Imperial College London

Final year MEng individual project

A combined ASR and VAD approach to long audio alignment in movies

Author:
Jonathan Roitgrund

Supervisor:
Dr. Bjoern Schuller

Abstract

Aligning audio to text has many applications ranging from increasing accessibility by automatically creating subtitles for scripted content (movies, webseries, etc) to the creation of voice corpora for acoustic model training. Few existing tools perform a complete unsupervised alignment.

Existing approaches focus solely on decoding the audio. By using advanced speech recognition with voice activity detection I am able to improve the quality of alignments when the audio is hard to correctly transcribe.

I build a fully fledged alignment system combining the traditional approach to long audio alignment with a voice activity detection based alignment. While the results are extremely affected by noise and the movie alignments are largely erroneous, this is due to lackluster performance from the speech recognition component, and the approach is still shown to be sound.

Acknowledgements

I would like to thank my supervisor Dr. Bjoern Schuller. His ASR tool, his enthusiasm for the project, and his guidance with some of the concepts behind ASR were all very helpful. In addition, his students Florian Eyben and Felix Weninger were kind enough to suggest a few ideas to get me started. I'd also like to thank Dr. Daniel Povey, lead developer of the Kaldi ASR project, and Vassil Panayotov, for their precious help with understanding Kaldi's intricacies.

Contents

1 Introduction **3**
- 1.1 Motivation . . . 3
- 1.2 Related work . . . 4
- 1.3 Objectives . . . 5
- 1.4 Contributions . . . 5
- 1.5 Summary of the report . . . 5

2 Background **7**
- 2.1 Advanced speech recognition . . . 7
  - 2.1.1 Basics . . . 7
  - 2.1.2 Front-end features . . . 8
  - 2.1.3 The HMM-GMM approach to acoustic modeling . . . 8
  - 2.1.4 Language modeling . . . 9
  - 2.1.5 System selection . . . 10
- 2.2 Basics of Kaldi's approach to ASR . . . 11
- 2.3 Long audio alignment . . . 11
  - 2.3.1 Text-to-text alignment . . . 12
- 2.4 Voice activity detection . . . 12

3 Procedure and implementation **15**
- 3.1 Data preparation . . . 15
- 3.2 Voice activity detection . . . 15
  - 3.2.1 Segmentation . . . 16
  - 3.2.2 Channel voicedness . . . 16
- 3.3 Time-based alignment . . . 16
  - 3.3.1 Method outline . . . 16
  - 3.3.2 Text length . . . 17
  - 3.3.3 Alignment scoring . . . 17
  - 3.3.4 An efficient algorithm to compute the alignment . . . 17
- 3.4 Speech recognition . . . 19
  - 3.4.1 Training . . . 19
  - 3.4.2 Feature generation . . . 19
  - 3.4.3 Language modeling . . . 20

3.5 Putting it all together . . . 22
3.5.1 Using the VAD-alignment to restrict the language model . 22
3.5.2 Compiling results . . . 23

4 Development practices **24**
4.1 Programming language . . . 24
4.2 Logging and debugging . . . 24
4.3 Testing . . . 25
4.4 Graphviz . . . 25
4.5 Version control . . . 25

5 Experiments and evaluation **26**
5.1 Test data . . . 26
5.2 Experiments . . . 26
5.3 Evaluation . . . 27
5.4 Selected results . . . 27
5.4.1 Movies and series . . . 27
5.4.2 Parameters . . . 29
5.4.3 Audiobooks . . . 29
5.5 Analysis . . . 29
5.5.1 Factor automaton . . . 29
5.5.2 Poor speech recognition performance . . . 29
5.5.3 VAD-only passes yield low maximum error . . . 30
5.6 Comparison with other tools . . . 30
5.6.1 SAIL . . . 30
5.6.2 YouTube . . . 30
5.6.3 Proof-of-concept results using YouTube for speech decoding 31

6 Conclusion and future work **32**
6.1 Achievements . . . 32
6.2 Unrealized expectations . . . 32
6.3 Future work . . . 33

7 Appendix **34**
7.1 External programs and libraries . . . 34
7.2 Distribution contents . . . 35
7.3 Running the aligner . . . 35
7.3.1 Kaldi . . . 35
7.3.2 Preparing the data . . . 35
7.3.3 Running the aligner . . . 36

Chapter 1

Introduction

1.1 Motivation

Often times audio or video material is produced from a pre-existing script. Whether it is a movie, a user-submitted clip on a video sharing website such as YouTube, or a lecturer at a university, the speakers are enunciating material that has been prepared in advance, and mostly sticking to the script. It seems natural to think that having access to an accurate transcript should make subtitling the audio an easy task. On the contrary, it is often a long, tedious process done manually using tools such as the one pictured in Figure 1.1

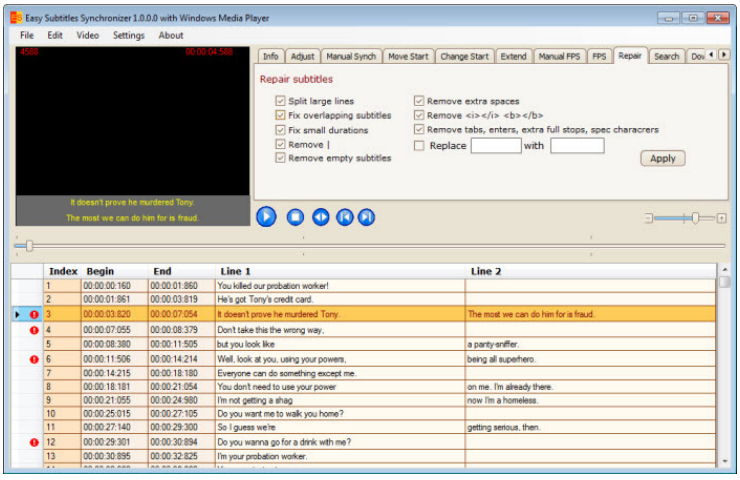

Figure 1.1: Typical software for manual subtitle alignment

In addition to saving time, a solution to the audio-to-transcript alignment task

would benefit accessibility. Many lower-budget producers, for instance those working with online media, do not have the time to manually create aligned subtitles for all their content. If the process could be automated, deaf users would be able to enjoy the material at no extra cost to the producer.

Another huge benefit of a reliable long audio alignment tool is its application to the creation of voice corpora for acoustic training in an automatic speech recognition system. The only freely available speech database is Voxforge, which depends entirely on user submitted content. Long audio alignment could produce usable corpora from long, unaligned audio data with known transcripts such as the LibriVox audio recordings of copyright-free literature from Project Gutenberg. At the time of writing, LibriVox contains recordings for 7781 different works, many of which last several hours. These are read by many different speakers and in many different languages.

The current approaches to automatic alignment focus exclusively on attempting to recognize the speech in the audio, biasing the recognizer towards preferentially recognizing words in the transcript. This is a solid, proven approach for easily recognizable speech such as that found in voice corpora or an amphitheatre lecture recording. When it comes to noisier environments such as films however, a substantial amount of speech is likely to be decoded with a very high word error rate, making the approach less feasible.

Noise robust automatic speech recognition is a hard problem. Instead, in this project, I investigate the suitability of combining the existing approach with one that is more robust to noise: voice activity detection. Accurate information about which segments of the audio are voiced can be used both to improve the performance of the automatic speech recognition system by narrowing down the candidate text segments for each audio segment, and to align text when audio segments cannot be successfully decoded to utterances in the transcription.

1.2 Related work

No prior long audio alignment implementation has made use of noise-robust voice activity detection as a means of improving its results. A primitive energy-thresholding based VAD system such as that implemented in the program "sox" is usually used to segment the audio into the parts that will be individually decoded by the automatic speech recognition system. However, no timing information is retained from the VAD pass, either for alignment or for narrowing down the parts of the transcript that are likely to correspond to each audio segment.

The only publicly available implementation of a long audio alignment system is SailAlign [6], a 2011 project implementing a state of the art version of the canonical long audio alignment solution, reporting good results on alignments of the TIMIT speech corpus and using HTK as the speech recognition system.

YouTube also performs automatic alignment of user-provided captions, but there is no publicly available information as to the methods used to perform the alignment.

1.3 Objectives

The goal of my project is to develop a fully automated solution for aligning an accurate transcript to a audio track. The system should work off-line (with no real-time constraint) and produce accurate alignments with noisy audio such as that found in a movie. The expected benefit is simply a higher accuracy compared to the traditional method of using only speech decoding information for alignment.

1.4 Contributions

During this project I developed an implementation of a complete long audio aligner for video files. The software takes as input a video and a subtitle file, strips the subtitle file of timing information, and attempts to align it automatically, evaluating its performance by comparison with the timings from the original subtitle file. The implementation consists of:

- Triphone acoustic models trained on the Voxforge audio corpus
- Language model generation from transcript excerpts
- A speech recognition system built from the above two components
- An algorithm for the alignment of subtitles based on the results of voice activity detection

1.5 Summary of the report

Chapter 2 introduces you to the necessary background for a complete understanding of the project. After a brief explanation of the modern HMM-GMM approach to advanced speech recognition, and in particular Kaldi's WFST implementation, I give a short history and comparison of voice activity detection systems and detail existing approaches to long audio alignment.

The individual components of the long audio aligner are described in detail, both in their design and in their implementation, in Chapter 3. Chapter 4 adds some detail about the development process.

Chapter 5 presents and analyzes the results of various experiments performed on a range of audio data. Chapter 6 concludes with some ideas on extending

and improving the long audio alignment system. The appendix in chapter 7 includes information about running the aligner and external dependencies.

Chapter 2

Background

2.1 Advanced speech recognition

2.1.1 Basics

Advanced speech recognition is the process of decoding a voiced audio segment to a textual transcription of the spoken words.

The fundamental equation of statistical speech recognition is

$$W* = \arg\max_W P(W|X)$$

.

$W*$ is the output of the speech recognizer, the most likely word sequence. Each W is a guess at the correct transcription, X is the audio data, and $P(W|X)$ is the probability of a word sequence given the audio data. In short, we are looking for the sequence of words that is most likely given the audio.

While this seems like a tautology, it is the basis for the way speech recognition works. Applying Bayes' theorem we get

$$W* = \arg\max_W P(X|W)P(W)$$

$P(X|W)$ is the acoustic model: it represents the likelihood of the audio data for a given transcription. $P(W)$ is the language model: it represents how likely each word is to appear in the transcript.

In practice, as seen in Figure 2.1, speech recognition systems work at the feature and phone level. A set of numerical features is produced from the audio. The acoustic model represents each phone in a way that it can calculate the probability of the observed features being generated when that phone is spoken.

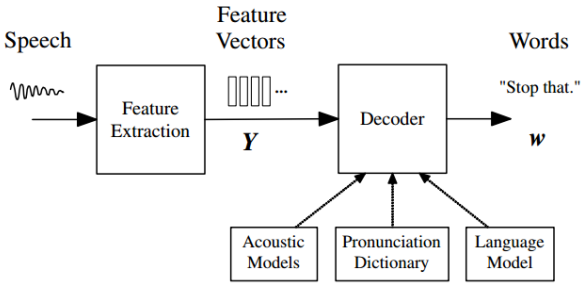

Figure 2.1: The fundamental approach to speech recognition

2.1.2 Front-end features

A set of 13 Mel-frequency cepstral coefficients is typically used as the front-end for speech recognition. The audio is divided into frames and for each frame a feature vector is computed representing the value of each of the 13 coefficients for that frame.

General purpose acoustic models generally yield very poor performance if they are used as is on the MFCC feature set. They tend to be too speaker dependent and word error rates increase dramatically when attempting to recognize other speakers. In supervised environments, acoustic models can be adapted to the new speakers on a very small data set. However, in a completely unsupervised environment, feature-space transformations are the best way to make the recognizer more generic. For instance, Gales [4] shows a significant increase in speech recognition accuracy obtained by performing feature space maximum likelihood linear regression.

2.1.3 The HMM-GMM approach to acoustic modeling

Modern approaches to speech recognition are mostly based on the HMM-GMM approach as described in Gales [3].

A Hidden Markov Model is a way of representing an occurrence as a sequence of states, with probabilities for transitioning from each state to the next, and probabilities of each state generating a certain output. The states are invisible to the observer, who only sees the outputs. Events, consisting of a sequence of outputs, are matched to the most likely model. In Figure 2.2, $X1$, $X2$, and Xn are the states.

In the case of speech recognition, each HMM is a phone, typically with three states (begin, middle, and end). Its outputs are feature vectors matching those extracted from the audio data. For each feature, the output probability is mod-

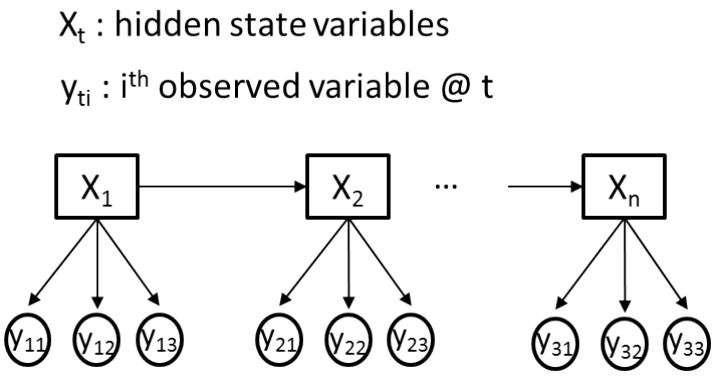

Figure 2.2: A hidden markov model

eled as a Gaussian Mixture Model (ie a weighted set of normal distributions). Each y output in Figure 2.2 is a GMM, and there is one output for each acoustic feature.

Acoustic models are trained by using the expectation-maximization algorithm to find the most likely parameters for the Gaussian given the training data.

A dynamic programming algorithm such as Viterbi decoding or beam search is used to optimize the search for the most likely phone given a sequence of feature vectors.

2.1.4 Language modeling

Language models contain information about the words we expect the decoder to produce. They are built from a corpus of text, and reflect the frequency with which words and phrases occur in that corpus. This lets us bias the recognition towards words and phrases we expect.

The typical approach to language modeling uses n-gram back-off models as described by Katz [7]. An n-gram model gives probabilities for sequences of n words. A back-off model also stores probabilities for shorter sequences (of length 1 to n) such that probabilities for sequences that don't occur in the training corpus can still be computed. The n-gram back off model is therefore the union of x-gram models for x from 1 to n. Higher accuracy models can be used for long word sequences that occurred in the corpus and that we have probability information for, but lower accuracy models are used as backup for sequences unseen in the corpus.

The most common n-grams are trigrams, because increasing the n-gram count usually results in a poor tradeoff in performance loss versus increased recognition accuracy.

Language modeling is of particular import to long audio alignment. While acoustic models are general purpose, pre-trained on large voice corpora, language models can be built specifically for each alignment task by using the transcript as a corpus.

Hazen's [5] approach uses a language model built from a combination of a general purpose english language model and the transcript. The idea is to compensate for imperfect accuracy in the transcript and improve the speech decoder's performance by allowing for errors in transcription. The part built from the transcript is much more heavily weighted, but the general purpose model allows recognition of words omitted from the transcript.

Moreno [12] shows the benefits of using factor automata for the language model. Factor automata are simple state-machine representations of strings. A factor automaton built from a string s accepts any substring of s. Using factor automata to build the language model is a way of taking advantage of the fact that the transcripts incorporate information about word ordering. Language modeling using an n-gram model, by definition, discards information about word sequences longer than n.

Liu, Shriberg, and Stolcke [8] list three types of common disfluencies in speech: repetition, revision, and restarts. While restarts (an utterance begins but it is left hanging and restarted differently before the end of the sentence) are usually present in a transcript, repetition and revision, both very common in movies (a character stuttering out of fear, or correcting himself to avoid being caught lying) are often left out. These disfluencies should be taken into account when building factor automata.

2.1.5 System selection

There is a wealth of competing speech recognition systems, the most prominent being CMUSphinx, HTK, and Kaldi.

Morbini [10] compares the performance of several state-of-the-art speech recognition systems. Word error rates are tallied for six different speech corpii. Kaldi offers significantly better performance than Sphinx. Google's cloud-based system is of little interest because it does not offer language model customization. While AT&T's speech mashup service offers promising quality and the ability to customize language models, it has no option to store audio segments online for repeated decoding, which means every pass and experiment would require the audio to be uploaded to the AT&T servers, which is clearly unfeasible.

The Kaldi [14] paper compared Kaldi's performance to that of its main competitor HTK. The word error rates are similar on the Resource Management and Wall Street Journal corpii, but the paper outlines a number of Kaldi's benefits. Kaldi's entire decoding graph construction is based on FSTs and the OpenFST toolkit. This makes experimenting with custom language models such as the

factor automata described above much easier. In addition, Kaldi comes with a wealth of example scripts and recipes, and is very modular and pluggable (about 90 different binaries, each with very specific functionality).

2.2 Basics of Kaldi's approach to ASR

Kaldi is based on the Weighted Finite-State Transducer approach to speech recognition outlined in Mohri [9]. Weighted Finite-State Transducers are state machines with weighted transition probabilities, where each transition has both an input and an output. Every part of the speech recognition decoding pipeline can be modeled as a WFST.

The acoustic model is a set of HMMs, and each HMM is a finite state machine. The WFST produced from the acoustic model is therefore simply the union of all the finite state HMMs, with features as input labels and phones as output labels.

The lexicon maps phone sequences to words. Each word is a linear acceptor of phones, and the WFST is the union of the linear acceptors. The input labels are phones and the output labels are features.

Finally, the grammar, be it a factor automata or an n-gram backoff model, is already a weighted finite state acceptor. Since its role is simply to weight certain word sequences above others, its input and output labels are always identical, ie it does not modify its input, it simply gives it a probability score.

Each WFST's output corresponds to the next WFST's input: this means we can compose them into one decoding graph that turns feature vector sequences into word sequences and probabilities. The graph is determinized and minimized as a performance optimization.

2.3 Long audio alignment

Long audio alignment is the main problem the project aims to solve.

The state of the art approach is still based on Moreno's original 1998 work [13] and his recursive algorithm using anchor points. As illustrated in Figure 2.3, a language model is built from the transcript and used for a first-pass speech recognition run on the audio. The result of the speech recognition run is optimally aligned to the transcript using dynamic programming. Long sequences of matches between the transcript and the ASR output indicate a high probability of correct ASR output. These long sequences are therefore retained as fixed anchors where the audio is considered to have been successfully matched to the transcript. The algorithm is then run recursively on the segments in between

the anchors with a now reduced language model built only from the parts of the transcript in between the anchors.

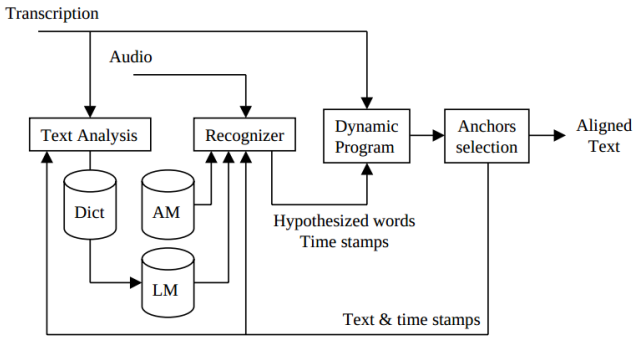

Figure 2.3: An illustration of Moreno's recursive approach to long audio alignment

SailAlign [6] provides a complete implementation of Moreno's algorithm written in Perl and using the HTK speech recognition system with an acoustic model trained on the Wall Street Journal corpus.

2.3.1 Text-to-text alignment

The decoded text output by the speech recognition system must be aligned with the transcript and scored. High scoring segments are then preserved as anchors where we are confident the text segment is correctly aligned, and the problem is reduced to the smaller problem of aligning the text in between two anchor points. The canonical alignment and scoring tool is sclite[1], part of the NIST scoring toolkit.

2.4 Voice activity detection

Voice activity detection consists in discriminating between voiced segments and silence segments in an audio sample, where voiced segments are those containing human speech. VAD is commonly used to pre-process audio to be analyzed by an automatic speech recognition system (for instance by dropping unvoiced frames). It is also important to the performance of speech compression codecs, as it enables them to discard unvoiced segments.

VAD is an important foundation of my approach to subtitle alignment. Its role is twofold. Firstly, I can use the VAD data to isolate the audio segments I will be

[1] http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm

performing ASR on. Secondly, knowing where the speech segments are enables me to carry out a first-pass alignment of the movie script with the audio, based solely on the duration of the speech segments and the length of the candidate subtitles. VAD is also useful in that it enables me to select the channel of audio data containing the most speech. In recent movies where speakers are clearly heard from a particular direction, choosing the right channel could provide a significant improvement in the cleanliness of the audio data to be used for ASR and syllable counting. In short, accurate output from our VAD system is an essential component of my approach, and the accuracy of the alignment as a whole is affected by that of the VAD.

Audio corpora for VAD evaluation usually include samples with a variety of background noise. For instance, the spanish SpeechDat-Car database [11] is split into three types of noise conditions: quiet, low, and high. This is because one of the key aspects of VAD is its ability to maintain its accuracy irrespective of the noise in the signal. This also means that a good VAD system should be able to work through unstationary noise such as babble or music-type noise. Evaluation data often has noise samples artificially added to it [2]. In short, when comparing candidate systems for VAD, it is important to pay particular attention to their performance in a variety of noisy environments.

g.729 [1] is a compression algorithm for speech which includes a VAD implementation. It measures the full-band energy, the low-band energy, and the zero-crossing rate of the audio signal, and uses a simple heuristic comparing their values to the observed averages on the full length signal to discriminate between speeched and voiced segments. Its weakness is apparent from its simplicity: it depends on the underlying noise to be stable and unchanging.

Sohn's [16] method uses a likelihood ratio test where the hypotheses are as follows:

$$H_0 : X = N$$
$$H_1 : X = S + N$$

The null hypothesis H_0 is that the signal X only has a noise component N, and the alternative hypothesis H_1 is that the signal also has a speech component S. In addition, Sohn uses a soft decision-based method to learn the characteristics of the noise signal which is able to quickly pick up on changes in the background noise, instead of assuming that change is an indicator of speech. Because of this, his method is able to update the noise spectrum during short silence intervals. This is vital for non-stationary noise, as demonstrated by the increased accuracy of Sohn's method on time-varying babble-type noise.

Recent approaches to VAD focus on using longer-term information from the audio sample to more accurately track changes in background noise [15]. However, all of the above approaches use simple heuristics.

One approach [2] developed by Eyben, favors a data-driven approach over heuristics. Long short-term memory recurrent neural networks are trained on audio data from the TIMIT corpus with various noise types added. This provides the ability not only to use past information from the audio sample, but also to determine precisely which past context is relevant to the classification of the current frame. This approach is ideal for movies given the likelihood of recurring locations with similar background noise. The difference in results compared to Sohn's and Ramirez's approaches is most significant with non-stationary noise such as music and babble.

Chapter 3

Procedure and implementation

3.1 Data preparation

I use a number of utilities to quickly prepare feature films such that I can run experiments on them with the alignment tool.

I use a simple clipping script to prepare short movie clips for analysis. It uses ffmpeg to extract video segments, and it generates a new subtitle file containing only the relevant subtitles for the clip, all of which are shifted backwards such that they are synchronized with the clip.

Subtitles are automatically stripped of anything that isn't simply an english language word spoken in the audio. This includes music lyrics, symbols indicating the character is offscreen, HTML formatting tags, quotes, symbols indicating a change in speaker, and punctuation.

3.2 Voice activity detection

The long short-term memory recurrent neural network voice activity detection system is run on the entire audio file. Its output is a CSV file containing a likelihood score from -1 to 1 for each centisecond of the audio data. A high likelihood score indicates a high probability of human speech.

3.2.1 Segmentation

The importance of correct segmentation is two-fold. In addition to the segments being used as input to the speech recognition system, the segment lengths are added up and used to calculate the total voiced duration and therefore the speech rate. Correct estimation of the speech rate is essential to ensure an accurate first pass alignment.

The voice activity detection tool outputs segments as well as voicedness likelihood, but after some empirical testing I found that the segmentation was more often than not leading to very incorrect estimations of total voiced time and speech rate. I therefore wrote my own segmentation tool using the voicedness likelihoods as input. Any segment longer than a certain minimum duration during which all the likelihood values are above a certain threshold is considered to be a separate utterance. The values for the threshold and minimum duration are inputs to the alignment tool, and I run experiments to find the best values.

3.2.2 Channel voicedness

Instead of mixing together the stereo channels of the audio, I split it up into individual components. The voice activity detection is run separately on each component. For each segment lasting from t_1 to t_2, I sum the voicedness likelihoods from t_1 to t_2 and extract the segmented audio from the channel with the highest total voicedness.

3.3 Time-based alignment

3.3.1 Method outline

Voicedness likelihood scores are used to align the audio and the text in a first pass without having to decode the audio.

The audio is a series of sequences of voiced segments and non-voiced segments. The transcript is a series of text segments that must be placed, in order, at the correct time. The correct alignment is the one that maximizes the overlap between the text segments and the voiced segments in the audio. Given perfectly accurate data about which segments are voiced and the length of each text segment, this alignment would be perfectly correct. In practice, both pieces of data are simply estimations, which is why the long audio aligner also requires a speech decoding component to function correctly.

3.3.2 Text length

The transcript must be broken up into contiguous text segments, where 'contiguous' means that the whole segment will be spoken at once, without pause. In principle, one would use sentence boundaries, marked by punctuation, to split the transcript up into segments. However, for evaluation purposes, I use the same segments as those in the original subtitles, such that the error can be calculated as the sum of the offsets from the original subtitles.

I then need to estimate the length of each individual text segment, that is the expected time the speaker takes to pronounce the full utterance. Speech rates are most accurately measured using syllables as a base pronunciation unit, since words and letters tend to vary much more in the time they take to pronounce. Speech rate can be estimated by counting the number of syllables in the transcript and dividing it by the total voiced time in the audio (computed from the segmentation described above). The length of each segment can then be computed by multiplying the number of syllables by the speech rate.

Syllable counting is done using the Moby hyphenation dictionary [18]. Each word is looked up in the hyphenation dictionary and its number of syllables is given by the number of hyphenation boundaries plus one. Out-of-dictionary words use a fallback value based on the number of letters in the word.

3.3.3 Alignment scoring

An alignment, represented as a list of start times for each text segment, can be scored against the VAD output. For each text segment, we sum the likelihood scores from its start time to its end time. The total alignment score is the sum of the score of each text segment.

For instance, given the following VAD scores:

t	1	2	3	4	5	6	7	8	9	10	11	12
vad	-0.8	-0.5	-0.4	-0.3	-0.2	0.3	-0.1	-0.2	0.7	0.8	0.6	0.5

a text segment of length 3 placed at time 2 would score $-0.5-0.4-0.2=-0.11$

Placing the subtitles in a way that maximizes the total alignment score yields a strong estimate of their correct locations.

3.3.4 An efficient algorithm to compute the alignment

The brute force solution (try and score every alignment and pick the best scoring one) has exponential time complexity, and given the large number amount of data it is intractable.

I first experimented with alpha-beta pruning before settling on a dynamic programming algorithm.

The first part, shown in algorithm 1, iteratively generates a table containing a partial maximum score. Each iteration i computes a vector of maximum achievable scores using only the i first text segments. Each element n of the vector is the maximum achievable score when placing the i segments such that none of them extend beyond time n. The main logic is in the inner loop on lines 6 and 7. The maximum score achievable by placing segment i at n is the sum of the value in the table for $(i-1, n)$ and the score obtained from segment i itself. To find the value for (i, m), we simply find the value of n such that $n + length(i) < m$ which maximizes this score.

Once the maximal score is calculated, the actual alignment is calculated by backtracking through the table as shown in algorithm 2. The required score is initialized with the maximum value in the table (that obtained by using all segments spread over as much audio as is available), and I start by aligning the very last text segment. Segment i is positioned at time n such that the value in the score table at $(i, n + length(i))$ matches the required score. Segment i's score is then subtracted from the required score, and we repeat the process for segment $i - 1$.

```
1 for m ← 0 to AudioLength do
2 |  F[0][m] ← 0;
3 end
4 for i ← 1 to NumSegments do
5 |  for m ← 0 to AudioLength do
6 |  |  NMax ← m−Len(i);
7 |  |  F[i][m] ← Max(F[i − 1][n]+ Score(i, n))∀n < NMax;
8 |  end
9 end
```

Algorithm 1: Calculate maximum scores using a limited amount of time and text segments

```
1 i ← F[NumSegments −1];
2 ReqScore ← F[NumSegments −1][AudioLength −1];
3 for m ← 0 to AudioLength do
4 |  EndPosition ← arg min_n F[i][n] = ReqScore;
5 |  Position ← EndPosition −Len(i);
6 |  ReqScore leftarrow ReqScore − Score(i, Position [i]);
7 end
```

Algorithm 2: Backtrack through the score table to obtain an alignment

Despite reducing the time complexity to polynomial time, the algorithm still runs very slowly because each call to the scoring function has linear time com-

plexity relative to the length of the segment being scored. To alleviate this, I further improve performance is further improved by precomputing the results of the score function. For each time t, the sum of activations before t is stored in a list, such that the sum of activations from t_1 to t_2 is simply before(t_2) − before(t_1), and the scoring function becomes a constant time lookup.

3.4 Speech recognition

3.4.1 Training

Acoustic models are trained on the freely available Voxforge corpus. Several passes are performed, with the alignments from the previous pass being incrementally used as input to the current pass. I start by training monophone models, then two passes of triphone models. All three of these use the standard dimension 13 MFCC feature vector plus its first and second order derivatives, yielding a dimension 39 vector.

The alignments from the last pass are used to accumulate statistics for linear discriminant analysis. Linear discriminant analysis produces a feature transformation matrix (40 by 13). The transformation adds 27 features to the 13 MFCC features. Each of these 27 features is a linear combination of the 13 MFCC features that has been calculated to maximize discrimination between HMM model states. From now on the LDA features are used rather than the first and second order derivatives.

A final discriminative training pass uses the MMI criterion to train the models. A denominator lattice is generated for each utterance in the test set. Rather than containing a single best-path through the decoding graph, the denominator lattice contains all the possible passes. Since the other paths in the lattice come from models that are incorrect but whose observed outputs are close to the features being decoded, we can now alter the incorrect models such that their observed outputs are further from the decoded features, and the correct models such that their observed outputs are closer.

The resulting model is the one used for all decoding in my tool.

3.4.2 Feature generation

Features are generated for all segments during the data preparation phase. When computing features, Kaldi can output both the actual features and a script file that gives the index of each segment's features in the feature file. Running the decoder on a subset of segments is therefore as simple as extracting the relevant lines from the script file using sed. Cepstral mean variance and

normalization, followed by the linear discriminant analysis transformation, is applies to the MFCC features.

3.4.3 Language modeling

Lexicon

The lexicon is generated using the CMU pronunciation dictionary. For out-of-dictionary words, I use Sequitur G2P to generate pronunciations.

Grammar

The alignment tool is able to use two different ways of building the grammar.

The first simply uses the SRILM [17] toolkit to build a trigram model. Since the models are generated from small corpii containing only a few candidate subtitles, Witten-Bell smoothing is used. Witten-Bell smoothing increases the backoff probability to account for the fact that a small corpus may yield poor results when there are some errors in transcription.

The second is a custom-built grammar generator that builds a weighted finite state transducer representing the union of factor automatons for each sentence in the corpus.

Figures 3.1 and 3.2 respectively represent a trigram model and factor automaton representation of a corpus consisting of the two sentences “I like bread”” and “Bread is good”.

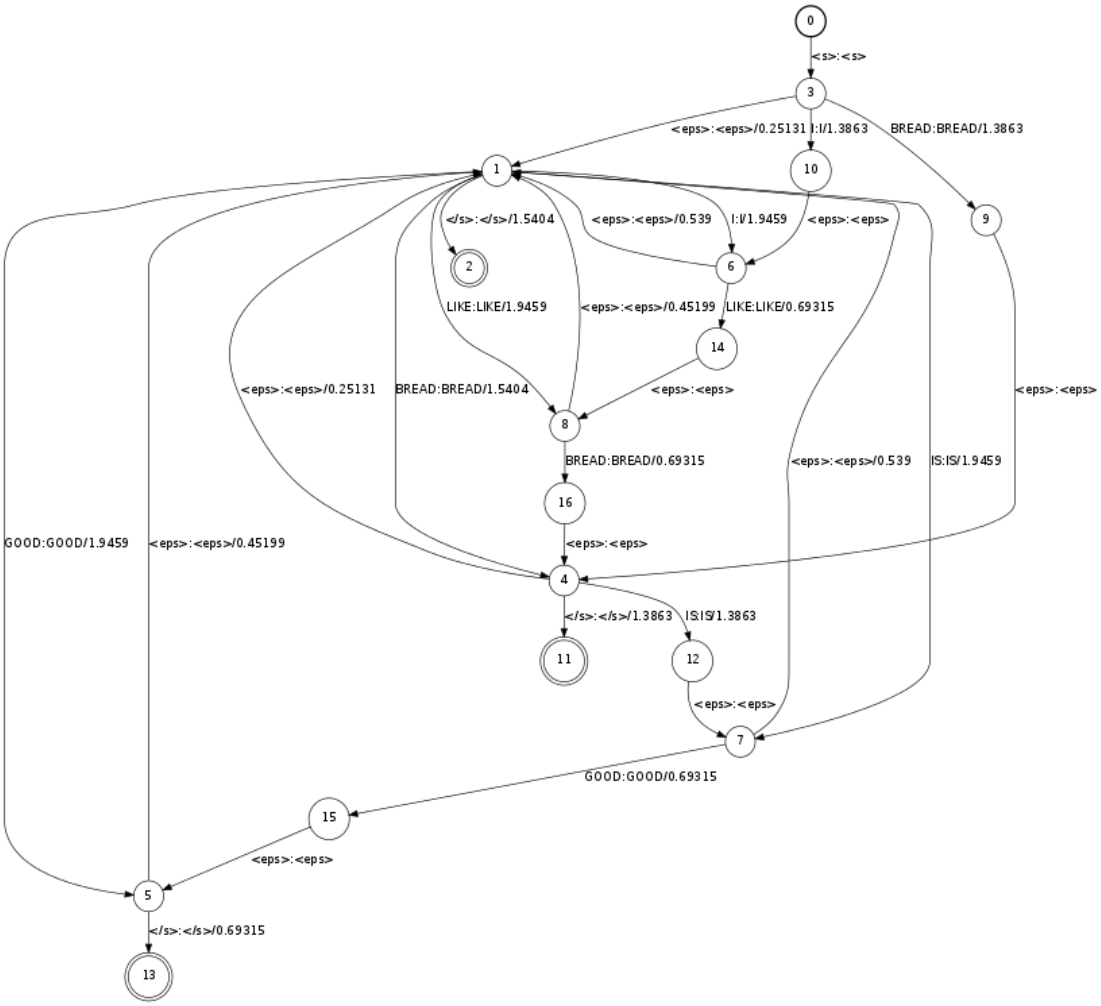

Figure 3.1: Trigram model

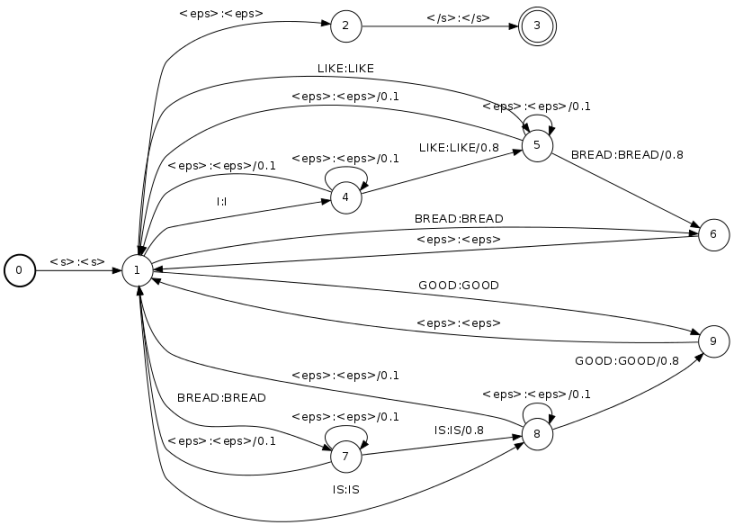

Figure 3.2: Factor automaton

The factor automaton is built as follows. I start with four states: a start state, an utterance beginning state, an utterance end state, and a final state, plus one state for each word in each sentence of the corpus.

The start state is connected only to the utterance beginning state. The utterance beginning state is connected with equal probability to all the word states, to account for speech starting in the middle of a sentence. The utterance end state is connected with equal probability to the final state and to the utterance beginning state, so that several utterances can be decoded.

Each word state has three transitions. One is to the utterance end state, to account for utterances being cut short or the decoder failing to recognize the last part of a sentence. One is a self-loop that accounts for omissions in the transcript. The last is a transition to the next word in the sentence. The two former transitions are weighted lower so as to bias the recognizer towards complete sentences with the correct word order, but they should account for disfluencies in speech and errors in transcription.

3.5 Putting it all together

3.5.1 Using the VAD-alignment to restrict the language model

The timing information generated by a highest-scoring alignment of the segment voicedness likelihoods is used to restrict the language model when decoding the

audio. The duration of the audio is divided into time windows, the length of which is an input to the aligner. For each window, a text corpus is built from the text segments guessed to be in that window or in the neighboring windows. The number of neighboring windows used is also an input to the script. Each audio segment in the current time window is decoded with a language model built from that corpus. The transcriptions are saved, as well as an alignment file representing the HMM state matched to each frame in the audio.

3.5.2 Compiling results

Textual alignment

The hypothetical transcripts are aligned using sclite to the known transcripts. sclite outputs an sgml file containing pairs of the form (alignment type, word), where alignment type is one of insertion, substitution, deletion, or correct. Sequences of correct words are extracted from the alignment file are assumed to indicate a match between the audio segment and the transcript, and are saved as anchors. The minimum number of sequential words is passed as a parameter to the aligner.

Alignment scope

There are two modes of operation for alignment.

Global alignment concatenates the hypothetical transcripts for each time window, and aligns it to the full known transcript. This tends to eliminate false positive matches resulting from an overly biased language model, but it can reduce the likelihood of finding correct anchors if some time windows are very poorly decoded.

Local alignment operates on a smaller scale. After all the segments in a time window have been decoded, their hypothetical transcripts are concatenated and aligned to the corpus used to build the language model.

Converting textual alignments to subtitle times

Once an alignment has been found, the previously saved alignment file is used to determine at what time in the audio segment the aligned words are spoken. The aligned expression is also looked up in the text segment, and the syllables that occur before it are counted and multiplied by the speech rate. With knowledge of the expression's time offset in both the audio and the text segment, the match is used to anchor the subtitle at the appropriate point in time.

Chapter 4

Development practices

4.1 Programming language

I chose Python for the project for several reasons. Rapid prototyping and improvement was an important requirement and I wanted to avoid having to compile anything. Despite Python's dynamicity, I was able to catch a majority of errors before runtime simply by running a live linter in my text editor. Numpy provides sufficient performance for the more data intensive tasks such as VAD-based alignment, and for most of the other tasks performance is unimportant, as the bottleneck for the system is the speech recognition system, which is invoked as a separate binary.

Throughout the development process Python's succinctness was a huge boost to development speed, whether it be for writing code, refactoring code, or debugging code.

4.2 Logging and debugging

Because of the long running time of the program, debugging non-deterministic errors that occurred towards the end of an alignment run was challenging. Guarding as much code as possible and throwing detailed exceptions containing information about most of the relevant execution context in their output was extremely helpful for debugging. In addition, all the bash scripts begin with

```
echo \$@
set -x
```

which logs both the full command-line used to invocate the script and every command executed by the script. This let me run the aligner on several files

and then grep for errors in the logs. Despite inspecting crashes after the fact rather than through a live debugger, I was able to fix bugs easily by ensuring that every step used rigorously verbose logging.

4.3 Testing

End-to-end tests were very useful for the algorithmic parts such as the one aligning time-segments, especially since changing parts of the algorithm usually meant changing a large portion of code whilst requiring the same correct outputs. Most of the rest of the code was harder to automatically test for a broad range of input cases. I did however perform a few tests to ensure the quality of my acoustic models, including comparing them to decodes with HTK trained on the Wall Street Journal corpus (much higher WER) and running them on a test set retained from the Voxforge corpus (achieving approximately a three percent error rate with a trigram language model trained on the test set.)

4.4 Graphviz

Graphviz was very helpful to visualize the factor automatons I was generating and debug the code accordingly.

4.5 Version control

I used git for version control. I used a simple approach where each feature had its own branch so as to be able to gradually merge functionality into the main codeline. I also used temporary work branches as much as possible so that the last working iteration was only a checkout anyway at any point, and I could commit partial implementations without worrying about breaking a working feature branch.

Chapter 5

Experiments and evaluation

5.1 Test data

I run the long audio alignment tool on 6 clips from each of the movies "12 Angry Men" and "Coffee and Cigarettes", 3 clips from each of the first four episodes of Friends, and the first chapters of audiobook recordings of Alice in Wonderland, Adventures of Huckleberry Finn, and Price and Prejudice.

12 Angry Men is used as an example of mostly clean speech but with frequent interruptions and a large number of speakers. This increases the difficulty of segmentation, voice activity detection, and speech decoding.

Clips from Coffee and Cigarettes have one or two speakers, but with background brouhaha or music.

Clips from Friends contain a variety of speakers, but the intention is for utterances to be cleanly segmented since they are separated by the laugh track.

Finally, audiobooks provide the easiest target for alignment, with a single speaker and no background noise.

5.2 Experiments

For each transcription target, many passes are run experimenting with the following parameters:

- The threshold for voice activity is set to either -0.1, 0.0, or 0.1
- The minimum consecutive time above or under the threshold for a part of the audio to be categorized as voiced or unvoiced is set to either a quarter of a second or half a second.

- The minimum number of matching words between the decoded text and the transcript to save the subtitle as an anchor ranges from 4 to 7
- Text-to-text alignment is done either for each time window or on the complete transcript and decoded text
- The language model is either a trigram back-off model generated by SRILM or my own factor automaton implementation

5.3 Evaluation

I compare alignments and extract the following statistics:

- Worst subtitle offset
- Location of worst subtitle offset (mostly for debugging purposes)
- Sum of offsets
- Mean offset
- Number of subtitles with an offset bigger than 15, 10, and 5 seconds

I also look at the logs to inspect the performance and errors made by the speech recognition section. Anchors are counted and compared against the reference subtitles to test for false positives.

For audiobooks there are no reference subtitles, so the evaluation is entirely subjective and empirical. A crude numerical result is given by counting the proportion of poorly aligned subtitles.

5.4 Selected results

5.4.1 Movies and series

Anchor counts are 0 for almost every movie clip. One anchor is successfully found for two of the clips from "Coffee and Cigarettes" and in both cases it is found to be a true positive when compared with the reference subtitles. The alignments are therefore almost exclusively the result of the VAD-based alignment.

Video	Mean	Max	Notes
Friends E1C1	3.4	15	Only 2 above 10
Friends E1C2	5	30	Only 2 above 15, 6 above 10
Friends E1C3	2.6	9	
Friends E2C1	6	12	
Friends E2C2	3	12	
Friends E2C3	1	11	
Friends E3C1	1	12	Only 2 above 10
Friends E3C2	4	44	Only 2 above 15
Friends E3C3	2	7	
Friends E4C1	4	14	
Friends E4C2	10	29	16% above 15, 27% above 10
Friends E4C3	4	18	Only 6 above 15, 3 above 10
12 Angry C1	2	10	
12 Angry C2	2	7	
12 Angry C3	9	73	11 above 15
12 Angry C4	5	24	7 above 10
12 Angry C5	3	9	
12 Angry C6	4	15	
Coffee C1	1	14	Only 1 above 10
Coffee C2	2	9	
Coffee C3	14	71	Only 4 above 15
Coffee C4	3	79	Only 3 above 15
Coffee C5	1	10	
Coffee C6	2	14	

5.4.2 Parameters

25 centisecond segments with a threshold of -0.1 consistently yields the least error. In the few cases where other values work better, the difference is marginal, whereas oftentimes higher segment length or thresholds yield very poor alignments.

5.4.3 Audiobooks

10 minutes of the Alice in Wonderland audiobook are perfectly aligned with 95 completely correct subtitles and 6 offset by a small amount between 1 and 3 seconds.

7 minutes of Huckleberry Finn are perfectly aligned with 96 completely correct subtitles, and 0 mistakes.

The error rate is higher on 5 minutes of a recording of Pride and Prejudice: 38 subtitles are correctly aligned and 8 are offset by a small amount. The error rate grows higher when using a shorter minimum match length or a factor automaton. The erroneous subtitles are consecutive, and all have a similar incorrect offset (for instance, there is a sequence of 4 subtitles each of which appears during the audio segment preceding the correct one).

5.5 Analysis

5.5.1 Factor automaton

On the Price and Prejudice recording factor automatons worsened the alignment. The type of error described above happens when we find false positives anchors, because each false positive skews the alignment of all the subtitles in between itself and its neighbour anchors by a constant offset. This leads me to conclude that the factor automaton is overly biasing the recognizer and causing it to output hypotheses matching the transcription despite the it not being spoken in the audio.

5.5.2 Poor speech recognition performance

Despite using a state-of-the-art toolkit and acoustic training on 83120 VoxForge utterances, the speech recognizer's performance is inadequate to say the least on anything but perfectly clean speech. Neither the feature transforms and discriminative training used to alleviate the recognizer's training-set specificity nor the extremely constrained language model are sufficient to yield good decoding results. Obtaining high accuracy on movies is also complex because most on-line

adaptive training is inapplicable when there are several speakers and a range of different noise types.

To help ensure that the poor results weren't due to programmer error, I ran the speech recognizer on specific utterances using a language model built solely from the utterance being decoded, in other words telling the recognizer exactly what to expect. This barely increased successful recognition, which led me to conclude that the problem was noise and speaker variety.

5.5.3 VAD-only passes yield low maximum error

While the speech recognizer was completely unable to generate anchors for movies and series, the VAD-only passes showed very promising results in that the offsets were quite small on average, and usually bounded by a low maximum, around 5% of the total length of the clip being aligned. In addition, the few clips showing very high maximums around 70 seconds still only had very few segments above 10 or 15 seconds error. The approach is therefore sound, and a significant improvement over the old method of recursively refining the language model in between finding anchors, since the language model can be sufficiently constrained from the get-go based on the preliminary alignments.

5.6 Comparison with other tools

5.6.1 SAIL

I attempted to use the SAIL alignment tool to generate alignments to compare my results with, as the paper only reported results on clean speech and I was curious to see if its performance would suffer as greatly on audio from sources other than voice corpii. However, I was unable to get the tool to run correctly, even on the sample data included in the distribution. SAIL crashes consistently, first with a division by zero error which I patched, then with a host of other errors.

5.6.2 YouTube

YouTube's speech decoder has a much higher successful recognition rate. I assume this is due to an immense amount of training data with a multitude of speakers. An interesting thing to note is that Moreno's good results aligning text to audio with factor automata were obtained using Google's speech-to-text infrastructure. A more reliable speech decoder is indeed less likely to be biased towards false positives by an overly constrained language model. This explains why the use of factor automata resulted in worce alignments on Price and Prejudice.

5.6.3 Proof-of-concept results using YouTube for speech decoding

I ran the aligner on an episode of Friends using YouTube's speech recognizer output (that produced witout providing captions). The entire hypothetical transcription from the YouTube video was aligned as usual, and 10 anchors selected as random. Running the aligner using these anchors as though they had been obtained by the usual decoding process, the VAD-based alignment was able to produce a near-perfect alignment with no subtitles offset by more than 15 seconds and only 6 offset by more than 10, out of a total of 300. The mean error was around 2 seconds, and only 40 subtitles were more than 5 seconds off. Empirically, the video was mostly watchable, and most of the mis-aligned subtitles were very short exclamations. This goes to show that combined with slightly stronger speech decoding performance, the VAD-based alignment works extremely well to fill in the blanks left by erroneous decoding.

Chapter 6

Conclusion and future work

6.1 Achievements

I successfully developed an alignment tool with near-perfect results on clean speech such as that found in an audiobook recording. In addition, I demonstrated the significant benefit of a previously unused technique. Using the results of noise-robust voice activity detection, I was able to accurately constrain the language model as soon as the first pass, yielding much better results than the existing solution based on iterative refinement of the language model in between anchors. In addition, the alignment estimation in between anchors yields great accuracy.

6.2 Unrealized expectations

I started the project fully expecting to be able to generate near perfect accuracy alignments on feature-length movies. The poor performance of my speech decoding system was a rude awakening and disapointment. Having read more about the speech decoding state of the art, I now understand that domain specificity (for instance via speaker adaptation on some training data) is required to get reasonable performance from a speech decoding system. While constraining the language model and applying feature-space transforms does yield better results, it is far from sufficient to successfully decode speech from a multitude of different speakers and intonations not following any predictable pattern, especially if that speech is overlayed with lots of noise.

6.3 Future work

Improvements to the alignment system should clearly focus on the speech decoding part of the process, as it is the weakest link. Besides having access to reliable acoustic models such as those at Google's disposal, an option might be to make the alignment process partially supervised to improve the speech recognizer's adaptation to the speaker set. The user would identify the speaker in a few segments and new models would be adapted to each speaker. For the remaining segments, since time and performance usually aren't a constraint for this kind of offline work, the decoder could run several times: once with each speaker adapted model, and the hypothesis closest to the transcript selected.

Chapter 7

Appendix

7.1 External programs and libraries

I use the following external programs and libraries:

- Kaldi[1] is used as the speech recognition system.
- sclite[2] is used to align hypothetical transcripts.
- srilm[3] is used to generate trigram grammars.
- Sequitur G2P[4] is used to generate pronunciations for words not found in the CMU pronunciation dictionary
- MPlayer[5] is used to extract audio as WAV from movies
- ffmpeg [6] is used to split movies into short clips
- SoundeXchange[7] ("the swiss army knife of speech processing") is used to convert audio to the format expected by the VAD (16-bit signed integer encoding) and to split multi-channel audio into its individual components
- I was kindly given a binary distribution of the LSTM-RNN VAD developed by Florian Eyben, Felix Weninger, Stefano Squartini and Bjorn Schuller.
- NumPy [8] arrays and operations are used to work on large sets of data such as the voicedness likelihoods

[1] http://kaldi.sourceforge.net/
[2] http://www1.icsi.berkeley.edu/Speech/docs/sctk-1.2/sclite.htm
[3] http://www.speech.sri.com/projects/srilm/
[4] http://www-i6.informatik.rwth-aachen.de/web/Software/g2p.html
[5] http://www.mplayerhq.hu/design7/news.html
[6] http://www.ffmpeg.org/
[7] http://sox.sourceforge.net/
[8] http://docs.scipy.org/doc/

- pysrt [9] reads and writes subtitle files in the SubRip format.

Each of these must be installed and available in the system path (except for Kaldi, see below) at the time of running the program. NumPy and pysrt can be installed using

```
pip install numpy
pip install pysrt
```

7.2 Distribution contents

The distribution includes the following data files:

- Triphone acoustic models trained on the Voxforge corpus
- The CMU pronunciation dictionary
- The Moby hyphenation dictionary
- A list of phones from training - these are necessary because the decision tree references hard-coded phone integer IDs
- A Sequitur G2P model for pronunciation generation

7.3 Running the aligner

7.3.1 Kaldi

There are no binary distributions of Kaldi available at the moment and it must be compiled from source. Once Kaldi is installed, you can either add it to your system path or modify the KALDI_ROOT variable in the script path.sh to point to the root of your Kaldi installation.

7.3.2 Preparing the data

Use clip.py if you wish to extract a short video segment and the relevant subtitles.

Use book_to_subs.py to convert an audiobook in text format to an unaligned srt file (all the programs expect srt as input).

Use prepare_data.py to dump audio, perform voice activity detection, and segment into decodable chunks selected from the voiciest stereo channel.

[9] https://pypi.python.org/pypi/pysrt

7.3.3 Running the aligner

Use main.py to run the aligner.

All these programs are self documenting, run them with the $-h$ or $--help$ flag for more details.

Bibliography

[1] A. Benyassine, E. Shlomot, H. Y. Su, D. Massaloux, C. Lamblin, and J. P. Petit. Itu-t recommendation g.729 annex b: A silence compression scheme for use with g.729 optimized for v.70 digital simultaneous voice and data applications. *Comm. Mag.*, 35(9):64–73, September 1997.

[2] Florian Eyben, Felix Weninger, Stefano Squartini, and Björn Schuller. Real-life Voice Activity Detection with LSTM Recurrent Neural Networks and an Application to Hollywood Movies, May 2013.

[3] Mark Gales and Steve Young. The application of hidden markov models in speech recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304, 2008.

[4] Mark JF Gales. Maximum likelihood linear transformations for hmm-based speech recognition. *Computer speech & language*, 12(2):75–98, 1998.

[5] Timothy J. Hazen. Automatic alignment and error correction of human generated transcripts for long speech recordings. In *In Proc. Interspeech*, 2006.

[6] Athanasios Katsamanis, MP Black, Panayiotis G Georgiou, Louis Goldstein, and S Narayanan. Sailalign: Robust long speech-text alignment. In *Proc. of Workshop on New Tools and Methods for Very-Large Scale Phonetics Research*, 2011.

[7] Slava Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *Acoustics, Speech and Signal Processing, IEEE Transactions on*, 35(3):400–401, 1987.

[8] Yang Liu, Elizabeth Shriberg, and Andreas Stolcke. Automatic disfluency identification in conversational speech using multiple knowledge sources. In *In Proc. Eurospeech*, pages 957–960, 2003.

[9] Mehryar Mohri, Fernando Pereira, and Michael Riley. Speech recognition with weighted finite-state transducers. In *Springer Handbook of Speech Processing*, pages 559–584. Springer, 2008.

[10] Fabrizio Morbini, Kartik Audhkhasi, Kenji Sagae, Ron Artstein, Dogan Can, Panayiotis Georgiou, Shri Narayanan, Anton Leuski, and David Traum. Which asr should i choose for my dialogue system?

[11] A. Moreno, L. Borge, D. Christoph, R. Gael, C. Khalid, E. Stephan, and A. Jeffrey. Speechdat-car: a large speech database for automotive environments. In *Proceedings 2nd International Conference on Language Resources and Evaluation, LREC 2000*, 2000.

[12] Pedro J Moreno and Christopher Alberti. A factor automaton approach for the forced alignment of long speech recordings. In *Acoustics, Speech and Signal Processing, 2009. ICASSP 2009. IEEE International Conference on*, pages 4869–4872. IEEE, 2009.

[13] Pedro J. Moreno, Christopher F. Joerg, Jean-Manuel Van Thong, and Oren Glickman. A recursive algorithm for the forced alignment of very long audio segments. In *ICSLP*. ISCA, 1998.

[14] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, et al. The kaldi speech recognition toolkit. In *Proc. ASRU*, pages 1–4, 2011.

[15] Javier Ramrez, Jos C Segura, Carmen Bentez, ngel de la Torre, and Antonio Rubio. Efficient voice activity detection algorithms using long-term speech information. *Speech Communication*, 42(34):271 – 287, 2004.

[16] Jongseo Sohn and Wonyong Sung. A voice activity detector employing soft decision based noise spectrum adaptation. In *Acoustics, Speech and Signal Processing, 1998. Proceedings of the 1998 IEEE International Conference on*, volume 1, pages 365–368 vol.1, May 1998.

[17] Andreas Stolcke et al. Srilm-an extensible language modeling toolkit.

[18] Grady Ward. Moby hyphenator ii, May 2002.