH R AH G AH N T L IY	CELLIO	CH EH L IY OW
ARROGATE	AE R OW G EY T	CELLIST	CH EH L AH S T
ARROW	AE R OW	CELLMARK	S EH L M AA R K
ARROW'S	AE R OW Z	CELLMARK'S	S EH L M AA R K S
ARROW'S(2)	EH R OW Z	CELLNET	S EH L N EH T
ARROW(2)	EH R OW	CELLO	CH EH L OW
OURS	AW ER Z	PSYCHOTIC	S AY K AA T IH K
OURS(2)	AA R Z	PSYLLIUM	S IH L IY AH M
OURS(3)	AW R Z	PTACEK	T AA CH EH K
OURSELF	AW ER S EH L F	PTAK	T AE K
OURSELF(2)	AA R S EH L F	PTOLEMAIC	T AA L AH M EY IH K
OURSELVES	AW ER S EH L V Z	PTOLEMY	T AA L AH M IY
OURSELVES(2)	AA R S EH L V Z	PTOMAINE	T OW M EY N
OURSO	ER S OW	PTOMAINES	T OW M EY N Z
OUSEBA	UW S AH B AH	PTOVSKY	P AH T AO V S K IY
OUSLEY	AW S L IY	PTY	T AY
OUST	AW S T	PTYON	T AY AO N
OUSTED	AW S T IH D	PU	P UW
OUSTER	AW S T ER	PUAT	P Y UW AE T
OUSTING	AW S T IH NG	PUB	P AH B

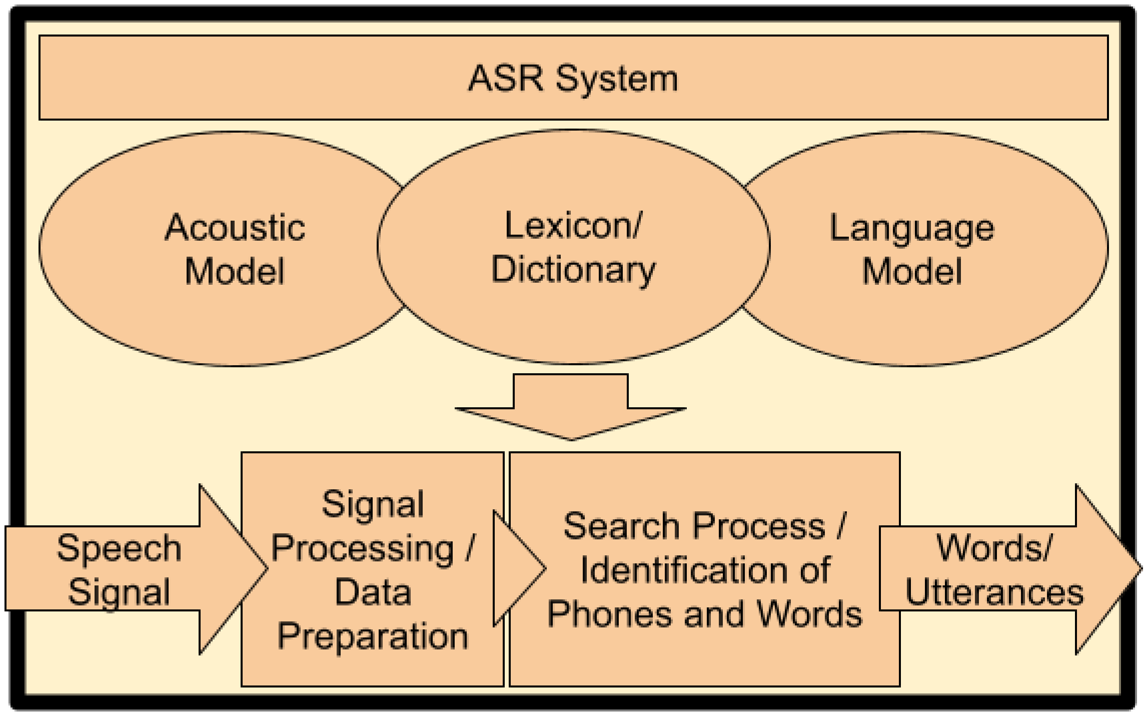

ASR System
Acoustic Model
Lexicon/ Dictionary
Language Model
Speech Signal
Signal Processing / Data Preparation
Search Process / Identification of Phones and Words
Words/ Utterances

Figure 2.1: Organization and function of a typical automatic speech recognition system

generally account for how closely a particular group of phones matched expected examples of those sounds. In these cases, if the acoustic model's 'best guess' seems not to be grammatically sensible, it may provide the next-best guess until the potential text output is more likely to be correct using the combined probabilities. Upon calculating these possibilities, the language model will then provide the most likely speech contents of the audio signal as text.

Figure 2.2 shows a general summary of several of the outputs at each of the steps undertaken by an ASR system, although these have been manually labeled by a human being, rather than an ASR system.

Forced alignment is a similar but distinct process from speech recognition. In

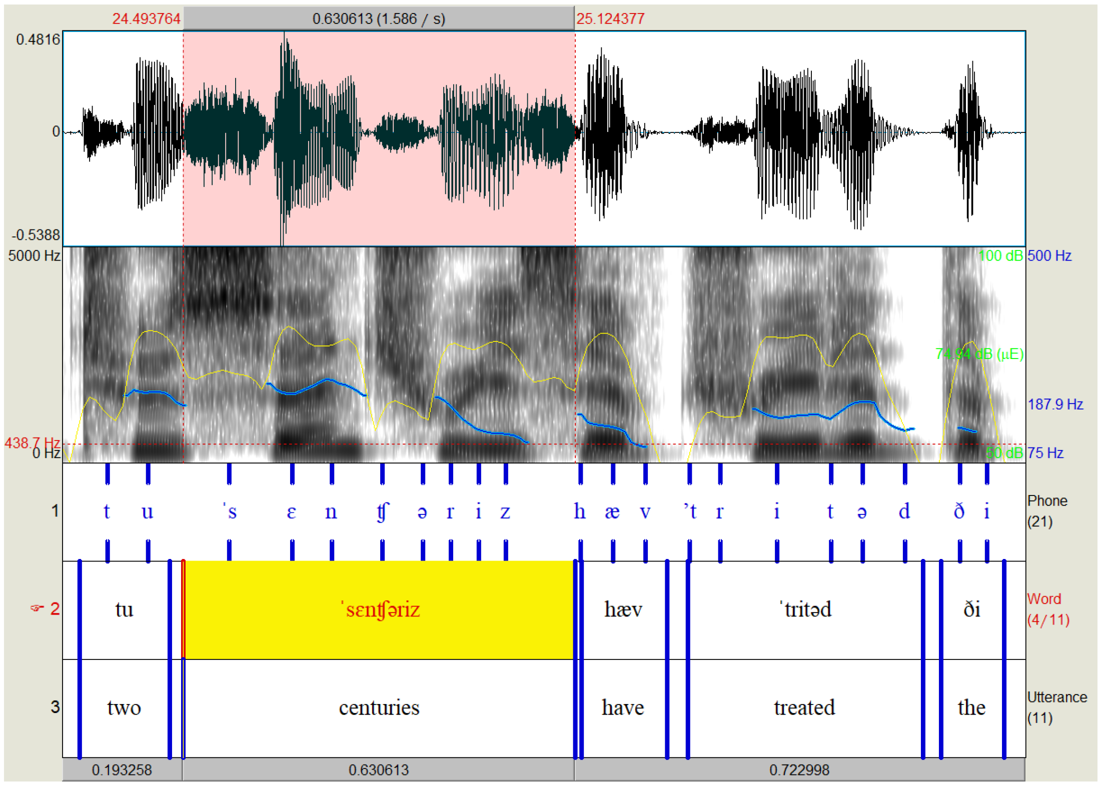


Figure 2.2: An audio signal being examined in PRAAT software, showing the signal waveform, spectrograph with formants, phones, words, and language output.

forced alignment, the words present in an audio signal are already known, and the goal is not to determine what is being said, but to positively identify at precisely which time interval each sound, word, or utterance is being spoken. The aligner receives both a text transcript and an audio signal as inputs, and produces time-aligned word strings as output.

As a process, alignment is as relevant to the goal of automatic error detection as general-purpose ASR. In the context of the typical use of ASR systems, alignment data is used in the training of acoustic and language models, but it is additionally applicable for error-detection. All audiobook recordings are the recitation of a known text, and the time-aligned strings produced by an aligner are also useful as a data-point in examining a recording's fidelity to that text, and finding portions of the signal that do not align with the words.

Chapter 3

Methodology

This chapter summarizes the systems and processes used in attempting to construct a speaker-independent speech recognition system from pre-existing software, with the intent that it be capable of detecting errors and mistakes in speech performances. This process involved creating a dataset from a series of recordings and their accompanying transcripts, processing those with Aeneas, a tool for forced alignment and synchronization, and using that processed dataset as the input for the Kaldi speech recognition system. Performance metrics were gathered at different stages in the project, including statistics on word and phrase-length during forced alignment calculated from Aeneas as well as word error rate (WER) provided by Kaldi.

3.1 Data Sources

The primary speech data used for this thesis was sourced from samples of audioblooks narrated by Leon Tietz. A total of twenty samples were used, typically recorded from the introduction segment of a given book, ranging between one and six minutes in length, and with both the transcript and audio available for use. A total of approximately 82 minutes of speech data was used in this work, divided into 668 utterances. The audio itself was provided in FLAC (free lossless audio codec) format, but was reprocessed into 16kHz wav files for compatibility with Aeneas and Kaldi. The text transcripts of all spoken audio segments were also included with the speech data, and required their own processing to remove superfluous punctuation and capitalization, as well as to manually divide the text into utterances.

The audiobook dataset included three versions of each sample: 1) the original unaltered recording, 2) a recording that had been automatically filtered to remove noise and normalize amplitude, and 3) a recording that had been filtered in the same way, and edited by hand to trim overly-long silences, with certain portions re-recorded for better comprehensibility and clarity.

At the beginning of the project, a speaker-dependent model based on the sample dataset was trained, but could not be finished. This was because the recommended minimum size of a dataset for training a speech model is 10,000 utterances, far more than the 668 utterances available.

Another collection of data was used to examine how Kaldi evaluated characteristics of audio signals, such as the ability to ignore background noise and non-speech sounds, as well as to interpret intentionally mispronounced words. This was much smaller, consisting of only dozens of utterances, and was recorded personally rather than sourced from another individual or voice actor.

This recording was performed from a script adapted from song lyrics spoken as prose, with the introduction of deliberate errors. The text of this recording is in Appendix A.5. For the portion regarding erroneous pronunciation, words were intentionally pronounced unusually, or substituted for similar-sounding words. To check environmental noise tolerance, errors included both human-produced non-speech sound, such as coughing and throat-clearing, and environmental noises such as the creak of a chair or door, the sound of a ventilation fan, or knocking on a table.

3.2 Summary of Tools Used

The initial research phase of this thesis was concerned with assembling a collection of freely-available software capable of performing tasks related to speech recognition and processing. It should be noted that several of the tools that seemed to show the most promise were unavailable or missing dependencies. These included Speech Kitchen, a resource used by several other speech recognition systems, which at the time of evaluation in Spring of 2021 had recently become defunct.

A constant risk with free and open-source software is that documentation and maintenance of a particular project may be lax or even completely absent. Moreso than is the case with other software, the software may become outdated, face compatibility issues, or that it and its dependencies may become unavailable. Any of these can result in considerable loss of time as components of the project must be reworked or even started from scratch, and can also complicate reproducibility for future research.

After evaluation, the software found to be most promising for initial research into speech-error detection included Kaldi, Aeneas, Praat, and Audacity. Scripts written in the Python programming language, Windows command-line, Linux Bash and SoX were used to streamline the process of implementing several of these programs into a processing pipeline, as illustrated in Figure 3.1.

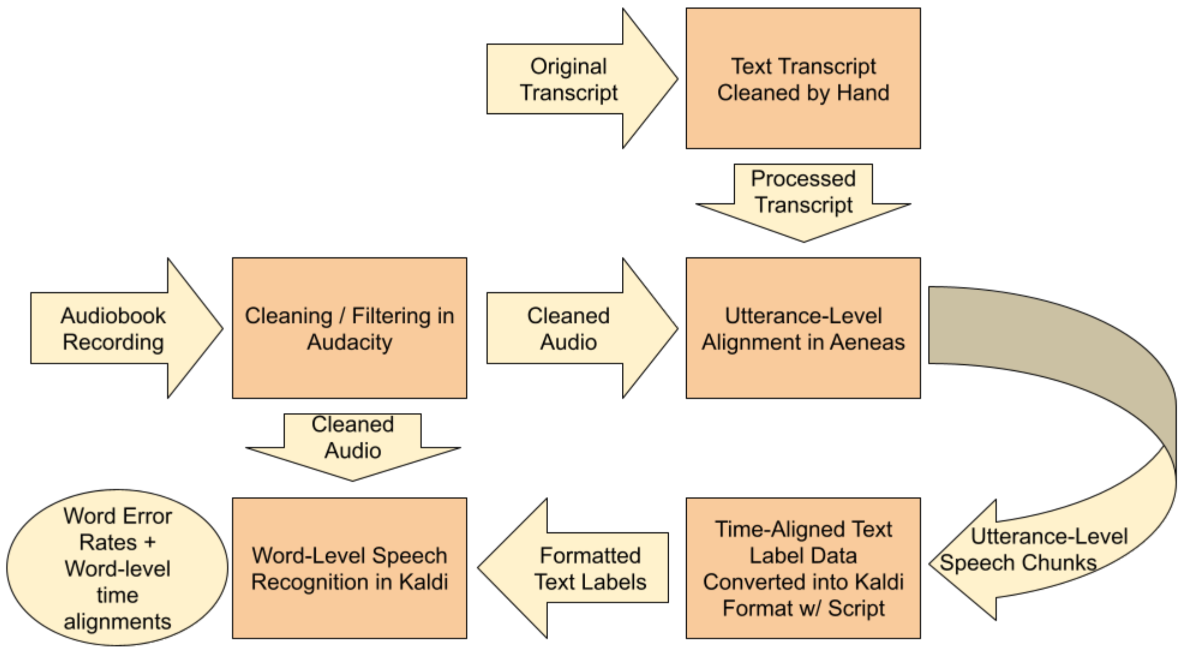

Original Transcript
Text Transcript Cleaned by Hand
Processed Transcript
Audiobook Recording
Cleaning / Filtering in Audacity
Cleaned Audio
Utterance-Level Alignment in Aeneas
Cleaned Audio
Word Error Rates + Word-level time alignments
Word-Level Speech Recognition in Kaldi
Formatted Text Labels
Time-Aligned Text Label Data Converted into Kaldi Format w/ Script
Utterance-Level Speech Chunks

Figure 3.1: An illustration of the pipeline used to convert audiobook samples and transcripts into an assessment of Word Error Rates and word-level time alignments, including Kaldi and Aeneas

3.2.1 Kaldi

Kaldi[1]is a C++ toolkit for speech recognition tasks, developed for Linux and Linux-based platforms, and intended primarily for use by researchers [13]. While some attempts have been made to enable Kaldi to run on other platforms such as Windows, a reliable implementation could not be located when preparing tools for this project. Kaldi was run on an Ubuntu Linux virtual machine WSL (Windows subsystem for Linux) on a laptop with 16GB of onboard memory and an Intel Core i7 8565U processor. Kaldi was used to recognize words given audio datasets, and forced alignments which provide word-level time annotation. Along with recognized word and time alignments, Kaldi provded word error rates for recognition of test sets, which can indicate the approximate accuracy of Kaldi's speech recognition.[13] Word error rate and other performance metrics are described in Section 3.4.

3.2.2 Aeneas

Aeneas[2] is a Python/C software library and tool, originally developed by the now-dissolved research lab ReadBeyond, designed to provide forced alignments of audio and text [11]. Although developed natively for Debian linux, pre-built versions for various operating systems, including Windows, are maintained by developers associated with the SIL organization, a group that supports language translation [8]. SIL's 64-bit windows build 1.7.4.0 was used for forced alignment in order to prepare and format a dataset for use in Kaldi.

[1]https://kaldi-asr.org/

[2]https://www.readbeyond.it/aeneas/

3.2.3 Praat

Praat[3] is a program developed for analysis of speech, providing the ability to view audio waveforms and spectrographs, identify speech formants, and to modify aspects of these waveforms, including pitch and rate. Praat was not directly used to gather data for this thesis, instead it was used for visualization and analysis of speech components and the characteristics of certain sounds and speech errors. In addition, there is interoperability between Praat and Kaldi, with the latter program being able to export pre-labeled speech data in a format Praat can open for further analysis. [4] There was insufficient time within the scope of this work to explore these capabilities, but this could prove useful for other projects focusing on the study of the transition points and individual phones.

3.2.4 Audacity

Audacity[4] is a free, open-source audio editing program, capable of recording audio signals, editing audio files, and exporting into most popular audio formats. It features both built-in filters and processing algorithms, as well as the capability to utilize user-created plugins to expand its functionality. Audacity version 2.3.2 was used to record several specific sets of test data, apply noise removal and normalization filters to various audio samples, and to export audio files for Kaldi and Aeneas in the correct format and bitrate. [1]

Audacity's noise reduction features may also have potential use as the basis for

[3]https://www.fon.hum.uva.nl/praat/

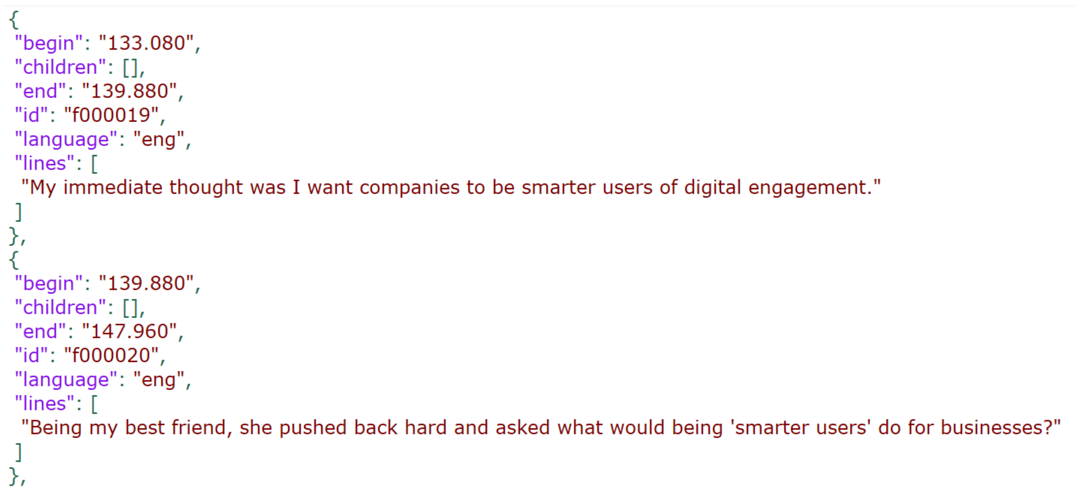
[4]https://www.audacityteam.org/

an error detection method. This technique uses Fourier analysis to determine which frequencies are present within a sample of known background noise, such as the whirring of a fan or electronic hum, and may then reduce those specific frequencies to render them less audible throughout [3]. This process of noise reduction does have a negative impact on the quality of the recording as a whole, as it is nearly impossible to avoid at least some parts of the spoken audio being unintentionally filtered out by the system. Primarily for this reason, the use of noise filtering is generally discouraged in professional productions, although in most cases, to most listeners, the difference is unlikely to be noticeable.

3.3 Procedures

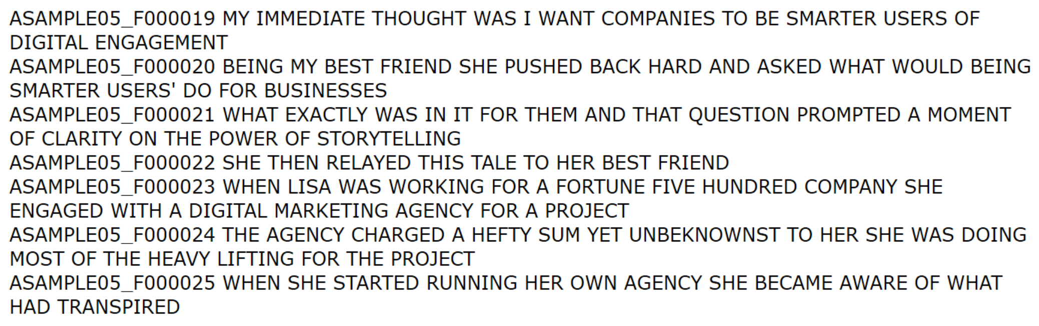
The first step in processing the audiobook dataset required each audio transcript to be subdivided into utterances. These were typically about the length of a sentence, although some were as short as a single word, such as the chapter and section headers within the text. This process involved stripping out any non-letter symbols from the text file that were not punctuation, including nonstandard forms of single and double quotation marks. In parallel, the audio files were converted from stereo FLAC into mono-channel 16-bit, 16-kHz wav files, which is necessary for compatibility with Kaldi.

Aeneas performed forced alignment on the dataset to provide indicators for where each utterance begins and ends within its corresponding audio file. Each file's indicators were exported into a JSON file containing each utterance, its unique identifiers, and other related data. The JSON files contained all necessary information for a



```
{
 "begin": "133.080",
 "children": [],
 "end": "139.880",
 "id": "f000019",
 "language": "eng",
 "lines": [
  "My immediate thought was I want companies to be smarter users of digital engagement."
 ]
},
{
 "begin": "139.880",
 "children": [],
 "end": "147.960",
 "id": "f000020",
 "language": "eng",
 "lines": [
  "Being my best friend, she pushed back hard and asked what would being 'smarter users' do for businesses?"
 ]
},
```

Figure 3.2: Aeneas output in JSON format

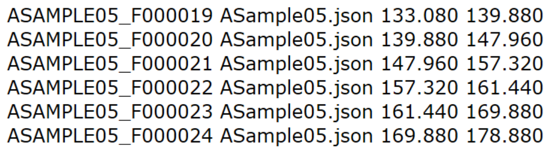


```
ASAMPLE05_F000019 MY IMMEDIATE THOUGHT WAS I WANT COMPANIES TO BE SMARTER USERS OF
DIGITAL ENGAGEMENT
ASAMPLE05_F000020 BEING MY BEST FRIEND SHE PUSHED BACK HARD AND ASKED WHAT WOULD BEING
SMARTER USERS' DO FOR BUSINESSES
ASAMPLE05_F000021 WHAT EXACTLY WAS IN IT FOR THEM AND THAT QUESTION PROMPTED A MOMENT
OF CLARITY ON THE POWER OF STORYTELLING
ASAMPLE05_F000022 SHE THEN RELAYED THIS TALE TO HER BEST FRIEND
ASAMPLE05_F000023 WHEN LISA WAS WORKING FOR A FORTUNE FIVE HUNDRED COMPANY SHE
ENGAGED WITH A DIGITAL MARKETING AGENCY FOR A PROJECT
ASAMPLE05_F000024 THE AGENCY CHARGED A HEFTY SUM YET UNBEKNOWNST TO HER SHE WAS DOING
MOST OF THE HEAVY LIFTING FOR THE PROJECT
ASAMPLE05_F000025 WHEN SHE STARTED RUNNING HER OWN AGENCY SHE BECAME AWARE OF WHAT
HAD TRANSPIRED
```

Figure 3.3: Sample text input to Kaldi, converted from Aeneas output

Kaldi speech model to be used on them, but required processing into the correct format. This processing was achieved with a Python script that scanned for the crucial information in each entry, creating two new files, ‘text’ and ‘segments’ as output, as illustrated in Figure 3.2, Figure 3.3 and Figure 3.4. (See Appendix A.1 for the conversion script.)

Before this data could be used directly in Kaldi, however, it was necessary to cor-



```
ASAMPLE05_F000019 ASample05.json 133.080 139.880
ASAMPLE05_F000020 ASample05.json 139.880 147.960
ASAMPLE05_F000021 ASample05.json 147.960 157.320
ASAMPLE05_F000022 ASample05.json 157.320 161.440
ASAMPLE05_F000023 ASample05.json 161.440 169.880
ASAMPLE05_F000024 ASample05.json 169.880 178.880
```

Figure 3.4: Segment timings input for Kaldi, converted from Aeneas output

rect vocabulary discrepancies, as some of the characters, words, and word-combinations used in the speech samples were not present in Kaldi's lexicon, or were not present in the same form as expected. These discrepancies were detected during the initial steps to prepare a language model, and required noting which words were missing from the existing lexicon. Some of these were simply words that were missing altogether, and were easily resolved by adding the word to the lexicon along with a phonetic pronunciation guide for identifying that word's sounds.

In other cases, however, words were spelled differently than expected by the lexicon. Initially, the text portion of the audiobook dataset retained hyphens. Simply adding hyphenated words into the lexicon proved fruitless for recognition because these forms were not included in the language model. For example, the compound word "part-time" is rejected in favor of individually detecting "part" and "time", and therefore produces a substitution error due to finding something other than expected. Removing all hyphens from the sample data was a fairly simple process and resolved these issues.

For an acoustic model, the pre-trained Multilingual LibriSpeech model was used, trained on an open dataset of LibriVox recordings by OpenSLR [14]. This was implemented through a process outlined by Desh Raj, which involves installing and using a pre-existing model for Kaldi rather than needing to train a model on that specific dataset [6]. This method allowed the use of a dataset which would otherwise be too small to be trained on. The Librispeech set also proved to be particularly well-suited for evaluating audiobook samples, due to the Librispeech dataset being assembled from public-domain audiobook recordings sourced from LibriVox [2].

The result of Raj's method is also complimentary to the main objective of this thesis. The process the article describes evaluates the LDC corpora WSJ0 and WSJ1 datasets, which consists of recordings of newspaper articles from the Wall Street Journal.[7][10] This type of intentional and planned speech is similar in tone and structure to audiobook recordings and other scripted performances.

Kaldi's forced alignment and segmentation systems were used to evaluate the scoring accuracy and error detection rate of this generalized model against the data set, producing most of the necessary metrics to judge the viability of error-detection using freely-available software tools.

An additional test was performed after the initial analysis in order to see what types of speech and noise errors could be detected. This test used the second audio dataset created for this work to examine how poor input data would influence the outcomes of Aeneas and Kaldi.

3.4 Process Pipeline

Converting a set of text transcripts and audio files into a scored result in Kaldi was performed as follows. This process generally following Desh Raj's outline, although with some differences. [6]

1. Collect sample text and audio.

2. Clean and edit text, removing unnecessary characters.
3. Format audio to 16Khz .wav files for Kaldi compatibility.
4. Run Aeneas as batch file to process each combination of text and audio file.
5. Run Python conversion script on Aeneas output to create 'segments' and 'text' files. (See Appendix A.1.)
6. Format converted 'segments' and 'text' files to Linux end-of-line style, otherwise Kaldi will not function.
7. Create a new project folder within Kaldi data directory.
8. Move 'segments' and 'text' to data directory and prepare wav.scp and utt2spk. Examples of wav.scp and utt2spk are in Appendices A.3 and A.4.
9. To generate other necessary files in the project folder, Run data_prep.sh, which is located in the subdirectory kaldi\egs\librispeech\s5\local. wsj_data_prep.sh is not suitable, as this dataset is not the WSJ dataset.
10. Generate high-res MFCCs (Mel-frequency cepstrum coefficients) for each utterance.
11. Extract CMVN (Cepstral mean and variance normalization) features and i-vectors.
12. Decode with tgsmall (small trigram model). If processing the data on local hardware rather than cloud computing, use run.pl rather than queue.pl.
13. Rescore with RNNLM (recurrent neural net language model) using either run.pl or queue.pl depending on local or cloud computation.

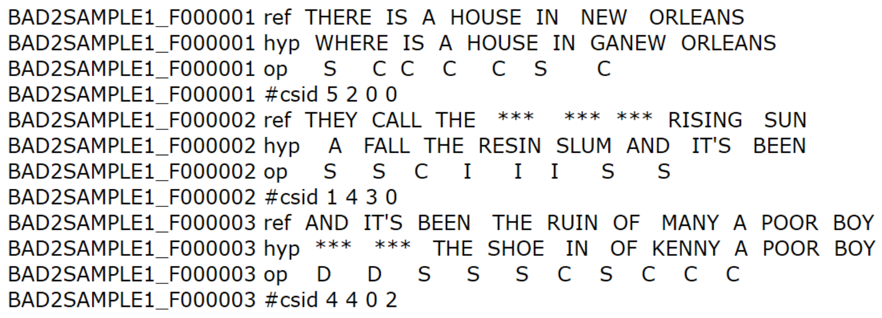
3.5 Performance Metrics

Word error rate (WER) represents what proportion of the output of the ASR system matched what was predicted when conducting forced alignment with the reference text transcript. Any unexpected word counts as an error, but the errors themselves are divided into insertions, substitutions, and deletions. Figure 3.5 shows a brief sample that demonstrates these errors in practice.

Insertions are labeled when the ASR system discovered or interpreted a word as having been part of the recording that was not present in the transcript. Substitutions occur when the audio output of the speech system interprets a word at a given time and place as being different than the word that was expected. Deletions involve a word present in the transcript that seemed not to have been said in the audio recording. The WER is calculated as a factor of all of these errors, where the combined total of insertions, deletions, and substitutions is divided by the number of words in the transcript as a whole, as calculated in the following formula:

$$WER = \frac{(Ins + Del + Sub)}{Number\ of\ Words}. \tag{3.5.1}$$

Points of difference were measured between the edited and unedited versions of the audiobook dataset, as one method of searching for sections of the audio that had been manually modified. These points were determined by analyzing differences in the lengths of segments and phrases in the time-aligned output provided by Aeneas. Areas where changes have been made are often a different length than in the original recording. Locating these areas is potentially applicable to tasks such as training for machine learning models to detect errors or corrections in recordings.



```
BAD2SAMPLE1_F000001 ref  THERE  IS  A  HOUSE  IN   NEW   ORLEANS
BAD2SAMPLE1_F000001 hyp  WHERE  IS  A  HOUSE  IN  GANEW  ORLEANS
BAD2SAMPLE1_F000001 op     S     C  C    C     C   S      C
BAD2SAMPLE1_F000001 #csid 5 2 0 0
BAD2SAMPLE1_F000002 ref  THEY  CALL  THE   ***    ***  ***  RISING   SUN
BAD2SAMPLE1_F000002 hyp    A   FALL  THE  RESIN  SLUM  AND   IT'S   BEEN
BAD2SAMPLE1_F000002 op     S     S    C     I      I    I     S      S
BAD2SAMPLE1_F000002 #csid 1 4 3 0
BAD2SAMPLE1_F000003 ref  AND  IT'S  BEEN   THE  RUIN  OF   MANY  A  POOR  BOY
BAD2SAMPLE1_F000003 hyp  ***   ***   THE  SHOE   IN   OF  KENNY  A  POOR  BOY
BAD2SAMPLE1_F000003 op    D     D     S     S     S    C    S    C    C    C
BAD2SAMPLE1_F000003 #csid 4 4 0 2
```

Figure 3.5: Sample of Kaldi output with many intentional mistakes and deviations from a transcript, with labels for correct (C), substituted (S), inserted (I), and deleted (D) words.

Chapter 4

Results

This chapter presents the results of three specific tests with the objective of determining the viability of building an error-detection system for speech performances out of existing speech recognition and filtering tools. These involved analysis of the audiobook dataset through both Aeneas and Kaldi, to determine its potential suitability for training a machine learning system, as well as the use of several original audio sequences to test the capabilities of those tools to locate errors within a performance.

4.1 Effect of Editing and Processing on Speech Comprehension

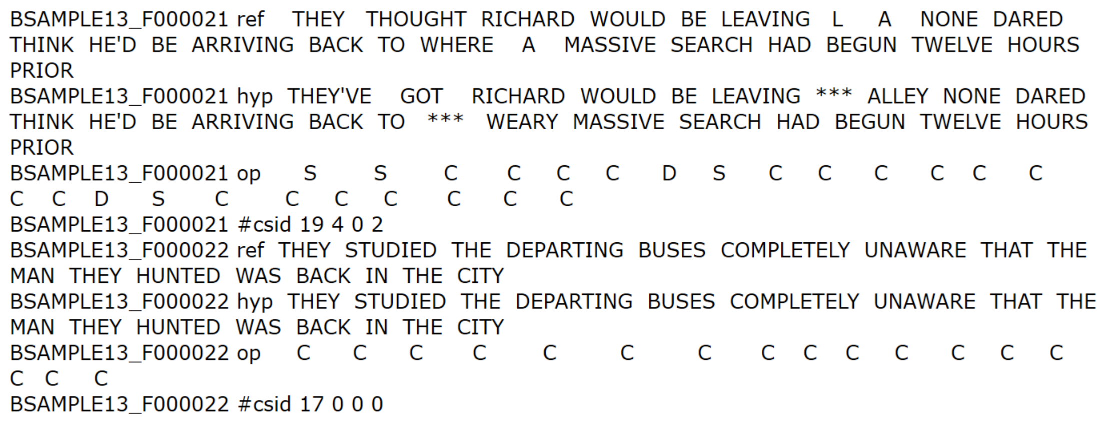
The first test was to determine how pre-processing and manual editing of each audio sample would impact the ability of Kaldi's speech recognition system to successfully transcribe the spoken text within the audio samples. For this purpose, the following three versions of the audiobook dataset were used:

- A raw, unmodified copy of the original recording.
- A copy of the recording that had been automatically filtered via noise reduction and normalization.
- The 'finished' version of the recording, both filtered and manually edited.

Each of these sets was recognized by Kaldi using the pre-existing LibriSpeech model. The results were created using two different language models available in Kaldi: the small trigram language model (tgsmall) and the recurrent neural network language model (RNNLM). The baseline word error rates are shown in Table 4.1. Results are presented in terms of word error rate, which is calculated as shown in Equation 3.5.1. The test set size was 10,978 words for the unfiltered, unedited data, and 11,020 words for the filtered sets. The discrepancy between the two was due to breaking up hyphenated words into their component parts between processing the first and second datasets. Sample word-level results are shown in Figure 4.1.

Table 4.1: Word error rates for recognition of audiobook data, including number of inserted words (ins), deleted words (del), and substituted words (sub) with two types of language models, the small trigram model (tgsmall) and the recurrent neural network language model (rnnlm). Test data may have been filtered and/or edited.

Processing Method		WER	Ins	Del	Sub
Unfiltered + Unedited	tgsmall	10.34%	300	102	733
	rnnlm	9.08%	287	110	600
Filtered + Unedited	tgsmall	9.57%	245	101	709
	rnnlm	8.31%	237	105	574
Filtered + Edited	tgsmall	9.64%	256	99	707
	rnnlm	8.17%	240	95	565



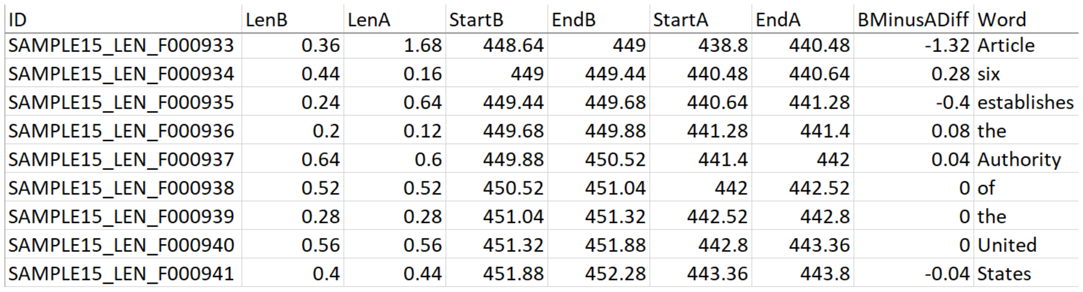
```
BSAMPLE13_F000021 ref   THEY   THOUGHT  RICHARD  WOULD  BE  LEAVING   L    A   NONE  DARED
THINK  HE'D  BE  ARRIVING  BACK  TO  WHERE   A   MASSIVE  SEARCH  HAD  BEGUN  TWELVE  HOURS
PRIOR
BSAMPLE13_F000021 hyp  THEY'VE   GOT   RICHARD  WOULD  BE  LEAVING  ***  ALLEY  NONE  DARED
THINK  HE'D  BE  ARRIVING  BACK  TO  ***   WEARY  MASSIVE  SEARCH  HAD  BEGUN  TWELVE  HOURS
PRIOR
BSAMPLE13_F000021 op     S      S      C      C    C    C     D    S     C    C     C     C   C     C
C   C   D    S     C      C    C    C      C     C     C
BSAMPLE13_F000021 #csid 19 4 0 2
BSAMPLE13_F000022 ref  THEY  STUDIED  THE  DEPARTING  BUSES  COMPLETELY  UNAWARE  THAT  THE
MAN  THEY  HUNTED  WAS  BACK  IN  THE  CITY
BSAMPLE13_F000022 hyp  THEY  STUDIED  THE  DEPARTING  BUSES  COMPLETELY  UNAWARE  THAT  THE
MAN  THEY  HUNTED  WAS  BACK  IN  THE  CITY
BSAMPLE13_F000022 op    C     C     C     C      C       C       C     C   C   C    C     C    C    C
C   C    C
BSAMPLE13_F000022 #csid 17 0 0 0
```

Figure 4.1: Sample of Kaldi output showing insertion/substitution. Many words are misinterpreted, rather than being genuine performance mistakes.

4.2 Kaldi Error Tolerance

This test evaluated the accuracy of Kaldi's speech recognition model when evaluating a recording containing deliberately substituted words, erroneous pronunciations, non-speech noises, and background noise within the sample. Kaldi was used to perform forced alignment on this audio, coupled with segmentation data based on the original text rather than the intentionally flawed performance. The output produced was not only the word error rate but also the system's best guess at what words and phones were being said and their associated time boundaries.

The result of this test demonstrated that mispronounced or substituted words were correctly identified as errors and deviations from the original transcript in almost all cases, and non-speech noise had no observable impact on the capability of Kaldi's language models to identify speech. This outcome was somewhat disappointing, but not surprising. Kaldi's primary purpose is to recognize and transcribe speech, not noise, and the ability to ignore non-speech sounds and focus only on speech is in



ID	LenB	LenA	StartB	EndB	StartA	EndA	BMinusADiff	Word
SAMPLE15_LEN_F000933	0.36	1.68	448.64	449	438.8	440.48	-1.32	Article
SAMPLE15_LEN_F000934	0.44	0.16	449	449.44	440.48	440.64	0.28	six
SAMPLE15_LEN_F000935	0.24	0.64	449.44	449.68	440.64	441.28	-0.4	establishes
SAMPLE15_LEN_F000936	0.2	0.12	449.68	449.88	441.28	441.4	0.08	the
SAMPLE15_LEN_F000937	0.64	0.6	449.88	450.52	441.4	442	0.04	Authority
SAMPLE15_LEN_F000938	0.52	0.52	450.52	451.04	442	442.52	0	of
SAMPLE15_LEN_F000939	0.28	0.28	451.04	451.32	442.52	442.8	0	the
SAMPLE15_LEN_F000940	0.56	0.56	451.32	451.88	442.8	443.36	0	United
SAMPLE15_LEN_F000941	0.4	0.44	451.88	452.28	443.36	443.8	-0.04	States

Figure 4.2: Word-by-word segmentation comparing pre- and post-edited samples

line with the program's design, where most users want speech transcribed and noise ignored. A portion of the result is shown in Figure 3.5. Kaldi ignored all noise, but guessed the spoken text fairly accurately. The test dataset used is described in more detail in Section 3.1.

4.3 Pre- and Post-Edit Data Analysis

One of the long-term goals of this work is to determine how plausible it would be to use machine learning to identify performance mistakes and correct them. While the full extent of implementing such an effort is beyond the scope of this thesis, it is a simpler proposition to examine test data to determine whether parts of the data processing pipeline implemented here prove useful in creating or organizing training data for a machine learning agent. The first such analysis compares the Aeneas segmentation of the speech data both before and after the samples had been manually edited to remove any defects, excessive silences, speech sounds, and other errors. This was performed using a Python script creating a .csv file directly from the segment data already generated by Aeneas. (See Appendix A.2.)

For this comparison, instead of segmenting a sample by utterance, Aeneas was tuned to provide word time boundaries. In comparing the delta between the two versions of the data, the intention was to identify with per-word accuracy exactly where edits were made, an example of which is shown in Figure 4.2. In practice, very few if any of the edits made concerned the content of the speech itself. Almost all substantial differences in the length of individual word-timings concerned adding or trimming silence between words or utterances. Most edits to adjust pronunciation of a given word or phrase resulted in a segment of near or almost identical length. As such, the usefulness of the process is non-exhaustive for finding most forms of corrected performance error, but could have utility as a secondary heuristic for identifying error points.

Chapter 5

Conclusions

This thesis realized several useful and informative conclusions. In summary, it is almost certainly possible to create a system for speech performance error detection using minimally modified, open source tools. To be successful, these tools need modifications requiring expertise in the specific software tools involved, and/or in the field of speech recognition and speech comprehension as a whole. These results raise questions about the actual usefulness of such tools, given current advances in other areas of speech recognition and speech synthesis.

5.1 Discussion of Experimental Results

Several tests were performed to determine the suitability of Kaldi and Aeneas to be used as part of an ASR system for detecting mistakes in audio performances. These began with a functional test on how well these systems perform when segmenting and interpreting the quality of an audiobook recording, as it is equally important to identify which parts of the performance are correct as it is to identify mistakes. Further tests were conducted to discover the system's capacity to detect various types of mistakes, as explained in Section 4.2. These tests showed that the tools used had potential in being repurposed for error detection, but that doing so would require time, effort, and greater familiarity with Kaldi and ASR systems in general in order to implement. In parallel with the system tests, output was analyzed from both Kaldi and Aeneas to assess whether there were any identifiable patterns that correlate with the presence or absence of mistakes, and which could potentially be used to assist in the training of an error-detection model.

5.1.1 Effects of Editing and Processing

The comparison between the three audiobook dataset samples with different levels of audio processing (raw, filtered, filtered+edited) using Kaldi demonstrated very little difference between the three in terms of the system's overall word error rate. The error rate decreased by a miniscule amount from the raw sample to the filtered one, and slightly further to the edited sample, but the degree of difference between samples was not significant. As for the strength of word recognition itself, Kaldi identified between 10.3 and 8.2 percent of words in the samples as being incorrectly voiced, depending on method.

In comparison to other known tests using a generic acoustic model, results were slightly improved. Desh Raj's result evaluating the WSJ datasets yielded a slightly higher word error rate than these audiobook samples, with RNNLM results of 11.8% and 9.8%, in comparison to 8.2% on the edited, filtered audiobook dataset. Given the similarity in size between the datasets, with the two WSJ-derived test sets containing 8234 and 5643 words, and the audiobook set's 11020 words, meaningful comparisons can be drawn in the accuracy of the Librispeech model in evaluating both datasets. This is further relevant as all sets involved are comprised of speech from formal writing, either news stories or published books. Despite this, a 90% accuracy rate still falls short of what would be reasonable for a finished error detection system, with an unacceptably high rate of false positives.

5.1.2 Error Detection

Kaldi and Aeneas have both demonstrated a high degree of noise rejection. As it is the intended function of these systems to be able to give the best possible evaluation of a speech sample, despite the presence of noise or poor speech quality, it is entirely logical that they ignore the sorts of non-speech noise or minor mistakes that an error-detecting model would be most interested in. These same characteristics that make them effective ASR systems does significantly hinder their use in performance evaluation without allowing for noise "words" to be recognized or made visible. It is potentially possible for such modifications to be made, as Kaldi at least possesses internal metrics for confidence and accuracy, but these are not exposed by default, and the process of making them available and using them fell outside the scope of this thesis.

5.1.3 Analysis of Segmentation Data

Without confidence metrics, as briefly discussed in the previous section, there is little that can be done to statistically model error points from data on the timing and frequency of words, or on the default outputs from Kaldi's forced alignment. Most of the points in which changes have been made did not change the length or position of word-segments, and most of the instances in which segments significantly changed in length concerned shortening or lengthening of silences and pauses, the detection and adjustment of which would be a sufficiently straightforward task to be handled by a simple algorithm rather than a statistical model.

5.2 Benefits and Drawbacks of Kaldi for Error Detection

Kaldi is a remarkably sophisticated toolset, an implementation of ASR with a range of varied capabilities, and freely available to anyone with the interest in studying it. However, Kaldi was not designed specifically for use as a filter to detect mistakes, and in that role, or indeed, in even its intended role, Kaldi is constrained by a high barrier to entry, requiring expertise in both the Linux operating system and speech systems in general, in order to even begin unlocking the program's potential. Any product or system being provided to an end-user would need radically more streamlining and user accommodation than either of these tools do in their current form, as it is designed for academic use rather than personal operation by someone without personal expertise in ASR systems.

5.3 Future Work

There are other methods by which Kaldi may still be effectively employed for error detection. One potential solution would be to use Kaldi's own confidence scores, or the certainty with which the program is able to guess that a particular combination of phonemes is expressing one word instead of another. While Kaldi does not normally output per-word confidence scores, it is possible to extract these, although gaining the familiariy with Kaldi and ASR necessary to do so was outside the reasonable scope of this thesis. Given this capability, it would be possible to assess whether a particular word has a high degree of confidence (e.g., 95%) or a low degree of confidence (e.g., 55%) and to flag particular words as suspect if their confidence is low. More than a cursory examination of this method was not attempted, due to time limitations. The projects that expose Kaldi's confidence data were too sparsely documented to be considered usable for an off-the-shelf system using free and open-source tools.

Although not done in this work, another approach to explore could be an evaluation of the noise reduction technique used in Audacity to test whether its methods were effective at discriminating signal from noise in an applicable way. Such a system could be particularly useful as a counterpart to Kaldi, detecting the noise and non-speech sounds that are ignored by the other program.

Even during the time spent researching this thesis, great strides have been made in several fields concerning speech recognition and speech synthesis. Starting in January 2023, a service called ElevenLabs [5] offered speech synthesis of arbitrary voices from samples provided by the user. This service is able to reproduce a naturalistic imitation of a voice from a provided prompt. This tool, and others of the same type, could permit an author or publisher to voice an entire novel without spending a single minute recording audio. It is not the same process as evaluating for errors, and the generated speech would still need to be manually checked by a human listener, but the listener would need no special skills or talent in order to fill that role.

5.4 Closing Thoughts

In summary, and to address the core question of this thesis, it is plausible that Kaldi, Aeneas and other open-source software could be used to create an error-detection system, but the question remains as to whether such effort would be justified given available alternatives. Further developments could be made in implementing features such as evaluation by confidence scores in Kaldi. Conceivably, if such a system could be streamlined it could serve as a direct source of training data to create a speaker-dependent error-detection system tailored to that speaker's voice.

Bibliography

[1] Audacity. https://www.audacityteam.org. Accessed 21 June 2023.

[2] Librivox. https://librivox.org/. Accessed 21 June 2023.

[3] Audacity Wiki. How audacity noise reduction works. https://web.archive.org/web/20221112175648/https://wiki.audacityteam.org /wiki/How_Audacity_Noise_Reduction_Works, Jan 2023. Accessed 2 July 2023.

[4] P. Boersma and D. Weenink. Praat: doing phonetics by computer [computer program]. version 6.2.08. http://www.praat.org/, 2022. Accessed 21 June 2023.

[5] P. Dabkowski and M. Staniszewski. Elevenlabs. https://beta.elevenlabs.io/. Accessed 21 June 2023.

[6] R. Desh. How to use the pre-trained librispeech model in kaldi. https://desh2608.github.io/2020-05-18-using-librispeech/, 2020. Accessed 2 July 2023.

[7] J. S. Garofolo, D. Graff, D. Paul, and D. Pallett. CSR-I (WSJ0) Complete. https://doi.org/10.35111/ewkm-cg47. ISBN: 1-58563-030-6.

[8] S. International. Sil international. https://software.sil.org/about/. Accessed 21 June 2023.

[9] B. Juang and L. Rabiner. Automatic speech recognition - a brief history of the technology development. In *Encyclopedia of Language and Linguistics, Second Edition*. Elsevier, Jan 2005.

[10] Linguistic Data Consortium, NIST Multimodal Information Group. CSR-II (WSJ1) Complete. https://doi.org/10.35111/q7sb-vv12. ISBN: 1-58563-006-3.

[11] A. Pettarin. Aeneas. https://www.readbeyond.it/aeneas/, 2017. Accessed 21 June 2023.

[12] M. Pinola. Speech recognition through the decades: How we ended up with siri. https://www.pcworld.com/article/477914/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html, 2011. Accessed 21 June 2023.

[13] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely. The Kaldi speech recognition toolkit. In *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011. IEEE Catalog No.: CFP11SRW-USB.

[14] V. Pratap, Q. Xu, A. Sriram, G. Synnaeve, and R. Collobert. MLS: A large-scale multilingual dataset for speech research. In *Interspeech 2020*. ISCA, Oct 2020.

[15] M. Rubery. *Audiobooks, Literature, and Sound Studies*. Taylor Francis Group, 2011.

[16] D. Spicer. A history of automatic speech recognition. https://computerhistory.org/blog/audrey-alexa-hal-and-more/, 2021. Accessed 21 June 2023.

[17] D. Stewart, M. Casey, and C. Wigginton. The rise of audiobooks and the podcast industry. https://www2.deloitte.com/us/en/insights/industry/technology/technology-media-and-telecom-predictions/2020/rise-of-audiobooks-podcast-industry.html, Dec 2019. Accessed 21 June 2023.

Appendix A

Appendix Placeholder

A.1 Aeneas to Kaldi import script

A Python 3 script written to convert Aeneas .json output into the 'segments' and 'text' files required by Kaldi

```
# importscrpt.py
    from os import listdir


def getJsonFiles():
    filesList = listdir(".")
    jsonFiles = []
    for file in filesList:
        if file [-4:] == "json":
            jsonFiles.append(file)
    return jsonFiles


def chop(filename):
    return filename.upper()+"_"
```

```
def processFiles(jsonList,speaker):
    outputText = open("text",'w')
    outputSegments = open("segments",'w')
    jsonList = getJsonFiles()
    for file in jsonList:
        txt, seq = processSingleFile(file,speaker)
        for line in txt:
            outputText.write(line)
        for line in seq:
            outputSegments.write(line)
    outputText.close()
    outputSegments.close()


def processSingleFile(filename,speaker):
    #open the file, trim the first two lines and the last two lines
    fileDump = open(filename,'r').readlines()[2:-2]
    intervals = range(0,len(fileDump),10)
    txtOutput = []
    seqOutput = []
    for value in intervals:
        startTime = fileDump[value+1][13:-3]
        endTime = fileDump[value+3][11:-3]
        rawId = fileDump[value+4][10:-3]
```

```
utt = fileDump[value+7][5:-2]

utt = utt.replace('\u2013','')

utt = utt.replace('\u2018','')

utt = utt.replace('\u2014','-')

utt = utt.replace(':',' ')

utt = utt.replace(';',' ')

utt = utt.replace(' \'',' ')

utt = utt.replace('\'.',' ')

utt = utt.replace('%',' PERCENT')

utt = utt.replace('\n\'','\n')

utt = utt.replace(',',' ')

utt = utt.replace('.',' ')

utt = utt.replace('!',' ')

utt = utt.replace('?',' ')

utt = utt.replace('(',' ')

utt = utt.replace('[',' ')

utt = utt.replace('{',' ')

utt = utt.replace(')',' ')

utt = utt.replace(']',' ')

utt = utt.replace('}',' ')

utt = utt.replace('  ',' ')



txtOutput.append(chop(filename[:-5])+rawId.upper()+" "+utt.upper()+"\n")
```

```
        seqOutput.append(chop(filename[:-5])+rawId.upper()+" "+filename+" "+startT
    return txtOutput,seqOutput

def main():
    processFiles(getJsonFiles,"Leon Tietz")

main()
```

A.2 Aeneas to CSV time alignment comparison

A Python 3 script written to convert two Aeneas .json output files into a CSV that compares the time data between them.

```
    from os import listdir

def getJsonBFiles():
    filesList = listdir(".")
    jsonBFiles = []
    for file in filesList:
        if (file [-4:] == "json" and file[0] == "B"):
            jsonBFiles.append(file)
    return jsonBFiles

def getJsonAFiles():
    filesList = listdir(".")
    jsonAFiles = []
    for file in filesList:
        if (file [-4:] == "json" and file[0] == "A"):
            jsonAFiles.append(file)
    return jsonAFiles


def chop(filename):
    return filename.upper()+"_LEN_"


def processFiles():
    outputDiff = open("lengthAB.csv",'w')
    jsonBList = getJsonBFiles()
    jsonAList = getJsonAFiles()
    count = range(len(jsonBList))
    outputDiff.write("ID,LenB,LenA,StartB,EndB,StartA,EndA,BMinusADiff,Word,\n")
    for x in count:
        diff = processFilePair(jsonBList[x],jsonAList[x])
        for line in diff:
            outputDiff.write(line)
    outputDiff.close()


def processFilePair(filenameB,filenameA):
    #open the file, trim the first two lines and the last two lines
    fileDumpA = open(filenameA,'r').readlines()[2:-2]
    fileDumpB = open(filenameB,'r').readlines()[2:-2]

    intervals = range(0,len(fileDumpA),10)

    diffOutput = []

    dash = ","

    sdash = ","


    for value in intervals:

        startTimeB = fileDumpB[value+1][13:-3]

        endTimeB = fileDumpB[value+3][11:-3]

        startTimeA = fileDumpA[value+1][13:-3]

        endTimeA = fileDumpA[value+3][11:-3]

        rawId = fileDumpB[value+4][10:-3]

        utt = fileDumpB[value+7][5:-2]


        lenA = str("%.2f" % (float(endTimeA)-float(startTimeA)))

        lenB = str("%.2f" % (float(endTimeB)-float(startTimeB)))

        deltaLen = str("% .2f" % (float(lenB)-float(lenA)))

        bracketB = str("%.2f" % float(startTimeB)).zfill(6) + "," + str("%.2f" % f

        bracketA = str("%.2f" % float(startTimeA)).zfill(6) + "," + str("%.2f" % f


        diffOutput.append(chop(filenameB[1:-5])+rawId.upper()+ "," + lenB + sdash

    return diffOutput


def main():
    processFiles()


main()
```

A.3 wav.scp

An example of a wav.scp file for a Kaldi dataset, which associates the names of various sample sources with the on-disk locations of that audio data.

```
ASAMPLE01 /home/*acct_name*/kaldi/egs/librispeech/s5/audio/Sample01A.wav
ASAMPLE02 /home/*acct_name*/kaldi/egs/librispeech/s5/audio/Sample02A.wav
ASAMPLE03 /home/*acct_name*/kaldi/egs/librispeech/s5/audio/Sample03A.wav
ASAMPLE04 /home/*acct_name*/kaldi/egs/librispeech/s5/audio/Sample04A.wav
ASAMPLE05 /home/*acct_name*/kaldi/egs/librispeech/s5/audio/Sample05A.wav
```

A.4 utt2spk

An example of an utt2spk file for a Kaldi dataset, indicating which utterances belong to which 'speaker' or sample audio file.

```
ASAMPLE01_F000001 ASAMPLE01
ASAMPLE01_F000002 ASAMPLE01
ASAMPLE01_F000003 ASAMPLE01
ASAMPLE02_F000001 ASAMPLE02
ASAMPLE02_F000002 ASAMPLE02
ASAMPLE02_F000003 ASAMPLE02
```

A.5 Misread song lyrics transcript

An alteration of the song ‘House of the Rising Sun’, read as prose, intentionally altered to evaluate Kaldi’s ability to detect mispronounced words.

```
Where is a house in gnu oar leans

They fall the resin slum

And it's been the shoe in of Kenny a poor boy

And Cod, I know I'm done


My other was a tail hair

She showed my new glue dreams

My farther was a ramblin' man

Clown in New Orleans


Now the only king a clamber leads

Is a suit race and a truck

And the only time he'll be stratified

Liz when he's all trunk


Ohm, other, tell your million

Not to do what I have gun

Spend your lives in skin and mister he

In the grouse of the Rising Sun
```

Well, I got one soot on the flat for em

The other foot on the train

I'm going back to gnu oar leans

To swear that spall and claim

Well, there is a house in New Orleans

They call the Rising Sun

And it's bin the rune of many a poor boy

And God, I gnaw I'm one