Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Linguistic, Speech and Communication Sciences,
Trinity College, Dublin

Extracting ASR Training Data from Conversational Speech.

Oisín Nolan

April 30, 2021

A Final Year Project submitted in partial fulfilment
of the requirements for the degree of
Computer Science and Language

Declaration

I hereby declare that this Final Year Project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Acknowledgements

A huge thank you to my supervisor, Prof. Ailbhe Ní Chasaide, and all the members of the ABAIR recognition group for their support, guidance, and good humour throughout the project. In particular, thanks to Liam Lonergan for many helpful discussions on speech technology, and for running the final experiments evaluating my work's impact on the ABAIR ASR system.

And thanks to my friends, family, and Sarah whose support (ability to put up with me!) undoubtedly helped get me through this difficult year.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Aims

The overarching goal of this project was to contribute to the development of automatic speech recognition (ASR) technology for the Irish language. Success in ASR systems is dependant on a number of factors: acoustic modelling, language modelling, and a large amount of high-quality training data. It was through the production of training data that this project aimed to make its contribution. The project looked to *Corpus na Cainte Beo*, an existing corpus containing conversational speech, as a source of training data. A number of challenging features of spontaneous conversational speech needed to be addressed in order to extract useful training data from this corpus. The project aimed to produce a system for automatically extracting training data that was of a quality high enough to improve the performance of an existing ASR system developed as part of the ABAIR initiative [1]. Methods of automatically extracting training data used in the past often rely on existing ASR systems in that language. The ASR system being developed at ABAIR was the first of its kind, however, and at the beginning of the project did not perform well enough on conversational speech to be used for this purpose. Thus, the aim was to develop a system that depended only on pre-trained, language-independent speech technologies. The pre-trained, language-independent nature of the system would enable its application to any corpus of the same format as *Corpus na Cainte Beo*, in any language.

1.2 Overview

Given an audio clip containing speech and an associated transcription of that speech, the proposed system, referred to as the *filtering system*, extracts pairs of corresponding text and speech segments that can be used as training data. The input audio clip may be, for example, a forty minute interview containing multiple speakers, music, singing, and laughing. The extracted training data should consist of short speech clips, each containing speech from a single speaker at a time, along with a fragment of the transcription text that corresponds to that speech. The transcription files in the corpus do not provide information indicating when each fragment of text is spoken in the corresponding audio. Thus, the first step is to estimate an alignment between the speech and its transcription. An alignment estimation technique called *forced-alignment* is used to this end, producing pairs of text and speech segments that are estimated to correspond with one another. *Speaker diarization* and *overlapped speech detection* systems are then used to provide some confidence that each pair of text and speech data

is well-aligned and produced by a single speaker. The system was validated at two levels of detail: firstly, each of the component systems (forced-alignment, speaker diarization, overlapped speech detection) were evaluated by comparing their output to manually created 'ground-truth' outputs, and secondly, the quality of the training data produced by the system as a whole was evaluated by examining its impact on the existing ASR system. It was found that adding the training data produced by the filtering system to the baseline ASR system resulted in a relative performance improvement of 4.5% on read speech, and 11.5% on conversational speech. Thus, the filtering system contributed to a higher-quality, more robust ASR system for Irish.

1.3 Motivation

Speech recognition technology is becoming more and more integrated into daily life thanks to new innovations in machine learning. A report[1] from Google shows that 27% of the global online population uses voice search. Devices such as Amazon Alexa and Google home bring speech technology into the home – seamlessly integrating it with everyday life. Speech technology is also proving useful in a variety of business applications[2], and finding important applications in areas such as healthcare [2], accessibility, and education [3]. For these technologies to perform well, they require large amounts of labelled training data, consisting of transcribed speech corpora. The manual creation of such corpora requires time, transcribers and money. A *low-resource* language is one for which few labelled corpora exist. It is important to develop speech technologies for low-resource languages, both to enable members of the language community to engage with the digital world through their language, as well as providing an opportunity to document, and possibly revitalise the language [3]. The availability of ASR model architectures is not an issue for most low-resource languages – collections of speech recognition models are available open-source and for free online, through sites such as GitHub and Hugging Face[3]. Rather, the bottleneck appears in the availability (or lack thereof) of training data. Some efforts have been made to generate speech corpora using crowd-sourcing, for example Mozilla Common Voice [4]. The system proposed in this project aims to draw usable training data from existing sources of annotated speech, such as transcribed radio interviews, captioned YouTube videos, or subtitled movies. While the evaluation in this paper was conducted on transcribed radio interviews, similar experiments could be conducted to validate its use on other, similar, sources. A benefit that this system provides is that it is automatic, meaning that it could be run on thousands of hours of audio, making potentially large training datasets without any need for typical resources such as paid transcribers. Relevant to the case of low-resource languages is the fact that the system does not depend on any existing speech technology for the language of the audio sources it processes.

This project was carried out as part of the ABAIR initiative [1], whose mission is to bring Irish into the digital space. ABAIR approaches this goal on multiple levels: linguistic resources, such as corpora, letter-to-sound rules, and normalisation rules,

[1] `https://www.thinkwithgoogle.com/marketing-strategies/search/voice-search-mobile-use-statistics/`

[2] `https://summalinguae.com/language-technology/new-technology-in-speech-recognition/`

[3] `https://huggingface.co/models?pipeline_tag=automatic-speech-recognition`

provide a foundation on which ABAIR's core language technologies (such as speech synthesis and speech recognition) are built. These core technologies are then applied in a number of areas, such as education [5] and accessibility [6]. The Irish language is classified as 'definitely endangered' by UNESCO [7]. In this light, developing speech technology for Irish aims to contribute to its revitalisation.

1.4 Structure

This paper is divided into a number of chapters. The **background** chapter provides some brief descriptions of technologies and toolkits referred to throughout the paper. It also describes similar work in the area, focusing on automatic corpus construction, and automatic methods of aligning speech and transcription text. The **methodology** chapter describes each of the speech tasks relevant to the filtering algorithm, providing some background information on each task before outlining the specific model architecture chosen. The **evaluation metrics** chapter defines some measurements used to evaluate the performance of the models discussed in the methodology chapter. **Experiments and results** describes a number of experiments that were conducted to validate the system components as well as the system as a whole. Some discussion and interpretation of the results is provided in each case. Finally, the **conclusion** summarises the project, and outlines future directions for work in this area.

2 Background

2.1 Automatic Speech Recognition

Automatic Speech Recognition (ASR) is defined by Jurafsky and Martin [8] as the task of mapping acoustic waveforms to sequences of graphemes. In general, ASR involves extracting a sequence of feature vectors $\mathbf{X}$ from the input waveform, each feature representing a short fixed-length window of audio, and mapping them to a much shorter output sequence Y of labels. The task can be viewed as finding the most likely output sequence Y given the input sequence X and output sequence so far [8] (eq. 1). A language model is also typically added to the system in order to enhance the naturalness of the output sequence.

$$p(y_1, ..., y_n) = \prod_{i=1}^{n} p(y_i | y_1, ..., y_i, \mathbf{X}) \tag{1}$$

Automatic transcription of speech is one of the earliest goals in language processing [8], and as such, many approaches to this task have been taken over the years. As is typically the case with speech tasks, HMM-GMM architectures were used throughout the 1990s, eventually being outperformed and replaced by neural architectures. Nowadays, common paradigms for speech recognition include encoder-decoder models, typically using either LSTM- or Transformer-based architectures, and Connectionist Temporal Classification (see Jurafsky and Martin [8] for detail).

2.1.1 ABAIR Model

The Irish ASR system being developed by the ABAIR group uses a Time-Delay Neural Network (TDNN) [9] architecture. TDNNs are feed-forward neural networks that have added *delays* which enable them to represent relationships between different events in time. The TDNN architecture proposed by Peddinti et al. [10] was used, sub-sampling frames from the input sequence in order to reduce training time. Parallelization during training further reduces training time. Features included MFCCs[1] and i-vectors [11]. The Kaldi [12] speech recognition toolkit was used to create this model. An 3-gram language model trained on a number of speech and text corpora was used to improve decoding naturalness. The model was trained on 20.2 hours of crowd-sourced recordings from the ABAIR Míleglór website[2], 19.4 hours of record-

[1] `https://en.wikipedia.org/wiki/Mel-frequency_cepstrum`

[2] `https://www.abair.tcd.ie/studio/ga/recorder/`

ings of native speakers, and 5.1 hours of spontaneous conversational speech. Further details about the ABAIR ASR system are provided by Qian et al. [13].

2.2 Corpus na Cainte Beo

Corpus na Cainte Beo is a speech corpus consisting of approximately four hundred hours of interview clips from Raidió na Gaeltachta. These four hundred hours are spread across one and a half thousand `.wav` files, each containing an interview typically ranging from a duration of twenty to forty minutes. Each `.wav` file is paired with a transcription document in the format shown in table 2.1, where *cainteoir* translates to *speaker* and *caint* to *speech*. The *cainteoir* column contains speaker ids, and the *caint* column contains the transcription of the speech spoken by that speaker. These transcription rows trace the interviews' speech speaker-by-speaker in chronological order.

Cainteoir	Caint
Tras_C0003_CC01	Go raibh maith agat, a Sheáin.
Tras_C0003_C50	Théis na Nollag, he he he...
Tras_C0003_CC01	Táimid beo fós, bail ó Dhia orainn.
Tras_C0003_C50	Daoine, daoine beo, daoine beadaíocht leob ar an Nollaig...
Tras_C0003_CC01	[Yeah], Agus an [Cholesterol] imith' suas aríst.

Table 2.1: Corpus na Cainte Beo transcription extract.

The audio clips contain spontaneous, multi-party speech which tends to pose a number of issues in automatic speech processing tasks [14]. One important issue is that of overlapped speech, which is not marked in the transcription text. This means that overlapped speech inevitably results in some mismatch between the audio and transcription text, which can cause significant degradation in ASR performance [15]. Overlapped speech detection is thus a core topic of this project.

2.2.1 Samples used for Evaluation

As described above, the annotations provided in Corpus na Cainte Beo contain only a sequence of text fragments speaker-by-speaker. The corpus does not contain the timing information that would be necessary to evaluate models for the speech tasks applied during this project, such as forced-alignment, overlapped speech detection, and speaker diarization. Two sample files from the corpus were manually annotated using Praat [16] for each task in order to create 'ground-truth' references that could be used for comparison and evaluation. The samples chosen have the ids `C0051` and `C0057`, and are referred to as such in following sections. These samples were chosen because they represent the variety present in the dataset. `C0051` contains seven different speakers, some male, some female, as well as telephone speech and singing. `C0057`, on the other hand, contains just two male speakers, speaking back and forth for the duration of the interview. `C0051` is 2165 seconds long, and `C0057` is 1647 seconds long. While ideally more clips would have been manually labelled, due to the scope

of this project and time-consuming nature of manual annotation, it was decided that choosing two representative clips would be sufficient.

2.3 Forced Alignment

Forced alignment is a method of automatically aligning speech audio and corresponding textual transcriptions. This subsection describes `aeneas`[3], an open-source forced aligner that was used to estimate an alignment for *Corpus na Cainte Beo*. Most forced aligners use speech recognition techniques to estimate alignment, for example, the Montreal Forced Aligner [17]. This typically involves using an ASR system to generate text from the speech provided, which is then mapped to the most similar text in the ground-truth transcription text according to some measure of textual similarity. The speech audio is thus transitively mapped to the transcription text. `Aeneas` approaches this problem from the opposite perspective, opting to first synthesise the transcription text and then match the synthesised audio to the original speech audio by a measure of audio similarity. This is process is carried out in a number of steps, as described on the `aeneas` wiki[4]:

- **Text-to-speech synthesis:** the transcription file is provided as a sequence of text fragments $F = [f_1, f_2, ..., f_q]$. Each of these fragments is synthesised, producing a single synthesised audio file S, along with a mapping M_1 from fragments in F to time intervals in S.
- **Computation of MFCC features:** next, `aeneas` computes Mel-frequency cepstral coefficients (MFCCs) for both the real audio, R, and the synthesised audio, S. MFCCs provide a representation of audio that highlights features relevant to speech processing, and are commonly used in speech-related tasks, such as ASR [18]. The MFCC features form matrix representations of S and R, with each row containing the coefficients for a fixed-length frame of audio.
- **Cost matrix:** a cost matrix is calculated as the dot product of the two MFCC matrices. Each entry $[i, j]$ in the cost matrix is a measure of similarity between the MFCC vectors representing the i^{th} and j^{th} frames of S and R respectively; the lower the value at $[i, j]$ the more similar those frames sound.
- **Dynamic time warping:** the dynamic time warping algorithm is then used to calculate a minimum-cost path through the cost matrix. This path constitutes a mapping, M_2, from frames in S to frames in R. Recall that each frame corresponds with an interval of time in each of S and R.
- **Compose mappings:** the mappings M_1 and M_2 can now be composed to form a final mapping M from the text fragments in F to time intervals in R.

The final output thus consists of text fragments and associated time intervals. These are output in a `JSON` file in the format of the example given in appendix A1.1.

[3] `https://github.com/readbeyond/aeneas`

[4] `https://github.com/readbeyond/aeneas/blob/master/wiki/HOWITWORKS.md`

2.4 Pyannote Toolkit

`Pyannote`[5] is a suite of open-source python libraries for intelligent multimedia processing tasks. Many `pyannote` tools were used throughout this project, for example `pyannote.audio`, which provides machine learning models for tasks such as *overlapped speech detection* and *speaker diarization*, and `pyannote.metrics`, which provides tools for evaluating speaker diarization systems. These are discussed in detail in chapters 3 and 4. This section describes the `pyannote.core` library and the `PyanNet` neural network architecture, both of which served as important tools in many areas of this project and, correspondingly, are referenced throughout the chapters to come.

2.4.1 Pyannote Core

`Pyannote.core` [19] is a python library providing advanced data structures for handling temporal segments. Such data structures include *segments*, *timelines*, and *annotations*, each of which are used extensively throughout this project. The following is a description of each of these data structures.

- **Segment:** a segment is essentially an interval of time, defined by its start and end times. A variety of operations can be applied to a segment. The unary operator `duration`, for example, returns the duration of a given segment. The intersection operator can be used to calculate the segment that is the overlapping portion of two segments, e.g. $[1, 4] \cap [2, 5] = [2, 4]$.
- **Timeline:** a timeline is an ordered set of segments. Many useful operations are provided for manipulating timelines. For example, the `duration` operation returns the sum of the durations of each segment in the timeline. The `crop` operation finds any segments in the timeline that are within a given interval, specified by a parameter in the form of a segment. `Pyannote.core` also provides visualisation for timelines, which is invaluable in manually inspecting results on-the-fly. See, for example, figure 2.1.

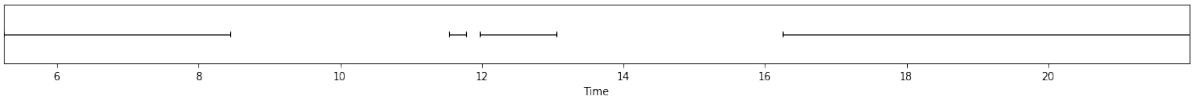

Figure 2.1: Timeline visualisation.

- **Annotation:** an annotation is a labelled timeline. In other words, it is a timeline and a mapping from the segments in that timeline to string labels. Annotation segments with different labels can occupy the same time interval, in which case they are thought of as existing on parallel *tracks*, as opposed to constituting the same segment, as is the case in timelines. An annotation can be converted into a timeline of its segments, using the `get_timeline` operation. Visualisation is also provided for timelines, colour-coding each segment according to its label. See figure 2.2 for an example.

[5] `http://pyannote.github.io/`

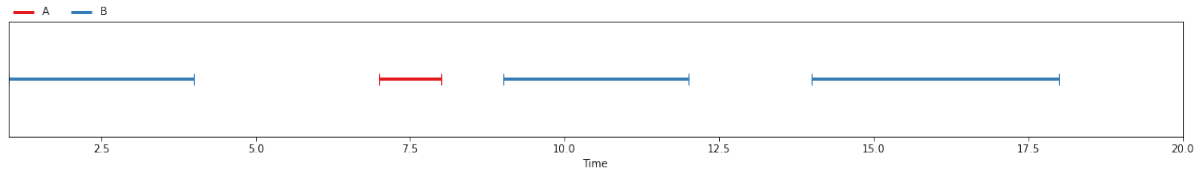

Figure 2.2: Annotation visualisation.

While visualisation is a key feature of `pyannote.core`, it does not provide any interface for listening to audio or interactively creating annotations. Thus, a number of python scripts were written to translate between Praat [16] TextGrid[6] files and `pyannote.core` data structures. Using these scripts, annotations could be created manually using Praat, and then evaluated in a python environment. Similarly, annotations produced automatically by machine learning algorithms could be converted into Praat TextGrids, which could be manually inspected and evaluated in Praat.

2.4.2 PyanNet Architecture

A sequence labelling task is defined by Bredin et al. [20] as one in which the input is a sequence of feature vectors $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_T\}$ and the output is a sequence of labels $\mathbf{y} = \{y_1, y_2, ..., y_T\}$ where each $\mathbf{x}_t$ is labelled y_t. Each label y_t will be one of a K of classes, i.e. $y_t \in [1, K]$. Each class will have some domain-specific interpretation. For example, if the sequence labelling task is voice activity detection, then we could let $K = 2$: $y_t = 1$ if there is no voice activity, and $y_t = 2$ if there is. A labelled dataset of this form can be used to train a neural network to learn a function $f : \mathbf{X} \rightarrow \mathbf{y}$ which can then be used to predict a sequence of labels given an unseen sequence of feature vectors. As discussed in 3.2, `pyannote.audio` [20] formulates the task of speaker diarization as a number of sequence labelling sub-tasks, including voice activity detection, speaker change detection, and overlapped speech detection. The generic `PyanNet` architecture is provided to approach such audio sequence labelling problems, and it can be applied to each of the sub-tasks mentioned. More specifically, the pre-trained `pyannote.audio` models used throughout this project are all based on the `PyanNet` architecture.

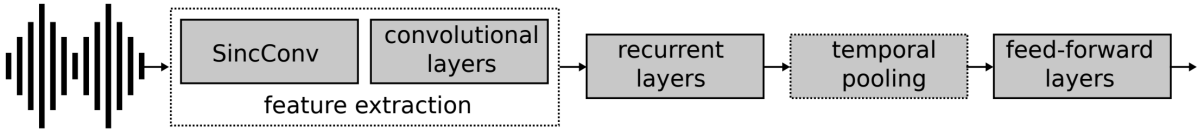

Figure 2.3: `PyanNet` sequence-labelling architecture.

As depicted in figure 2.3, `PyanNet` consists of a number of neural network modules that transform an input audio waveform into some output label, which varies depending on the task. Each module transforms the data according to a number of parameters, or weights, whose values are determined during training. Training involves giving the model some audio input, and then comparing its output label to a *correct* label for that input given by a training dataset. The model parameters are iteratively improved by calculating the degree to which each parameter contributes to the error in the model's

[6] `https://www.fon.hum.uva.nl/praat/manual/TextGrid.html`

predictions, using the backpropagation algorithm, and updating it to reduce that error. Repeating this process should find a parameter configuration that minimises the model error with respect to the training data. Thus, this model can be trained to solve a number of speech tasks, given a training dataset with sample input / output pairs for that task. The following provides some brief descriptions of `PyanNet`'s modules, specifying in each case the parameters to be tuned during training.

- **SincNet** [21] is a convolutional neural network architecture used in `PyanNet` for feature extraction. Feature extraction is the process of transforming input data into a new form, representing it in a way that is useful for the task at hand. SincNet performs a convolution with the input chunk of speech signal $x[n]$ and a bandpass filter function g, that allows through components of the signal within a specified frequency band.

$$y[n] = x[n] * g[n, f_1, f_2] \tag{2}$$

  In the frequency domain, g is the difference between two rectangular functions that serve as lowpass filters, and in the time domain these rectangular functions become $sinc$[7] functions.

$$G[f, f_1, f_2] = rect(\frac{f}{2f_2}) - rect(\frac{f}{2f_1}) \tag{3}$$

$$g[n, f_1, f_2] = 2f_2 sinc(2\pi f_2 n) - 2f_1 sinc(2\pi f_1 n) \tag{4}$$

  The parameters f_1 and f_2 defining these bandpass filters are learned from training data in such a way that they minimise the overall model error. This enables the model to learn how the input signal should be filtered to best solve the task at hand. Ravanelli and Bengio [21] showed that SincNet extracted signal features relevant to speech, such as pitch and formant frequencies, on a speaker verification task. See Ravanelli and Bengio [21] for more detail on SincNet's architecture.

- **Recurrent layers** are neural network layers that contain cycles within their network connections [22]. They are designed to process sequences of data, and are thus suitable for processing speech data. The input data for the recurrent layers in `PyanNet` is a sequence of features extracted by `SincNet`. The recurrent layers process this sequence X one-by-one, taking into account for each chunk of input x_t some information from the previous chunks, enabling the model to learn temporal relationships in the data. In the case of `PyanNet`, a type of recurrent layer called long short-term memory [23] (LSTM) is used. LSTMs learn temporal relationships by performing two tasks for each new input chunk they process: (i) removing contextual information that is no longer needed, and (ii) adding new contextual information that may be relevant in future. The LSTM model parameters determine the contextual information that should be retained or discarded at each chunk in the sequence. The LSTM layers in `PyanNet` should output an encoding of the input sequence of speech features that takes into account temporal relationships in the sequence. A visual guide to LSTM models is given by Olah [24], and an in-depth description by Jurafsky and Martin [22].

[7] `https://en.wikipedia.org/wiki/Sinc_filter`

- **Feed-forward layers** constitute the final module in `PyanNet`. These are standard, fully-connected layers, in which the output of each node is the weighted sum of its inputs (outputs from each node in the previous layer) fed through some non-linear activation function (in this case, the ReLU[8] function). The parameters for this module are weights that modify the strengths of connections between nodes. This module should learn a function that maps the output of the recurrent layers to the final output of the model, which will be a label that is relevant to the task at hand. See Jurafsky and Martin [25] for more detail on feed-forward layers.

Temporal pooling layers are also included for the task of generating speaker embeddings, which are vector representations of audio segments. In general, pooling layers serve to change the dimensionality of the input while retaining relevant information.

2.5 Related Work

A variety of automatic ASR dataset construction techniques have been used in the past. For example, the LibriSpeech corpus [26] was created from a set of thousands of audio books. Audio book data contains the necessary ingredients for ASR training data; namely, speech audio and corresponding textual transcriptions in a machine-readable format. Panayotov et al. [26] align the audio book speech and text by using an existing ASR system trained on another dataset to convert the speech to text, which can then be compared to the transcription text to find an optimal alignment. Other works construct corpora using spontaneous speech. For example, Ahmed et al. [27] construct a Bangla speech corpus using a variety of publicly available audio, including TV news recordings. In this case, the audio did not come with transcriptions, and so, rather than using alignment methods, two different ASR systems were used to generate transcriptions. If the two ASR systems output text that was above a specified degree of similarity, then that text was considered to be of sufficient quality to serve as a transcription. Mansikkaniemi et al. [28] construct a speech corpus from recordings of Finnish parliamentary meetings. They use an existing ASR system and the Levenshtein distance algorithm [29] to align segments of text and speech. ASR systems trained on this corpus showed improved WER for in-domain data, particularly when paired with an in-domain language model. A similar corpus was constructed by Helgadóttir et al. [30] from Icelandic parliamentary speeches. In this case, a first-pass alignment was created using as existing ASR system, and the resulting aligned segments were used to train a new, domain-specific recogniser, which was used for the final alignment. Lakomkin et al. [31] automatically construct a corpus by scraping YouTube videos with English closed captions. Once downloaded, the captions and corresponding audio clips are filtered by a number of heuristics identifying high-quality samples. In other words, as opposed to focusing on finding an optimal alignment of speech and text, samples with poor-quality alignment are simply discarded. This method is enabled by the endless stream of data available on YouTube.

It is clear that the use of pre-existing ASR systems is common in automatic speech corpus construction, in particular to align audio with text, and also to derive measures of alignment confidence. This was not a viable option in the case of this project,

[8] `https://en.wikipedia.org/wiki/Rectifier_(neural_networks)`

as the inital ABAIR ASR system showed poor performance on speech from Corpus na Cainte Beo. This is likely due to the fact that it was trained on read speech by non-native speakers, as opposed to spontaneous speech by native speakers. However, using training data extracted via the methods presented in this project, along with a domain-specific language model, it is possible that a semi-supervised approach could be taken, in which an ASR system is used to extract new or higher-quality data from the corpus, which is in turn used to improve the ASR system.

In addition to alignment of text and speech, mitigation of overlapped speech was also important in this project. Multiple approaches to this problem are possible. For example, Zorilă et al. [32] used data enhancement on the acoustically challenging CHiME-5 dataset [33]. As is the case with Corpus na Cainte Beo, CHiME-5 contains a large amount of speaker overlap; 23% of the audio in the dataset contains more than one active speaker [34]. Zorilă et al. [32] used *Guided Source Separation* [35], which involves separating speech from different speakers into their own tracks, in order to enhance the dataset. They found that this enhancement procedure improved recognition performance when also applied to the test set. It was thought that similar methods may be applied to *Corpus na Cainte Beo* in order to overcome the overlapped speech problem. `SpeechBrain`'s[9] transformer-based *SepFormer* model [36] was considered, but experiments were limited by computational capacity. Instead, methods for overlapped speech detection were applied, choosing to discard segments of audio containing large proportions of overlapped speech. This is discussed further in chapter 3.1.

[9] `https://speechbrain.github.io`

3 Methodology

3.1 Overlapped Speech Detection

3.1.1 Background

Overlapped speech may be defined as any region of speech in which more than one speaker is actively speaking at the same time. This is a natural phenomenon in spontaneous multi-party speech [37]. In the past, overlapped speech has been studied in relation to ASR performance, where it's presence has been shown to degrade performance, causing significant increases in word error rate [15, 38]. Shriberg et al. [38] suggest that overlapped speech should be incorporated into acoustic models. Çetin and Shriberg [15] observe a correlation between language model perplexity and the occurrence of overlapped speech. More recently, overlapped speech has been recognised as a key contributor to error in speaker diarization systems [39]. This has provided motivation for the development of overlapped speech detection systems. Boakye et al. [39] develop a hidden Markov model based architecture to distinguish between those segments of speech that contain overlapped speech and those that do not. Features used include MFCCs, root-mean-square energy, and LPC coefficients[1]. Shokouhi and Hansen [40] propose a novel time-frequency representation of audio called the *Pyknogram* to model sound in a way that is well-suited to distinguishing between the physical characteristics of overlapped versus non-overlapped speech. This enables an unsupervised approach to overlapped speech detection, in which spikes in the Euclidean distance between neighbouring frames' Pyknograms can indicate segments of overlapped speech.

3.1.2 Pyannote Model

A state-of-the-art overlapped speech detection model is proposed by Bullock et al. [41]. It is a neural network model using the `PyanNet` architecture described in section 2.4.2. Accordingly, overlapped speech detection is conceptualised as a sequence-labelling task, in which $K = 2$, i.e. there are two possible classes for a label to take on: either the label $y_t = 0$, indicating that the corresponding segment of audio $\mathbf{x}_t$ contains either zero speakers or one speaker, or $y_t = 1$, indicating that $\mathbf{x}_t$ contains two or more speakers. The model doesn't directly output the labels, however. Instead, it outputs a probability score that a given segment contains overlapped speech. This time series of overlap probabilities can then by converted to a sequence of overlap segments through a process called *binarization*. Binarization involves flattening the

[1] `https://en.wikipedia.org/wiki/Linear_predictive_coding`

continuous array of overlap detection scores into an array of ones and zeros, signifying overlap and non-overlap. This is implemented by the `binarize` function, based on a smoothing technique for voice activity detection output by Gelly and Gauvain [42], which uses a number of parameters to determine how the overlap scores should be flattened. Among these parameters are *onset* and *offset*, which together determine the threshold that overlap score must meet in order for it to qualify speech as overlapped. Other tunable parameters are *min-duration-on* and *min-duration-off*, which determine the smallest lengths of time that can be occupied by overlapped speech segments (on) and non-overlapped speech segments (off). Of most relevance to this project was the overlap threshold, i.e. the onset and offset parameters, as their values have an impact on (i) the amount of overlap that we can successfully eliminate, and (ii) the amount of non-overlap that gets mistakenly labelled as overlap, and thus needlessly discarded. A visualisation of this threshold and its impact in determining overlapped speech segments is provided in figure 3.1, in which the above plot shows the overlapped speech detection scores, with the overlap threshold in red, and the plot below shows the final overlapped speech segments. Tweaking the onset and offset parameters can be visualised as shifting this red threshold line up and down vertically, resulting in the output of different overlapped speech segments.

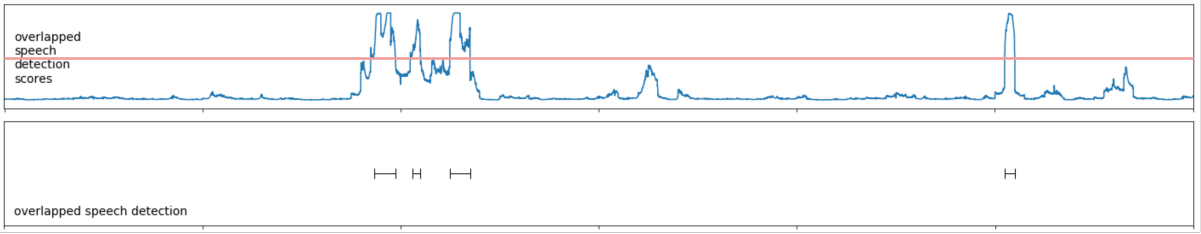

Figure 3.1: Overlap detection score binarization.

Pyannote makes this model available in two pre-trained forms: one trained on the AMI meeting corpus [43], and the other trained on data from the first DIHARD challenge [44]. When given an input `.wav` file, the model outputs a pyannote `Timeline`, in which the contained segments indicate those regions of the input audio clip containing overlapped speech.

3.2 Speaker Diarization

Speaker diarization is often defined as the task of labelling 'who spoke when' in a piece of audio. As Park et al. [45] points out, the word 'diarize' means to make note of some event, as in a diary. Park et al. [45] provides a recent summary of development in the field of speaker diarization, which is used to provide some background information here, detailing various techniques that have been used in the past to accomplish this task.

3.2.1 Background

In general, speaker diarization is conceptualised as a series of sub-tasks, as displayed in figure 3.2. Each of these modules was historically developed in isolation, although recent deep learning methods have enabled joint optimisation of sub-modules, which

is discussed in section 3.2.2. The following is a brief description of the problems proposed by each module, and mentions some methods that have been used to solve them.

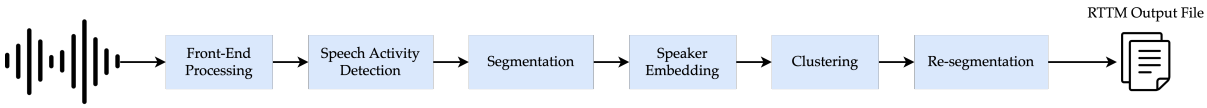

Figure 3.2: Speaker diarization sub-tasks.

- **Front-end processing** involves applying various transformations to the input audio signal to create an enhanced representation of speech, ultimately improving diarization performance. Methods used include dereverberation and speech separation.
- **Speech activity detection** aims to predict those regions of audio that contain speech, distinguishing them from those that do not. SAD systems have typically consisted of feature extraction and classification. MFCCs are typically used as features, with Guassian mixture models [46], hidden Markov models [47], and deep neural networks being used as classifiers.
- **Segmentation** partitions the audio into a sequence of segments. This may be done in a uniform way, choosing some fixed length for each segment, or it may be done at speaker-change points, as estimated by a speaker change detection model.
- **Speaker embeddings** are numerical representations of the segments of speech created in the previous step. MFCCs have been used in the past, but more sophisticated measures have seen more success. The i-vector embedding [11], for example, models the speakers' vocal tracts. Further improvements again have been made with x-vectors [48], which use deep neural networks to generate embeddings.
- **Clustering** involves grouping the speech segment embeddings according to their similarity. If grouped correctly, each cluster will contain speech segments produced by the same speaker. Methods include mean-shift, agglomerative hierarchical clustering, and, more recently, spectral clustering.
- **Re-segmentation** seeks to improve the boundaries estimated by clustering, as a final post-processing step before outputting the final predictions. Gaussian mixture models and hidden Markov models have been used for this purpose in the past.

3.2.2 Pyannote Model

Bredin et al. [20] jointly optimise these diarization sub-tasks, as opposed to training them each in isolation, yielding state-of-the-art performance. Of these sub-tasks: speech activity detection, speaker change detection, and re-segmentation are treated as sequence-labelling tasks. This enables the use of the `PyanNet` neural network architecture, described in 2.4.2, in each case. Tunable parameters include a threshold for speech activity detection, which involves binarizing a continuous probability as with

overlapped speech detection, and a similar threshold for speaker change detection. Speaker embeddings are generated using a metric learning approach. This means that they are designed to optimise for a given vector similarity metric, e.g. cosine similarity, such that segments that sound similar have similar embeddings according to that similarity metric. This makes the clustering step easier, as straight-forward clustering methods can be applied.

As is the case for overlapped speech detection, `pyannote.audio` provides pre-trained models trained on the AMI corpus [43] and dataset for the DIHARD challenge [44]. When given a `.wav` file as input, these models produce a pyannote `Annotation` such as is shown in figure 3.3. In this case, `Annotation` labels serve as speaker names.

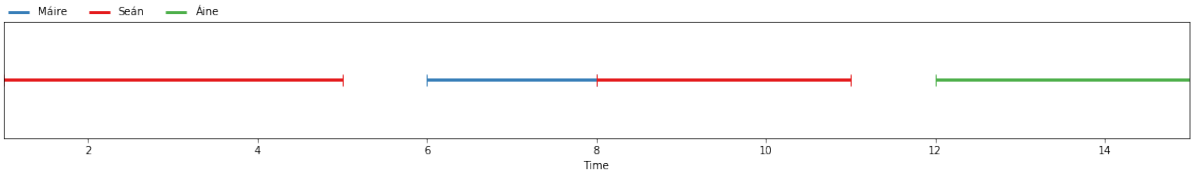

Figure 3.3: Speaker diarization output plot.

3.2.3 Speaker Embedding Visualisation

Speaker embeddings were plotted in two dimensions using a dimensionality reduction technique called t-SNE [49]. This technique projects high dimensional data points into lower dimensions in such a way that the spatial relationship between points is maintained, so that they can easily be plotted and examined visually. This technique was used to visualise speaker embeddings created by the `pyannote` speaker diarization pipeline. The embeddings should naturally form clusters where they represent speech from the same speaker. Plotted embeddings can be coloured according to their speaker using a 'correct' reference annotation (e.g. a manual diarization annotation). This way, the performance of the embedding algorithm can be validated visually: examining whether speakers of the same colour are clustered together and, if not, which speakers. An example t-SNE plot of speaker embeddings on the file `C0051` from *Corpus na Cainte Beo* is given in figure 3.4.

3.2.4 Guided Speaker Clustering

The clustering used by the pre-trained `pyannote.audio` models used is unsupervised. That is to say that the clustering algorithm is given no information about the embeddings that it must group together – it simply finds the most likely groupings using probabilistic methods. Figure 3.5 illustrates this process, showing on the left a plot of unlabelled speaker embeddings, and on the right the labelling that has been applied by the clustering algorithm. As per the metric learning approach to embedding, the segments from the same speaker are nearby one another in the embedding vector space, and so the clustering algorithm can easily identify three groups.

The *Corpus na Cainte Beo* transcription files specify the speaker for each fragment of speech, enabling calculation of the number of speakers present in a given file. The transcription files thus provide information relevant to diarization that was not being used by the `pyannote` system, and it was hypothesised that informing the clustering

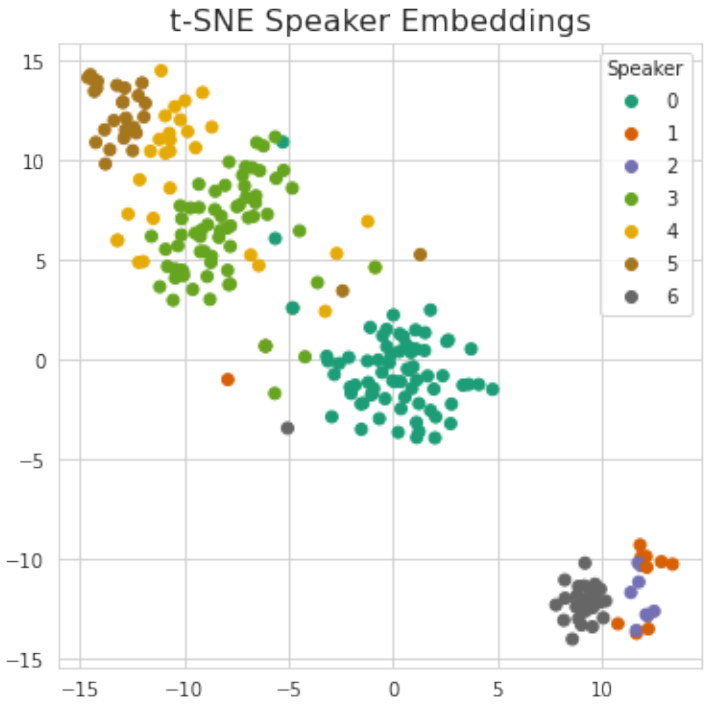

Figure 3.4: t-SNE visualisation of speaker embeddings for `C0051`.

algorithm of the number of clusters to be assigned may improve performance. This process may be considered a form of *guided speaker clustering*, in which the clustering algorithm is given a number of clusters that it must fit to the data in advance, rather than estimating that number itself based on the data. To this end, the `pyannote.audio` diarization pipeline was split into its component parts (i.e. speech activity detection, speaker change detection, speaker embedding), each of which are provided as pre-trained models that can be used in their own right. The clustering module, however, was replaced with algorithms for which a number of clusters could be specified in advance. Both Gaussian mixture model clustering and spectral clustering were considered as options here. Decriptions of these algorithms are as follows, based on sci-kit learn documentation:

- **Gaussian mixture model** involves using the expectation-maximization (EM) algorithm to fit a mixture of Gaussian models to the data. The number of Gaussians to be fit can be specified in advance. The EM algorithm consists of two steps that are applied iteratively: the expectation (E) step computes the probability that each data point was generated by each of the Gaussian components. Then, the maximisation (M) step tweaks the Gaussians to maximise the likelihood of the data. These tweaked Gaussians can then be used to compute the probabilities for the next E step, and so on. By applying these steps repeatedly, it is guaranteed that the Gaussians will fit the data in a way that is locally optimal (multiple random parameter initialisations can be considered to increase likelihood of finding global optimum). Once the algorithm has converged, each Gaussian represents a cluster.
- **Spectral clustering** uses methods from graph theory and linear algebra. First, an affinity matrix must be calculated. This is a square $n \times n$ matrix in which each

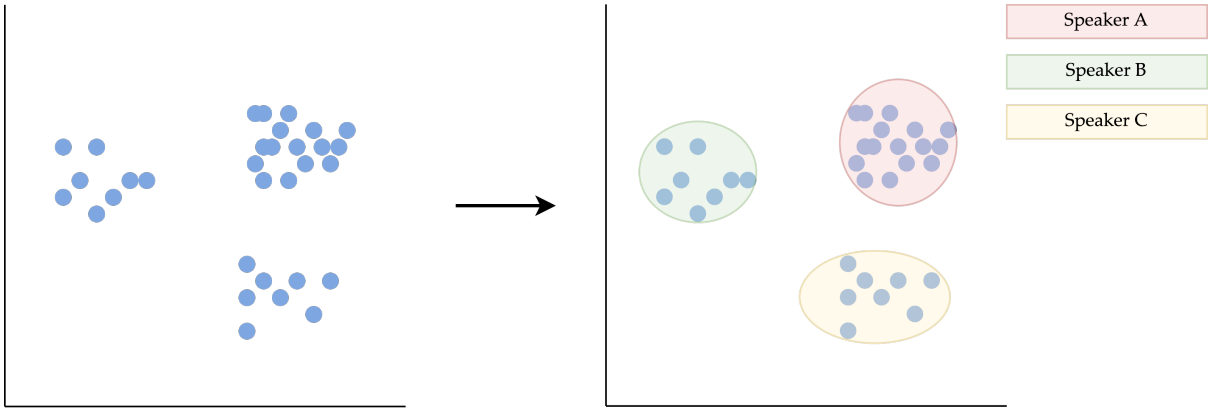

Figure 3.5: Unsupervised speaker clustering.

entry (i, j) gives the similarity between the i^{th} segment embedding and the j^{th} segment embedding. Cosine similarity, which measures the similarity in direction of two vectors, is used as a similarity metric to generate this affinity matrix. Eigenvectors of the Laplacian of this affinity matrix are then clustered using the k-means algorithm, which is a standard clustering method similar to GMM described above. These eigenvectors can then be used to label the segments.

These algorithms, while performing the same job (clustering), do so in different ways, and yield different results. Python implementations from sci-kit learn were used to apply these algorithms.

3.3 Diarization Tools Scripts

In order to run experiments on the pre-trained models described above with ease during the project, they were packaged in a number of python scripts, available on GitHub[2]. Both overlapped speech detection and speaker diariztion models are included. These scripts run the chosen model with specified parameters, and output the results in an RTTM-like file format. See appendix A1.2 for a description of the RTTM format.

3.4 Forced Alignment Filtering Algorithm

This section describes the forced alignment filtering algorithm. This algorithm was designed to filter the audio segments output by forced alignment according to suitability for use as ASR training data, or segment *quality* for short. Quality in this case is measured in terms of the following:

(i) The proportion of the segment that contains overlapped speech.

(ii) Confidence that the segment shows a correct alignment between transcription text and the corresponding speech.

Both measures of quality aim to ensure that the transcription text for each segment reflects as closely as possible the corresponding speech. In order to compute these

[2] `https://github.com/OisinNolan/Speaker-Diarization-Tools`

measures of quality, both overlapped speech detection and speaker diarization systems are used. A schematic for the overall system is given in figure 3.6.

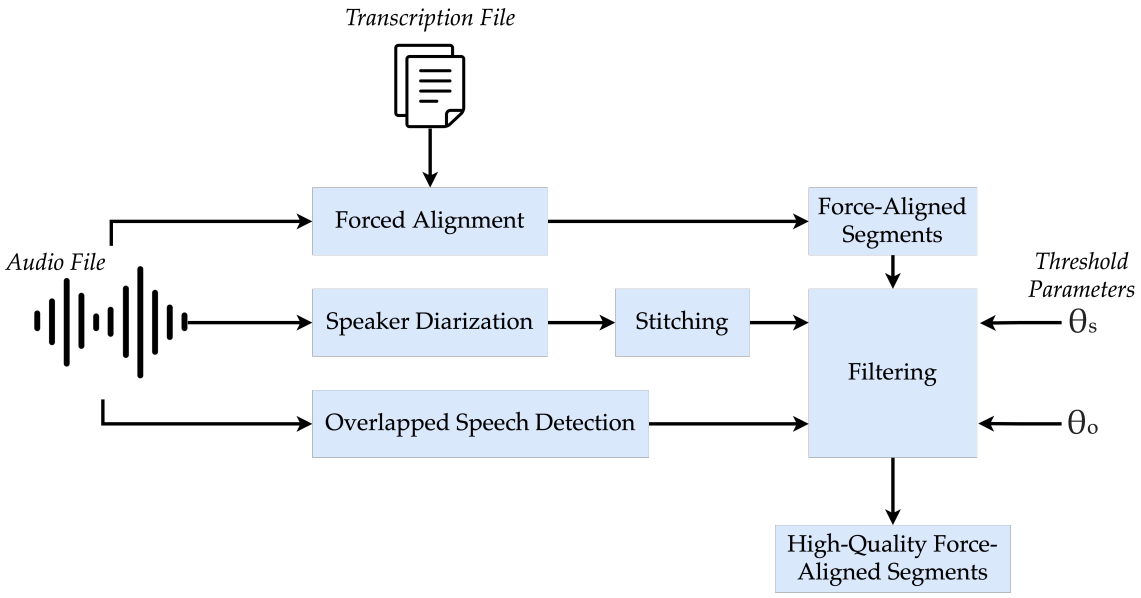

Figure 3.6: Forced alignment filtering.

The filtering algorithm itself takes an array of force-aligned segments as input, and outputs a subset of those segments that are deemed *high-quality* according to two parameters θ_o and θ_s which act as thresholds for overlap and diarization segment similarity measures. The filtering algorithm depends on two sub-routines, *alignment confidence* and *overlap marking*. Each of these algorithms is defined in detail in the following. Note that *predicted text length error* is not included in the system depicted by figure 3.6. This design choice is based on results discussed in section 5.3.

3.4.1 Alignment Confidence

Alignment confidence aims to gives a measure of the probability that a given force-aligned (FA) segment correctly aligns text and speech. This could be achieved in a number of ways, for example using an existing ASR system to compare the decoded speech to the transcription text for each segment. As mentioned previously, using existing ASR was not feasible in this case. Thus, two other methods were considered proposed, each of which are discussed in the following sections.

Predicted Text Length Error

This method looks to the relationship between text fragment length and segment duration to predict alignment error. The idea is that segments whose text length differs greatly from the expected text length given segment duration may be poorly aligned. For example, given the text *'dia duit'* we may expect a relatively short segment duration, maybe a few seconds. If this segment was much longer, maybe fifty seconds, we might make an educated guess that it is misaligned. In order to capture the expected text length given duration, we train a linear regression model on segment durations to

predict text lengths. Sci-kit learn's linear regression implementation[3] was used. Linear regression involves calculating a line of best fit given a set of data points and labels, such that the linear relationship between the data is captured. This line of best fit is essentially a function $f(.)$ that can then be used to predict text length, $\hat{y}$ given a new, previously unseen, segment duration, x (eq. 1). The absolute difference between actual and predicted text length (eq. 2) was used to derive alignment confidence – it was hypothesised that lower text length error might give high alignment confidence.

$$f(x) = \hat{y} \tag{1}$$

$$\text{text length error} = |y_{true} - \hat{y}| \tag{2}$$

Diarization Segment Simlarity

Next, a more complex method was considered. This method involves calculating diarization speech segments and then seeing which of them is maximally similar to the FA segment in question. The intuition behind this is that an ideal diarization system and an ideal forced alignment system should produce exactly the same segmentation of a given audio clip. The difference is in the labels: diarization labels the segments with speaker ids, and forced-aligners label them with transcription text. The idea is that if both the diarization system and the forced-aligner compute a similar segment, then it is likely to be correct, and thus likely that the force-aligned transcription is correct. It is like comparing the force-aligned segments to an unlabelled ground-truth segmentation. Raw diarization segmentation, however, differs slightly from force-aligned segmentation. The difference lies in the fact that a diarization system takes speech activity detection into account, and so if a speaker stops speaking mid-sentence for a short period of time, the diarization system will break that sentence up into two speech segments. Contrast this with the forced-aligner, whose segments span from the start of a speaker utterance to the end, despite pauses or silence that may occur within that interval. This difference necessitated the *stitching* algorithm, a simple algorithm that combines, or *stitches together*, neighbouring segments if they are by the same speaker. When applied to the diarization segments, they become more comparable to the force-aligned segments. Pseudo-code for the stitching algorithm is given in algorithm 1, and a visual representation is given in figure 3.7.

[3] `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html`

Algorithm 1: Segment stitching

Input: a: `Annotation`
segment$_{i-1}$ ← a[0]
start ← segment$_{i-1}$.start
end ← segment$_{i-1}$.end
for *segment$_i$* ∈ *a* **do**
  if *a[segment$_i$]* ≠ *a[segment$_{i-1}$]* **then**
    end ← segment$_{i-1}$.end
    segment$_{stitch}$ ← [start, end]
    a$_{stitch}$[segment$_{stitch}$] ← a[segment$_{i-1}$]
    start ← segment$_i$.start
  end
end
Output: a$_{stitch}$: `Annotation`

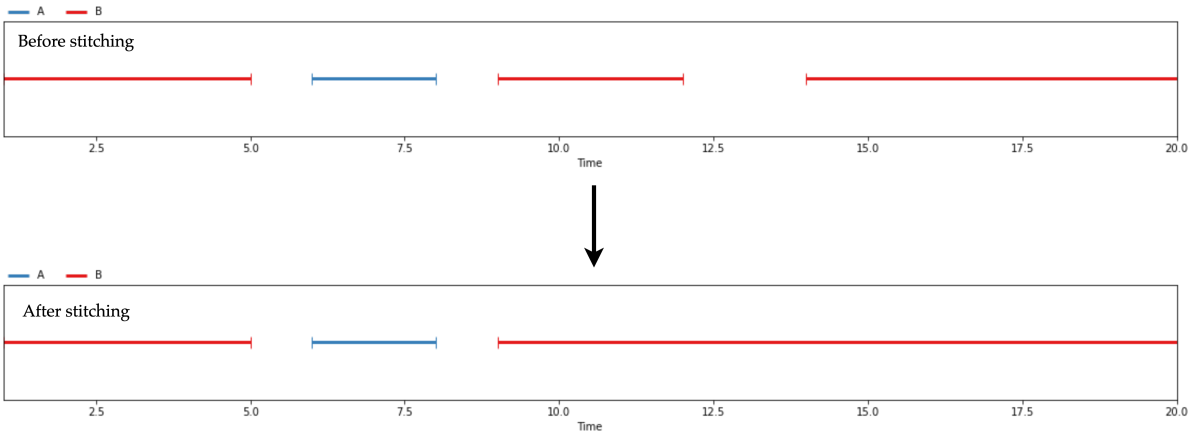

Figure 3.7: Segment stitching.

Once the diarization segments have been stitched, a similarity measure can be used to estimate how well each force-aligned segment has been aligned. This is computed using the *segment similarity* algorithm, which is defined in algorithm 2. This algorithm, given a force-aligned segment s, calculates the similarity between s and s', for each s' in the stitched diarization annotation. The similarity is calculated as the proportion of the larger of s and s' that is taken up by their intersection. This is defined in equation 3. Segment intersection is simply the duration of overlap between segments, as visualised by figure 3.8. Once similarity has been calculated for each s', the maximum of those similarities is returned. Thus, it is the similarity for the best possible fit that is returned for each force-aligned segment.

$$\Sigma(s, s') = \frac{s \cap s'}{max(s, s')} \tag{3}$$

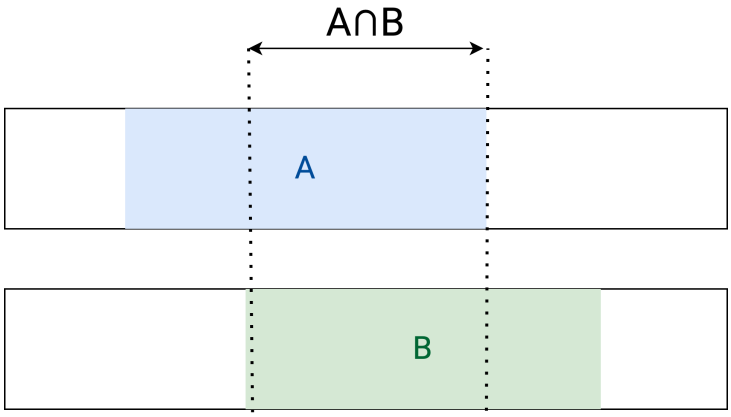

Figure 3.8: Segment intersection.

Algorithm 2: Segment similarity

Input: s: `Segment`, a_{stitch}: `Annotation`
similarities $\leftarrow$ []
for $s' \in a_{stitch}$ **do**
  similarities $\leftarrow$ similarities $\cup$ Σ(s, s')
end
similarity$_{max}$ $\leftarrow$ max(similarities)
Output: similarity$_{max}$: `float`

3.4.2 Overlap Marking

This sub-routine simply involves calculating the proportion of a given force-aligned segment that is occupied by overlapped speech. This proportion is calculated using the `pyannote.Timeline` `crop` operation which returns any segments in that timeline that fall within a given interval, specified as a segment. In this case, the force-aligned segment is used to crop a `Timeline` containing the overlapped speech for the audio clip in question. This cropping is performed in *intersection* mode, denoted crop-$\cap$, meaning that segments that fall at the border of the crop interval are trimmed. See `pyannote.core` docs[4] for more details on cropping. The duration of each cropped overlap segment is then summed together to get the total amount of overlap covering this force-aligned segment. The proportion of the force-aligned segment duration that is overlap is then returned. In algorithm 3, overlap$_\cap$ is a `Timeline` containing intersecting overlapped speech segments.

Algorithm 3: Overlap marking

Input: s: `Segment`, overlap: `Timeline`
overlap$_\cap$ $\leftarrow$ overlap.crop-$\cap$(s)
π_{ovl} $\leftarrow$ overlap$_\cap$.duration / s.duration
Output: π_{ovl}: `float`

[4] `http://pyannote.github.io/pyannote-core/reference.html#pyannote.core.Timeline.crop`

3.4.3 Filtering

The filtering algorithm uses the above algorithms 1 - 3 to filter force-aligned segments according to quality. Two parameters, θ_o and θ_s may be tuned to decide the level of quality required both in segment similarity and proportion overlap for a force-aligned segment to make it through the filter. The algorithm is fully specified in algorithm 4. The algorithm assumes that a diarization `Timeline` and an overlapped speech `Timeline` have already been calculated.

Algorithm 4: Filtering

Input: fa: `Annotation`, overlap: `Timeline`, diarization: `Timeline`, θ_o: `float`, θ_s: `float`

$\text{fa}_{filter} \leftarrow []$

$\text{diarization}_{stitch} \leftarrow \text{stitch(diarization)}$

for $s_{fa} \in fa$ **do**

  $\text{similarity}_{max} \leftarrow \text{maxSimilarity}(\text{s}_{fa}, \text{diarization}_{stitch})$

  $\pi_{ovl} \leftarrow \text{markOverlap}(\text{s}_{fa}, \text{overlap})$

  if $similarity_{max} \geq \theta_s$ ***and*** $\pi_{ovl} \leq \theta_s$ **then**

    $\text{fa}_{filter} \leftarrow \text{fa}_{filter} \cup \text{s}_{fa}$

  end

end

Output: fa_{filter}: `Annotation`

4 Evaluation Metrics

This section describes a number of metrics that enable the evaluation of the various components of the forced alignment filtering algorithm. These metrics include *segment confusion*, *diarization error rate*, *alignment error*, and *word error rate*. Each are discussed in detail in the following sections.

4.1 Segment Confusion

This metric was implemented in python to provide a general means of comparing two same-length `pyannote Timelines`. This means that it could be applied to other domains that make use of `Timelines` or `Segments`. In this case, it was used to evaluate the performance of overlapped speech detection models, as discussed in section 5.1. Segment confusion involves first partitioning the timelines into sequences of fixed-length frames. The frame-based partitions are implemented as binary vectors, each frame containing a 1 if it is covered by overlapped speech (a segment in the timeline, more generally) or a 0 otherwise. Because the `Timelines` t and t' being compared must be of the same length, each frame f_i in t will have a corresponding frame f'_i in t'. Each f'_i is compared to f_i, and is classified as a *true positive*, *false positive*, *true negative*, or *false negative* accordingly. These comparison classifications are detailed in table 4.1. A visual depiction is given by figure 4.1. This metric is named *segment confusion* after confusion matrices, as it is confusion matrix statistics that are calculated on the binary vector representations of the timelines.

f'_i	f_i	Classification
1	1	True Positive (TP)
1	0	False Positive (FP)
0	1	False Negative (FN)
0	0	True Negative (TN)

Table 4.1: Segment confusion comparison classification.

4.2 Diarization Error Rate

Diarization error rate (DER) is a standard metric for computing the performance of speaker diarization systems. The predicted diarization (hypothesis) may be compared

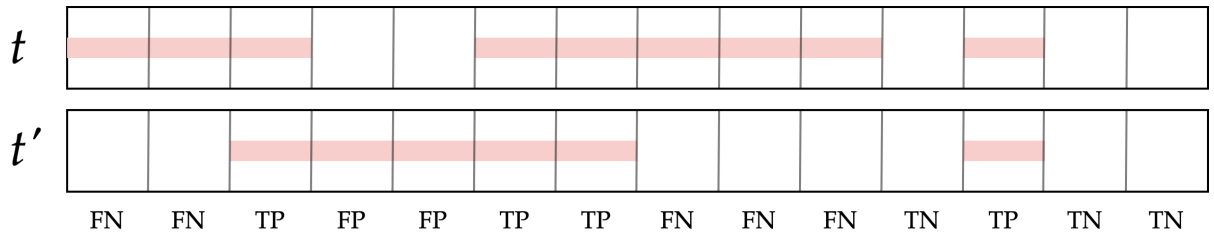

Figure 4.1: Segment confusion.

to a manually annotated diarization (reference) in order to calculate DER. DER aims to find the proportion of the diarized audio clip that is incorrect in some way. There are three ways in which a diarization system's output may be incorrect, for some region of the audio clip: false alaram, missed detection, or confusion. The following list defines each. A visual representation is given by figure 4.2.

- A **false alarm** occurs when the diarization system has predicted that there is speech in some region that does not contain speech.
- A **missed detection** occurs when the diarization system fails to predict that there is speech in some region that contains speech.
- **Confusion** occurs when the diarization system predicts that some speech was spoken by speaker A, when in fact that speech was spoken by speaker B, where $A \neq B$.

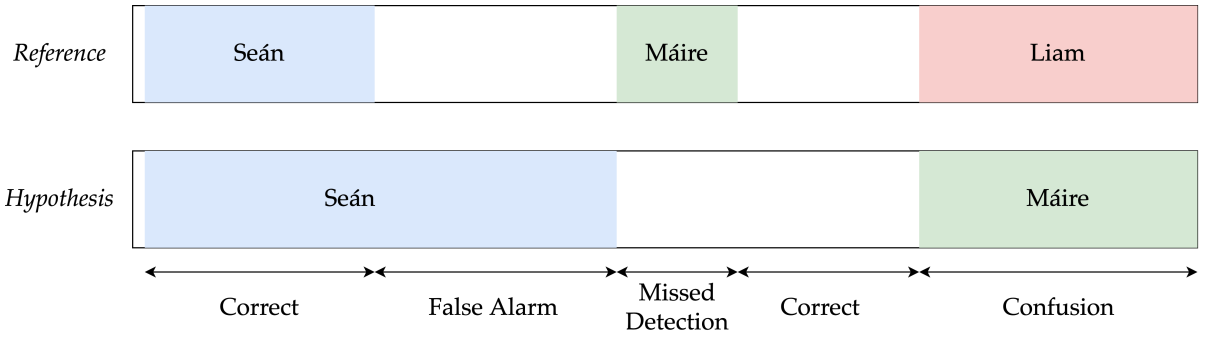

Figure 4.2: Diarization error rate.

The sum of the duration of all occurrences of the above errors is divided by the total duration of speech in the reference diarization to calculate the DER. This is given by equation 1. This formulation of DER is from Bredin [19], and the implementation provided in `pyannote.metrics`[1] was used accordingly.

$$\text{DER} = \frac{\text{false alarm} + \text{missed detection} + \text{confusion}}{\text{total}} \tag{1}$$

[1] `https://pyannote.github.io/pyannote-metrics/`

4.3 Alignment Error

Alignment error is a measure of of difference between two segments. Our alignment error function defines the error in a predicted segment s' given a reference segment s in terms of the difference between the start of s and s' and the difference between their ends, similar to Keshet et al. [50]. This function is given by equation 2, where $s.start$ and $s.end$ give the start and end times of segment s, respectively. The mean absolute difference is taken so that the difference intuitively corresponds with seconds. Visual representation given by figure 4.3.

$$e(s, s') = \frac{|s.start - s'.start| + |s.end - s'.end|}{2} \tag{2}$$

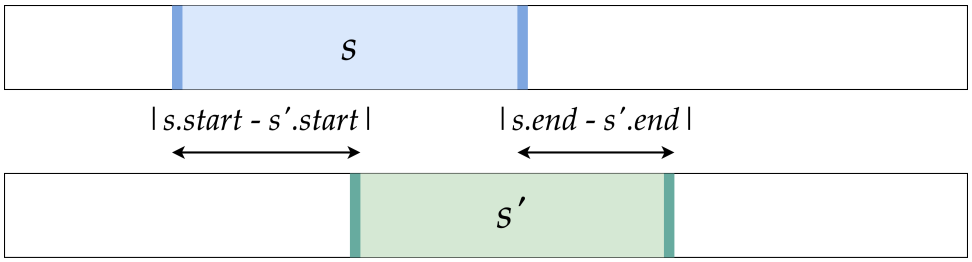

Figure 4.3: Alignment error.

4.4 Word Error Rate

Word error rate (WER) is the standard evaluation metric for ASR systems. WER is not, however, without criticism, and new, more nuanced ASR evaluation metrics may emerge in future [51]. More sophisticated measures will likely be favoured in the future, in particular for Irish and other highly inflected languages [13]. WER was used for the purposes of this project, however. WER is based on the Levenshtein distance [29] algorithm, which is a metric for calculating the difference between two strings. WER differs in that it works on the word level rather than the character level. It is calculated in terms of the normalised numbers of word substitutions, insertions, and deletions decoded by the ASR compared to the reference text. A substitution involves replacing or mis-transcribing a word, insertion involves adding a word that isn't present in the reference, and deletion involves leaving out a word that is in the reference. The formula for WER is given in equation 3. S, D, I are the numbers of substitutions, deletions, and insertions, respectively, and N is the total number of words in the reference.

$$WER = \frac{S + D + I}{N} \tag{3}$$

5 Experiments & Results

This chapter describes a number of experiments that were carried out in order to evaluate the performance of the various methods discussed in chapter 3. In each case, one or more evaluation metrics from chapter 4 were used. Evaluation was carried out on overlapped speech detection (OSD) and speaker diarization (DIA) models as a means of ensuring that their performance was such that they could be used in the main filtering algorithm. This evaluation enabled informed parameter tuning and model selection for both OSD and DIA. Having chosen suitable parameters and models, two possible measures of alignment confidence for use in the filtering algorithm were evaluated and compared. The filtering algorithm as a whole was then evaluated in terms of the impact that the training data acquired through filtering had on word error rate (WER) of the ASR system. The following sections provide setup description, results, and discussion of the experiments that determined the design choices made in the filtering system. Evaluations were typically carried out by comparing predicted annotations to manual annotations that were created to serve as 'correct' references for each task. The files chosen to be manually annotated are identified by `C0051` and `C0057`, briefly described in section 2.2.1.

5.1 Overlapped Speech Detection

5.1.1 Experimental Setup

Experiments were conducted on pre-trained `pyannote` overlapped speech detection models in order to choose parameter values and model type. Two different pre-trained models were available, one trained on the AMI meeting corpus [43] and the other on data from the first DIHARD challenge [44]. Thus, one choice to be made was between those: AMI model or DIHARD model? The value of the threshold parameter described in section 3.1.2 also had to be chosen. The experiment involved calculating segment confusion statistics (see section 4.1) at various parameter values and various models. According to the results, an optimal configuration was then chosen. In order to represent segment confusion in an intuitive way, true positive and false positive rates were calculated by normalising across true and false confusion statistics respectively. This normalisation is given by equations 1 and 2, and a confusion matrix is plotted in figure 5.1 for reference.

$$\text{TP rate} = \frac{TP}{TP + FN} = \frac{TP}{\text{Overlap}} \tag{1}$$

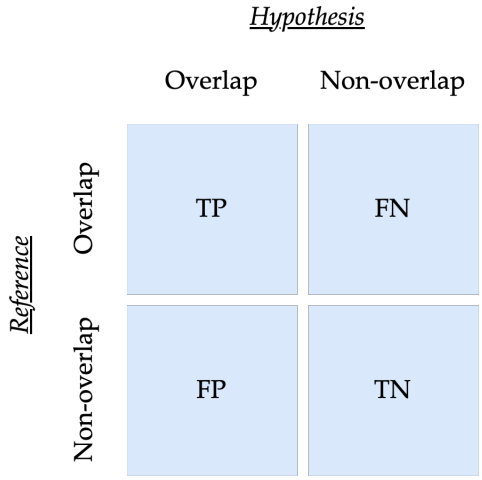

Figure 5.1: Overlap confusion matrix.

$$\text{FP rate} = \frac{FP}{FP + TN} = \frac{FP}{\text{Non-overlap}} \tag{2}$$

These true / false positive rates communicate important information on how the OSD model will impact data filtering. TP rate indicates the proportion of overlapped speech in the reference that was *correctly identified*. FP rate, on the other hand, indicates the proportion of non-overlap that was *falsely identified* as overlap. In terms of filtering the data, TP rate tells us the proportion of overlap that we have successfully removed from the dataset, and FP rate tells us how much potentially good data we have lost. TP rate and FP rate for both AMI and DIHARD models were calculated for threshold parameter values $\theta_{osd} \in \{0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$. θ_{osd} is short for binarization onset and offset. The results are given in tables 5.1 and 5.2, and plotted in figures 5.2 and 5.3.

θ_{osd}	AMI TP rate	AMI FP rate	DIHARD TP rate	DIHARD FP rate
0.01	1.0	0.903	0.988	0.428
0.05	0.994	0.492	0.906	0.129
0.1	0.982	0.295	0.815	0.063
0.2	0.911	0.122	0.727	0.027
0.3	0.802	0.059	0.654	0.016
0.4	0.691	0.03	0.583	0.009
0.5	0.569	0.018	0.503	0.006
0.6	0.443	0.010	0.416	0.004
0.7	0.311	0.005	0.319	0.003
0.8	0.199	0.002	0.254	0.001
0.9	0.072	0.000	0.164	0.000

Table 5.1: C0051 OSD performance.

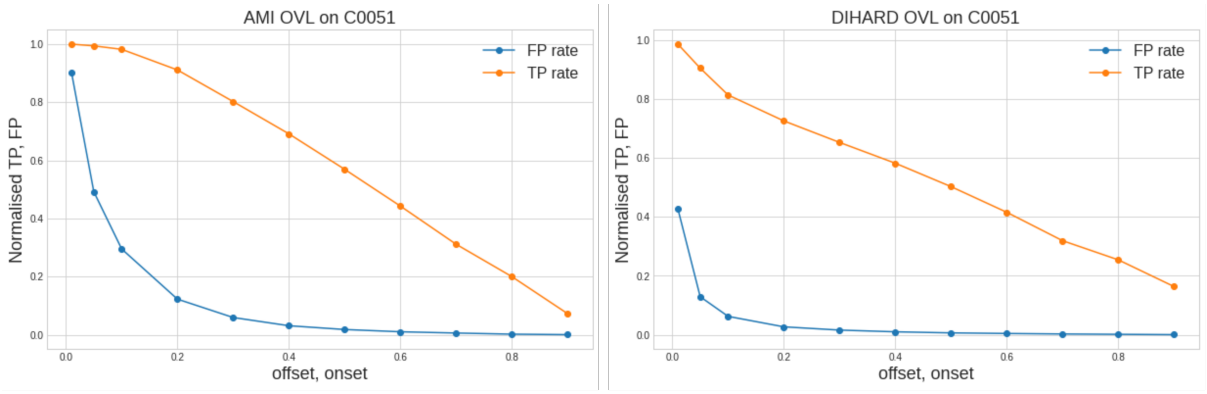

Figure 5.2: TP rate, FP rate for OSD on C0051.

θ_{osd}	AMI TP rate	AMI FP rate	DIHARD TP rate	DIHARD FP rate
0.01	1.0	0.727	0.989	0.430
0.05	0.998	0.483	0.886	0.159
0.1	0.968	0.295	0.814	0.085
0.2	0.896	0.137	0.704	0.038
0.3	0.778	0.074	0.562	0.022
0.4	0.685	0.047	0.483	0.012
0.5	0.549	0.030	0.392	0.008
0.6	0.383	0.018	0.284	0.004
0.7	0.277	0.01	0.204	0.002
0.8	0.181	0.005	0.159	0.001
0.9	0.074	0.001	0.083	0.000

Table 5.2: C0057 OSD performance.

5.1.2 Results

The results presented above are generally positive, and show that the pre-trained `pyannote` OSD models perform well on this unseen dataset. While the AMI and DIHARD models differ slightly in TP rate and FP rate for a given θ_{osd}, the TP rate relative to FP rate, i.e. $\frac{\text{TP rate}}{\text{FP rate}}$, is generally the same. TP rate was affected differently by θ_{osd} than FP rate. TP rate grows somewhat linearly as θ_{osd} decreases, whereas FP rate grows exponentially. These results enabled us to choose a value for θ_{osd} that maximises TP rate while minimising FP rate. The configuration that was settled upon, was {model = AMI; θ_{osd} = 0.3}. This configuration successfully removes ~80% of the overlapped speech while losing ~6% of non-overlapped speech in `C0051`, with similar results in `C0057`. While ideally the performance would be evaluated across the entire dataset, it was reassuring to see that the results were so similar on such different audio clips.

5.1.3 Descriptive Statistics

Descriptive statistics for the overlap segments in both the reference (manually annotated) and hypothesis (predicted by model) were also compared in order to give further insight into differences between the two. These statistics include % *overlap*, the

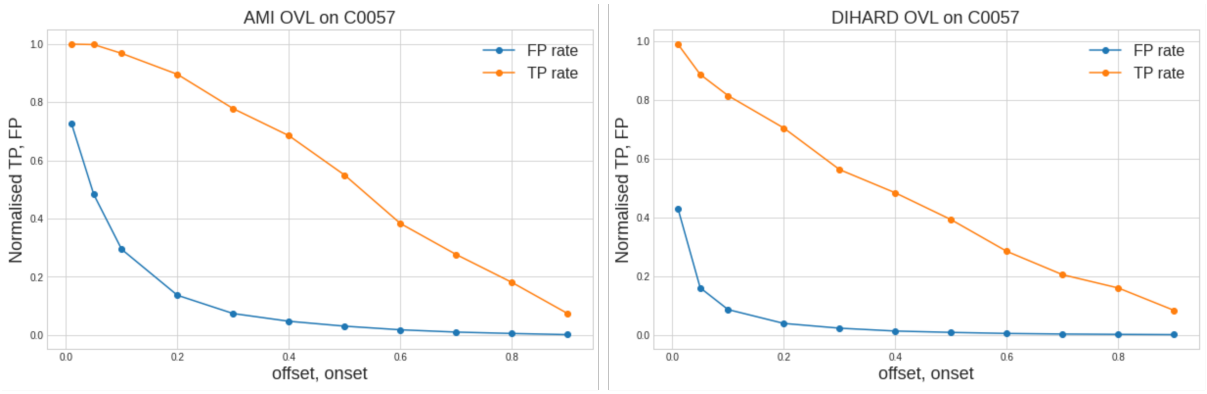

Figure 5.3: TP rate, FP rate for OSD on C0057.

percentage of the clip that was annotated as overlapped speech, the number of overlapped segments annotated, and the mean and standard deviation (S.D.) of those segments' durations. These statistics are presented in tables 5.3 and 5.4, and plotted in figure 5.4. The hypothesis in these tables and plots was generated using a model with the configuration chosen above, namely {model = AMI; θ_{osd} = 0.3}. In general, the hypothesis annotation contained more overlap than the manual annotation. This is not surprising, as ~6% of the non-overlap in the clip was falsely labelled as overlap by the OSD model. The model appeared to predict overlap segments of a similar mean length in both audio files, whereas the manual annotation shows a greater difference between the two. As is clear in figure 5.4, while the OSD model predicts many more segments than manual annotation, the shapes of segment duration distributions are quite similar per file.

Annotation	% overlap	# segments	Duration mean (s)	Duration S.D. (s)
Reference	5.6%	107	1.134	1.4
Hypothesis	10.16%	235	0.936	1.293

Table 5.3: C0051 OSD descriptive statistics.

Annotation	% overlap	# segments	Duration mean (s)	Duration S.D (s)
Reference	4.48%	148	0.499	0.246
Hypothesis	12.94%	250	0.835	0.715

Table 5.4: C0057 OSD descriptive statistics.

5.2 Speaker Diarization

5.2.1 Experimental Setup

As with OSD, a number of experiments were conducted in order to choose an optimal model for speaker diarization. Pre-trained diarization models from `pyannote` were available. The choices in model configuration lied in the model training dataset (AMI / DIHARD) and the clustering method (end-to-end / individual modules + guided

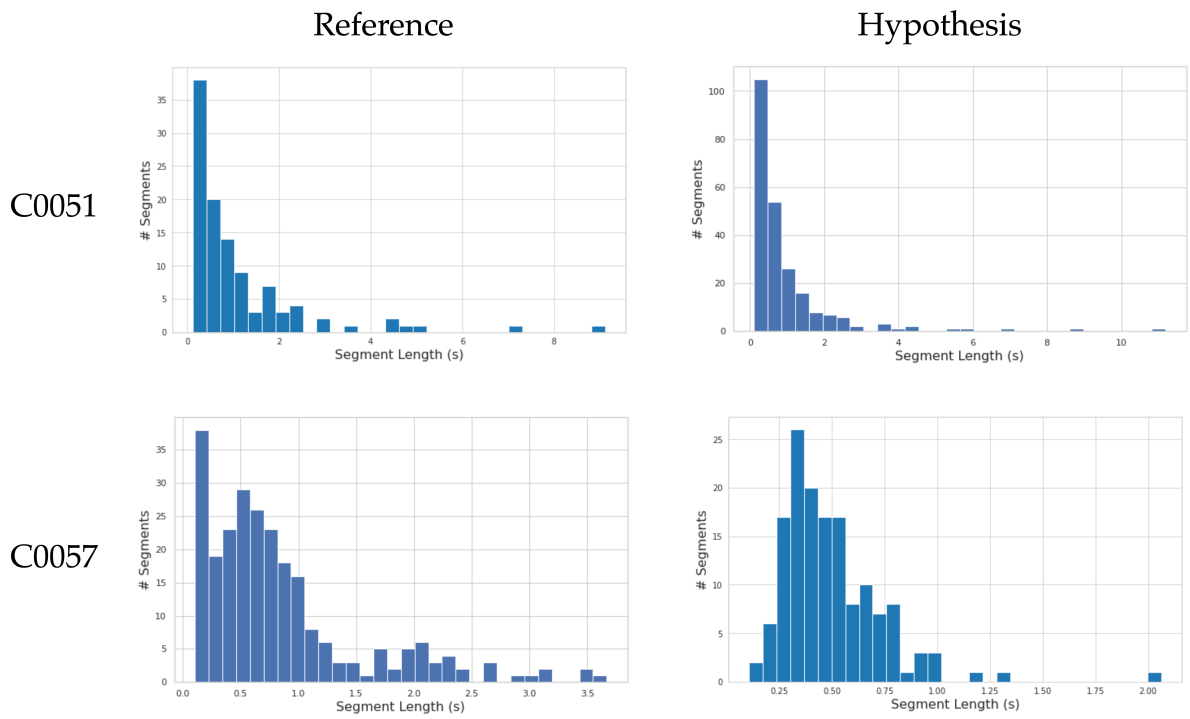

Figure 5.4: OSD segment descriptive statistics

speaker clustering). The sample audio files `C0051` and `C0057` were again manually annotated to be used as references. The diarization error rate (DER), described in section 4.2, was calculated for annotations generated by both AMI and DIHARD models on both `C0051` and `C0057`.

Audio	Model	DER	Confusion	Correct	False alarm	Missed	Total
C0051	DIHARD	**0.127**	123.372	1881.94	84.437	53.216	2058.529
C0051	AMI	0.282	474.864	1578.147	99.177	5.518	2058.529
C0057	DIHARD	0.608	666.278	683.08	115.584	96.93	1446.289
C0057	AMI	**0.263**	184.777	1252.372	187.143	9.14	1446.289

Table 5.5: Diarization model performance.

5.2.2 Results

These DER results are presented in table 5.5 (*'Missed detection' shortened to 'Missed' for formatting). The best results were achieved on `C0051` using the DIHARD model. This is interesting because `C0051` contains speech by seven different speakers, which one might assume would lead to poorer performance. On both files, a significant difference in performance is evident between models. For `C0051`, the DIHARD model shows a 54% relative improvement on the AMI model. For `C0057`, the AMI model shows a 57% relative improvement on the DIHARD model. It is noteworthy that a different model performs better in each case. This is likely reflective of the differences in audio characteristics between the files, in particular the number of speakers: in both cases DIHARD predicts more, and AMI less, speakers. This may be indicative of bias

with respect to number of speakers in the AMI and DIHARD datasets, although this would need to be explored in its own right. Table 5.6 shows the number of speakers predicted versus actual number of speakers in both files. Interestingly, the AMI model predicted that there were four speakers in `C0051`, although it barely assigned two of the labels to any of segments. In general, the models' performances seem to differ most in *confusion*, which is determined in large part by the clustering step. Based on these results, it was hypothesised that, in general, the AMI model may perform better on few-speaker audio, and the DIHARD model better on many-speaker audio. This could be validated by writing a script that counts for each file the actual versus predicted number of speakers for each model and calculates which model is more accurate in which scenarios. Speech segment distributions for predicted (hypothesis) versus actual (reference) diarization annotations are plotted in figure 5.5. Upon examining the speech segment distributions, it is clear that, in general, the reference annotation contains more, typically shorter, segments, while the hypothesis contains less, typically longer, segments.

Audio	Model	Predicted # speakers	Actual # speakers
C0051	DIHARD	7	7
C0051	AMI	4	7
C0057	DIHARD	4	2
C0057	AMI	4	2

Table 5.6: Predicted versus actual number of speakers.

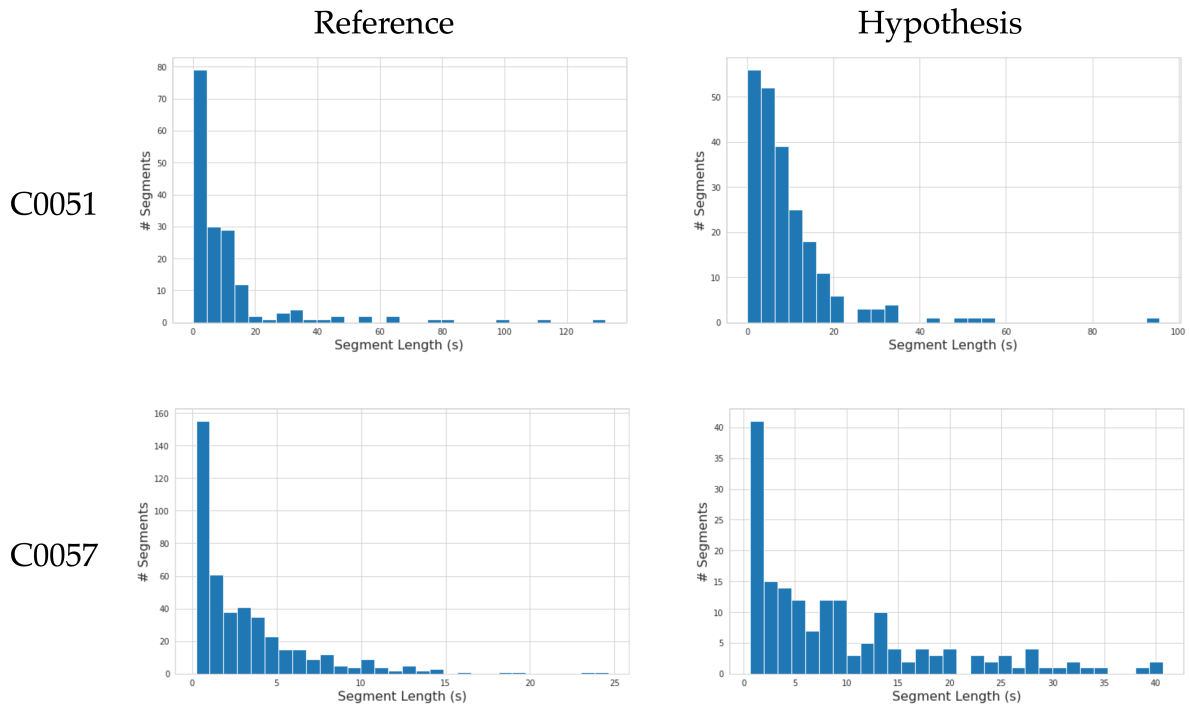

Figure 5.5: Speech segment distributions.

5.2.3 Guided Speaker Clustering

It was hypothesised that guided speaker clustering, i.e. clustering in which the number of speakers is known in advance, could be used to improve DER. Guided speaker clustering is defined and discussed in section 3.2.4. Pre-trained `pyannote` models for the various diarization sub-tasks were used to identify speech clips by different speakers, i.e. the pipeline: *speech activity detection* $\rightarrow$ *speaker change detection* $\rightarrow$ *speaker embedding* was applied. These sub-task models were available trained on either AMI or DIHARD datasets, as with the OSD and DIA models. Thus, models were chosen according to the best performing DIA model for each file. Specifically, DIHARD models were used for tests on `C0051`, and AMI models for tests on `C0057`. Both Gaussian mixture model (GMM) clustering and spectral clustering (SC) were applied to speech segments produced by the sub-task pipeline. The resulting annotations were evaluated using DER, given in table 5.7.

File	Method	DER	Confusion	Correct	False alarm	Missed	total
C0051	GMM	0.383	663.175	1336.2	66.821	59.155	2058.529
C0051	SC	**0.363**	621.933	1377.44	66.821	59.155	2058.529
C0057	GMM	0.311	260.784	1164.482	167.28	21.023	1446.289
C0057	SC	**0.294**	237.288	1187.978	167.28	21.023	1446.289

Table 5.7: Performance of guided speaker clustering methods.

Interestingly, these attempts at guided speaker clustering yielded poorer results than those achieved by the end-to-end diarization systems. In both cases, spectral clustering slightly outperforms Gaussian mixture model clustering. It is possible that sub-task models were optimized per task, rather than jointly optimized for diarization, as is the case with the end-to-end model, which may have caused poorer performance. Another possible explanation could be in the re-segmentation step, which was not applied in these experiments. This may be an avenue worth exploring further in future.

5.3 Alignment Confidence

It was important to evaluate the various measures of alignment confidence considered so that they could be compared and the best one chosen for use in the filtering system. In general, a variable that could be automatically calculated per force-aligned segment and that correlated with alignment error was required. Thus, it was this correlation that was evaluated for both *predicted text length error* and *diarization segment similarity*, the two measures of confidence that were considered for this project. In order to calculate alignment error correlation, alignment error was first calculated for each force-aligned segment. This was achieved using the *alignment error* metric defined in section 4.3. In order to calculate alignment error, an alignment for `C0051` was created manually in Praat (ideally, the following experiments would be validated with more manual alignments, but this was not feasible within the time constraints of this project). Plotting alignment error for each force-aligned segment in sequence gives a sort of error timeline throughout the interview, see figure 5.6. The plot is warped with respect to time, as each segment occupies an equal length on the plot, despite each

having distinct durations. A pattern seems to emerge in this plot: generally, we see spikes of misalignment, where the height of the spike appears to be loosely related to the number of segments it covers.

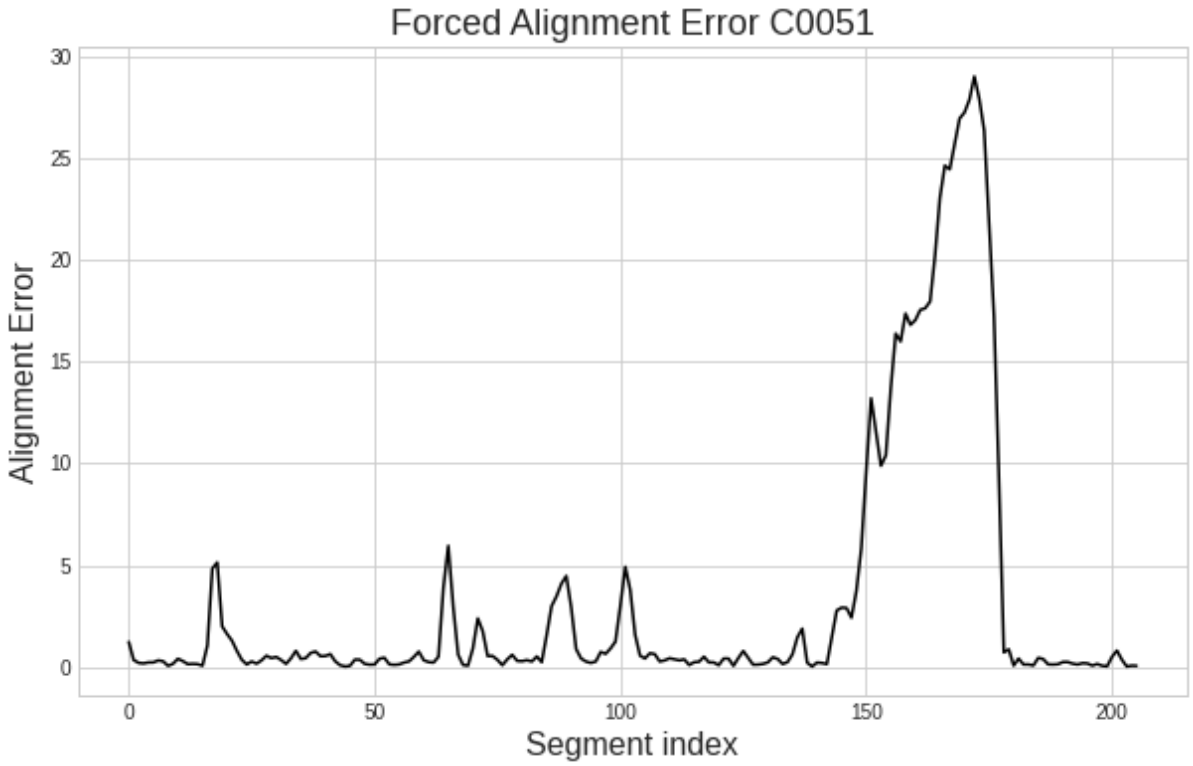

Figure 5.6: Alignment error for `C0051`.

5.3.1 Predicted Text Length Error

This measure of confidence aimed to predict alignment error based on the difference in expected text length versus actual text length given segment duration. In order to calculate expected text length, a linear regression model was fitted to the segment duration and text length data for the manually annotated reference alignment. The reference regression model showed strong correlation between segment duration and text length, with an R^2 score of ~95%. A model was also trained on the force-aligned segment durations and text lengths – the expectation being that there would be a poor correlation here, and that it would be exactly those segments that showed large discrepancies between segment duration and text length that were poorly aligned – but, interestingly, the force-aligned segments showed a correlation that was even stronger than the reference alignment (R^2 =~96%). This correlation implies that there is little information to be gained by examining the difference in predicted versus actual text length given duration. Indeed, predicted text length error showed practically no correlation with alignment error. Many segments that had exactly the predicted text length also had high alignment error. Plots of these correlations are given in figure 5.7. These results indicate that predicted text length error is not a viable measure of alignment confidence.

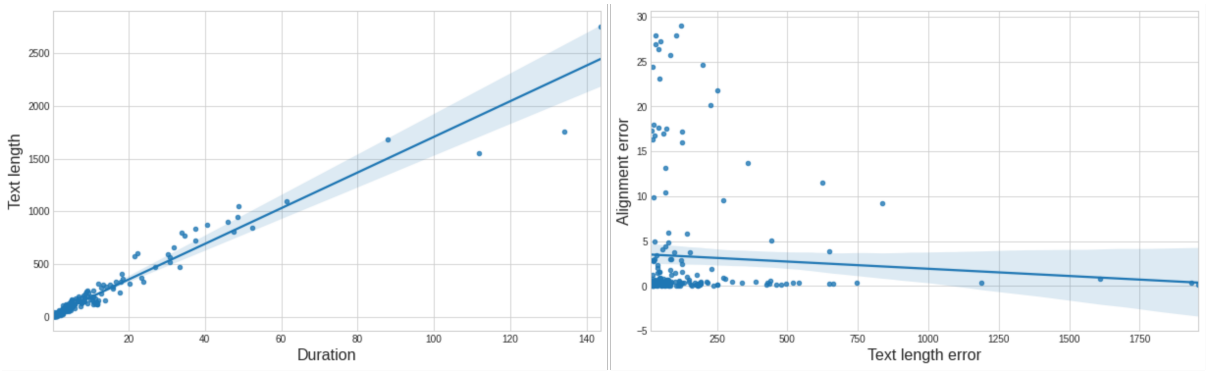

Figure 5.7: Force-aligned text/duration and text error/alignment error correlations.

5.3.2 Segment Similarity

Similarity to diarization segments was also considered as a measure of alignment confidence. The similarity score for each force-aligned segment was compared to the alignment error for that segment in order to examine correlations between the two. This experiment was carried out for both default (unstitched) diarization segments as well as stitched segments (see algorithm 1). Regression plots visualise the correlation between both stitched and unstitched diarization similarity in figure 5.8. Default diarization similarity shows practically no correlation with alignment error ($\rho = -0.016$). Stitched similarity shows a slight negative correlation with alignment error ($\rho = -0.158$). In general, neither account for much of the variance seen in the data. It is noteworthy, however, that segment similarity is thought of in terms of a threshold parameter which may be tuned. Thus, the typical alignment error per threshold interval is given in tables 5.8 and 5.9, and plotted as histograms in figures 5.9 and 5.10. On inspecting these results, a significant difference is seen between the stitched versus unstitched similarity measures, particularly for similarity ≥ 0.8. Unstitched similarity of ≥ 0.8 yields 17 segments with a mean alignment error of ~2s, whereas stitched similarity of ≥ 0.8 yields 36 segments with mean alignment error of ~0.8s. The unstitched mean alignment error has been pulled up by an outlier, but even ignoring that outlier it captures less than half the number of segments that the stitched similarity measure does. These results provide some validation for the stitching process.

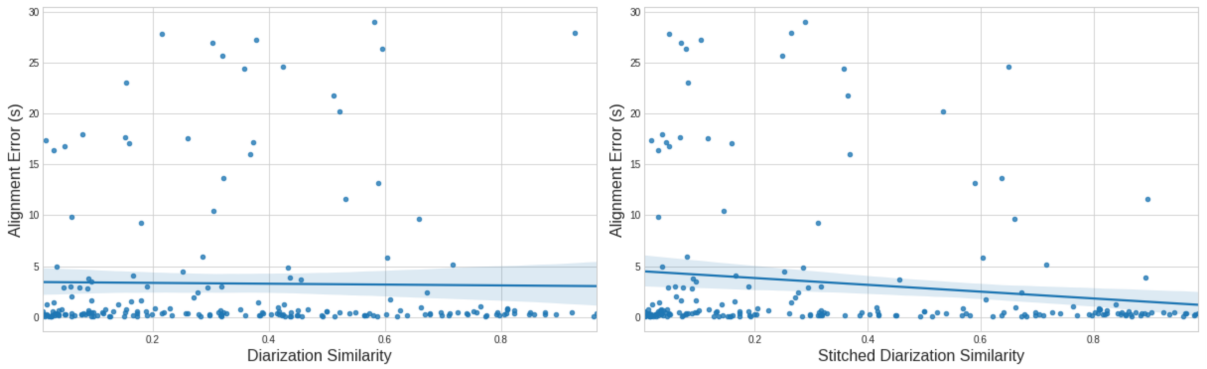

Figure 5.8: Default / stitched diarization similarity and alignment error.

Despite capturing a number of high-quality force-aligned segments, it is clear from

Threshold range	# segments	Mean alignment error
$[0, 0.2]$	78	2.597
$[0.2, 0.4]$	42	5.658
$[0.4, 0.6]$	41	4.123
$[0.6, 0.8]$	28	1.168
$[0.8, 1.0]$	17	2.046

Table 5.8: Unstitched segment similarity distribution statistics.

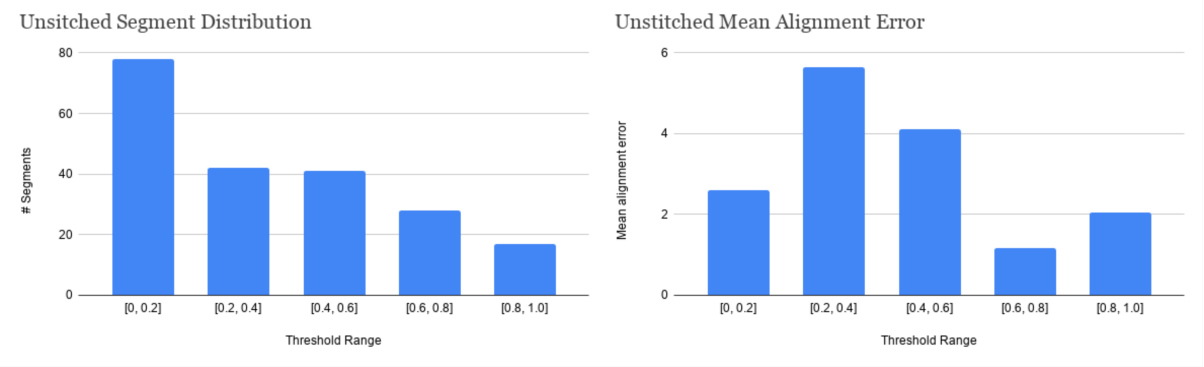

Figure 5.9: Unstitched segment similarity distribution plots.

looking at the bottom left corner of the right-hand plot in figure 5.8 that many segments with low alignment error also have low stitched similarity. Those high-quality segments would thus be discarded by the filtering algorithm. In order to understand why certain high-quality segments had such low similarity, a manual inspection was carried out in Praat. It appeared that, in many cases, the diarization system would fail to pick up on speaker change in short segments with a high concentration of speaker change. An illustration is given in figure 5.11, which shows the reference diarization beside the diarization predicted by a `pyannote` model. It is likely this general pattern that both contributes to DER and causes the filtering system to miss some high-quality segments. It appears that it is mostly short segments that are affected by this issue, although this has yet to be measured. This idea is evidenced in part by figure 5.5, which shows that the hypothesised segments were typically longer and less numerous than those found in the reference. Ultimately, fixing this issue is a matter of improving upon DER. Some small experiments were carried out with speaker change detection thresholds but with little improvement to DER. This is an area that could be worked on further in future.

Threshold range	# segments	Mean alignment error
[0, 0.2]	94	3.782
[0.2, 0.4]	32	5.654
[0.4, 0.6]	20	2.153
[0.6, 0.8]	24	2.846
[0.8, 1.0]	36	0.804

Table 5.9: Stitched segment similarity distribution.

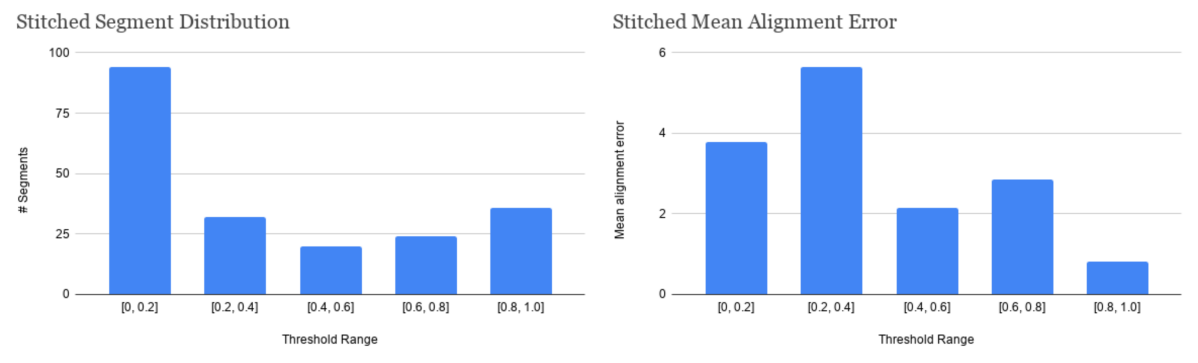

Figure 5.10: Stitched segment similarity distribution plots.

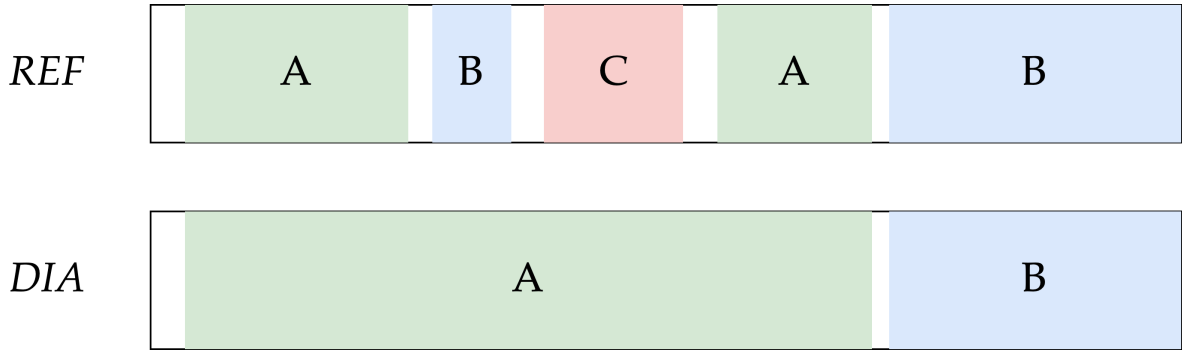

Figure 5.11: Manual inspection of diarization versus reference.

5.4 Impact on ASR System

5.4.1 Experimental Setup

A number of experiments[1] were carried out in order to evaluate the impact that training data extracted by the filtering algorithm would have on the performance of the ASR system. The experiments involved first applying the filtering algorithm to interviews with only two speakers from Corpus na Cainte Beo with the following parameter configuration: $\{\theta_s = 0.7, \theta_o = 0.05\}$. This extracted ~47 hours of training data in the form of force-aligned segments that were at least 70% similar to some diarization segment and contained no more than 5% predicted overlapped speech. Additionally, 47 hours of unfiltered force-aligned segments were extracted, to be used for comparison. Both the filtered and unfiltered training data were added to the origi-

[1]The WER results were calculated by Liam Longergan of the ABAIR research group.

nal training data. This created three distinct datasets: original training data; original training data + filtered segments; original training data + unfiltered segments. Each of these datasets was used to train a model with the architecture described in section 2.1.1. These models were then evaluated on two test sets: the Mozilla read-speech set and the Comhrá conversational-speech set. The Mozilla set contained read-speech by mostly non-native speakers, and the Comhrá set contained conversational speech by native speakers.

5.4.2 Results

The WER for each of the models on both test sets was calculated, and is presented in table 5.10.

Model	WER on Mozilla read-speech	WER on Comhrá conversational-speech
Baseline	22.49%	63.07%
Filtered	**21.44**%	**55.66**%
Unfiltered	27.91%	62.26%

Table 5.10: Impact of filtered segments on ASR system.

These results indicate that the filtered model is the best performing on both the Mozilla and Comhrá sets. Interestingly, adding random segments to the baseline model significantly degrades its performance. Adding filtered segments, on the other hand, improves the performance, even if only slightly. The improvement in performance on read-speech gives some indication that the filtering algorithm is indeed extracting high-quality, low-noise training data, with respect to unfiltered chosen segments – adding unfiltered segments significantly degrades performance on read-speech. Adding filtered segments results in a 23.5% improvement on the Mozilla set relative to adding unfiltered segments, and a 10.6% relative improvement on the Comhrá set. It is noteworthy that the Comhrá set is particularly noisy, and so most attention should be given to the relative improvement in WER, rather than absolute measures. In general, adding filtered segments to the training data resulted in a more accurate ASR system in both read and conversational speech. The additional improvement on conversational speech contributes to the robustness of the system, enabling it to recognise a wider range of speech. Compared to the initial baseline model, adding training data extracted by the filtering algorithm improved performance by a relative 4.67% on read-speech, and 11.75% on conversational-speech. A detailed account of the WER for each model on each dataset is presented in tables 5.11 and 5.12. The WER breakdowns show that, in general, the filtered models have far fewer substitution errors than the unfiltered. On conversational-speech test data, adding filtered or unfiltered training data significantly decreases deletion errors.

Model	Insertions	Deletions	Substitutions
Baseline	155	393	1602
Filtered	122	416	1511
Unfiltered	199	501	1968

Table 5.11: Mozilla test set WER breakdown.

Model	Insertions	Deletions	Substitutions
Baseline	14346	35632	38977
Filtered	13988	30290	34227
Unfiltered	15131	30437	42238

Table 5.12: Comhrá test set WER breakdown.

6 Conclusion

6.1 Summary

This project proposes a system for automatically extracting ASR training data from conversational speech. By definition, low-resource languages are less likely to have access to high quality speech corpora, such as audio-books. The proposed system aims to enable the gathering of speech data from a wide variety of sources, such as transcribed radio interviews, subtitled movies, and captioned YouTube videos, without any dependencies on pre-existing speech technologies for that language. Indeed, the pre-trained models used in the system perform language-independent tasks. The system first force-aligns the text and audio, and then filters the force-aligned segments according to a measure of confidence that the alignment is correct, based on speaker diarization, and a measure of the proportion of the clip that contains overlapped speech. By filtering according to alignment confidence and overlapped speech, the system aims to produce well-aligned, low-noise ASR training data. Filtering is defined by threshold parameters which may be modified in order to change the amount and quality of speech that is extracted. The system was validated by adding 47 hours of filtered training data to an existing ASR model, which resulted in a 4.67% relative improvement on a read-speech test set, and an 11.75% relative improvement on a conversational-speech test set.

6.2 Remaining Work

More work could be done on this project in validating the results presented. In particular, the performance of the overlapped speech detection and speaker diarization models was evaluated using two manually annotated interviews. By manually annotating more interviews, one could be more confident of the general performance of these models. Further parameter tuning could result in improved performance. The parameters used for the experiments described in section 5.4.1 were chosen somewhat intuitively. WER results could almost certainly be improved upon, even if only slightly, by evaluating performance at a number of different values for parameters θ_s and θ_o. Furthermore, end-to-end optimisation could be explored, in which parameters for overlapped speech detection and speaker diarization models are tuned to optimise WER. Another area that in which more work could be done is guided speaker clustering. Specifically, applying a re-segmentation algorithm, or new clustering methods could improve performance. It would also be interesting to explore the contribution that each of the components of the filtering systems makes to its impact on ASR sys-

tem WER. In particular, letting $\theta_s = 0$ will result in filtering only by overlapped speech content, and letting $\theta_o = 1$ will result in filtering only be alignment confidence. This may give insights into the characteristics of high-quality ASR training data.

6.3 Future Research

Improvements in ASR performance on conversational speech resulting from the filtering system may enable future research in this area. In particular, an improved measure of alignment confidence may be calculated by comparing text decoded by the ASR system to text in the transcription, as done by Mansikkaniemi et al. [28], Helgadóttir et al. [30]. This may be improved further by training a biased language model on the transcription text in Corpus na Cainte Beo. In this way, a semi-supervised approach may be followed, in which the ASR system is used to extract training data which is used to improve that ASR system iteratively. Another avenue to explore is that of source-separation, which may be used to mitigate the overlapped speech problem [36]. In general, improvements in speaker diarization and overlapped speech detection will also enable improvements in segment filtering, as the current models used in the system may be replaced. It would also be interesting to see the degree to which this system is applicable to speech sources of different kinds, such as subtitled TV programmes or captioned YouTube videos, as well as to sources in languages other than Irish.

Bibliography

[1] Ailbhe Ní Chasaide, Neasa Ní Chiaráin, Christoph Wendler, Harald Berthelsen, Andy Murphy, and Christer Gobl. The abair initiative: Bringing spoken irish into the digital space. In *INTERSPEECH*, pages 2113–2117, 2017.

[2] Kshitij Saxena, Robert Diamond, Reid F Conant, Terri H Mitchell, Guido Gallopyn, and Kristin E Yakimow. Provider adoption of speech recognition and its impact on satisfaction, documentation quality, efficiency, and cost in an inpatient ehr. *AMIA Summits on Translational Science Proceedings*, 2018:186, 2018.

[3] Ailbhe Ní Chasaide, Neasa Ní Chiaráin, Harald Berthelsen, Christoph Wendler, Andrew Murphy, Emily Barnes, and Christer Gobl. Can we defuse the digital timebomb? linguistics, speech technology and the irish language community.

[4] Rosana Ardila, Megan Branson, Kelly Davis, Michael Henretty, Michael Kohler, Josh Meyer, Reuben Morais, Lindsay Saunders, Francis M Tyers, and Gregor Weber. Common voice: A massively-multilingual speech corpus. *arXiv preprint arXiv:1912.06670*, 2019.

[5] Neasa Ní Chiaráin and Ailbhe Ní Chasaide. An scéalaí: autonomous learners harnessing speech and language technologies. In *SLaTE*, pages 94–98, 2019.

[6] R McGuirk. Exploration of the use of irish language synthesis with a screen reader in the teaching of irish to pupils with vision impairment. *Unpublished M. Phil. dissertation. Trinity College Dublin, Ireland*, 2015.

[7] Christopher Moseley. *Atlas of the World's Languages in Danger*. Unesco, 2010.

[8] Daniel Jurafsky and James H Martin. Speech and language processing (draft). *Chapter 26: Automatic Speech Recognition and Text-to-Speech (Draft of December 30, 2020). Retrieved April*, 2020.

[9] Alex Waibel, Toshiyuki Hanazawa, Geoffrey Hinton, Kiyohiro Shikano, and Kevin J Lang. Phoneme recognition using time-delay neural networks. *IEEE transactions on acoustics, speech, and signal processing*, 37(3):328–339, 1989.

[10] Vijayaditya Peddinti, Daniel Povey, and Sanjeev Khudanpur. A time delay neural network architecture for efficient modeling of long temporal contexts. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[11] Najim Dehak, Patrick J Kenny, Réda Dehak, Pierre Dumouchel, and Pierre Ouellet. Front-end factor analysis for speaker verification. *IEEE Transactions on Audio, Speech, and Language Processing*, 19(4):788–798, 2010.

[12] 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 *IEEE 2011 workshop on automatic speech recognition and understanding*, number CONF. IEEE Signal Processing Society, 2011.

[13] Mengjie Qian, Harald Berthelsen, Liam Lonergan, Andy Murphy, Claire O'Neill, Neasa Ní Chiaráin, Christer Gobl, and Ailbhe Ní Chasaide. Cross-dialect variation but no spoken standard: testing global and multi-dialect lexicons in irish automatic speech recognition. In *SUBMITTED INTERSPEECH*, 2021.

[14] Elizabeth Shriberg. Spontaneous speech: How people really talk and why engineers should care. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[15] Özgür Çetin and Elizabeth Shriberg. Overlap in meetings: Asr effects and analysis by dialog factors, speakers, and collection site. In *International Workshop on Machine Learning for Multimodal Interaction*, pages 212–224. Springer, 2006.

[16] Paul Boersma and Vincent Van Heuven. Speak and unspeak with praat. *Glot International*, 5(9/10):341–347, 2001.

[17] Michael McAuliffe, Michaela Socolof, Sarah Mihuc, Michael Wagner, and Morgan Sonderegger. Montreal forced aligner: Trainable text-speech alignment using kaldi. In *Interspeech*, volume 2017, pages 498–502, 2017.

[18] Vibha Tiwari. Mfcc and its applications in speaker recognition. *International journal on emerging technologies*, 1(1):19–22, 2010.

[19] Hervé Bredin. pyannote.metrics: a toolkit for reproducible evaluation, diagnostic, and error analysis of speaker diarization systems. In *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association*, Stockholm, Sweden, August 2017. URL `http://pyannote.github.io/pyannote-metrics`.

[20] Hervé Bredin, Ruiqing Yin, Juan Manuel Coria, Gregory Gelly, Pavel Korshunov, Marvin Lavechin, Diego Fustes, Hadrien Titeux, Wassim Bouaziz, and Marie-Philippe Gill. Pyannote. audio: neural building blocks for speaker diarization. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7124–7128. IEEE, 2020.

[21] Mirco Ravanelli and Yoshua Bengio. Interpretable convolutional filters with sincnet. *arXiv preprint arXiv:1811.09725*, 2018.

[22] Daniel Jurafsky and James H Martin. Speech and language processing (draft). *Chapter 9: Deep Learning Architectures for Sequence Processing (Draft of December 30, 2020). Retrieved April*, 2020.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[24] Christopher Olah. Understanding lstm networks. URL `https://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[25] Daniel Jurafsky and James H Martin. Speech and language processing (draft). *Chapter 7: Neural Networks and Neural Language Models (Draft of December 30, 2020). Retrieved April*, 2020.

[26] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an asr corpus based on public domain audio books. In *2015 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5206–5210. IEEE, 2015.

[27] Shafayat Ahmed, Nafis Sadeq, Sudipta Saha Shubha, Md Nahidul Islam, Muhammad Abdullah Adnan, and Mohammad Zuberul Islam. Preparation of bangla speech corpus from publicly available audio & text. In *Proceedings of The 12th Language Resources and Evaluation Conference*, pages 6586–6592, 2020.

[28] André Mansikkaniemi, Peter Smit, Mikko Kurimo, et al. Automatic construction of the finnish parliament speech corpus. In *INTERSPEECH*, volume 8, pages 3762–3766, 2017.

[29] Vladimir I Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710. Soviet Union, 1966.

[30] Inga Rún Helgadóttir, Róbert Kjaran, Anna Björk Nikulásdóttir, and Jón Guðnason. Building an asr corpus using althingi's parliamentary speeches. In *INTERSPEECH*, pages 2163–2167, 2017.

[31] Egor Lakomkin, Sven Magg, Cornelius Weber, and Stefan Wermter. Kt-speech-crawler: Automatic dataset construction for speech recognition from youtube videos. *arXiv preprint arXiv:1903.00216*, 2019.

[32] Cătălin Zorilă, Christoph Boeddeker, Rama Doddipatla, and Reinhold Haeb-Umbach. An investigation into the effectiveness of enhancement in asr training and test for chime-5 dinner party transcription. In *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, pages 47–53. IEEE, 2019.

[33] Jon Barker, Shinji Watanabe, Emmanuel Vincent, and Jan Trmal. The fifth'chime'speech separation and recognition challenge: dataset, task and baselines. *arXiv preprint arXiv:1803.10609*, 2018.

[34] Cătălin Zorilă and Rama Doddipatla. On reducing the effect of speaker overlap for chime-5. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 6645–6649. IEEE, 2019.

[35] Christoph Boeddeker, Jens Heitkaemper, Joerg Schmalenstroeer, Lukas Drude, Jahn Heymann, and Reinhold Haeb-Umbach. Front-end processing for the chime-5 dinner party scenario. In *CHiME5 Workshop, Hyderabad, India*, 2018.

[36] Cem Subakan, Mirco Ravanelli, Samuele Cornell, Mirko Bronzi, and Jianyuan Zhong. Attention is all you need in speech separation. *arXiv preprint arXiv:2010.13154*, 2020.

[37] Gail Jefferson. A case of precision timing in ordinary conversation: Overlapped tag-positioned address terms in closing sequences. *Semiotica*, 9(1):47–96, 1973.

[38] Elizabeth Shriberg, Andreas Stolcke, and Don Baron. Observations on overlap: Findings and implications for automatic processing of multi-party conversation. In *Seventh European Conference on Speech Communication and Technology*, 2001.

[39] Kofi Boakye, Beatriz Trueba-Hornero, Oriol Vinyals, and Gerald Friedland. Overlapped speech detection for improved speaker diarization in multiparty meetings. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 4353–4356. IEEE, 2008.

[40] Navid Shokouhi and John HL Hansen. Teager–kaiser energy operators for overlapped speech detection. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(5):1035–1047, 2017.

[41] Latané Bullock, Hervé Bredin, and Leibny Paola Garcia-Perera. Overlap-aware diarization: Resegmentation using neural end-to-end overlapped speech detection. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7114–7118. IEEE, 2020.

[42] Gregory Gelly and Jean-Luc Gauvain. Minimum word error training of rnn-based voice activity detection. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[43] Iain McCowan, Jean Carletta, Wessel Kraaij, Simone Ashby, S Bourban, M Flynn, M Guillemot, Thomas Hain, J Kadlec, Vasilis Karaiskos, et al. The ami meeting corpus. In *Proceedings of the 5th International Conference on Methods and Techniques in Behavioral Research*, volume 88, page 100. Citeseer, 2005.

[44] Neville Ryant, Kenneth Church, Christopher Cieri, Alejandrina Cristia, Jun Du, Sriram Ganapathy, and Mark Liberman. First dihard challenge evaluation plan. *2018, tech. Rep.*, 2018.

[45] Tae Jin Park, Naoyuki Kanda, Dimitrios Dimitriadis, Kyu J Han, Shinji Watanabe, and Shrikanth Narayanan. A review of speaker diarization: Recent advances with deep learning. *arXiv preprint arXiv:2101.09624*, 2021.

[46] Tim Ng, Bing Zhang, Long Nguyen, Spyros Matsoukas, Xinhui Zhou, Nima Mesgarani, Karel Veselỳ, and Pavel Matějka. Developing a speech activity detection system for the darpa rats program. In *Thirteenth annual conference of the international speech communication association*, 2012.

[47] Thilo Pfau, Daniel PW Ellis, and Andreas Stolcke. Multispeaker speech activity detection for the icsi meeting recorder. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001. ASRU'01.*, pages 107–110. IEEE, 2001.

[48] David Snyder, Daniel Garcia-Romero, Gregory Sell, Daniel Povey, and Sanjeev Khudanpur. X-vectors: Robust dnn embeddings for speaker recognition. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5329–5333. IEEE, 2018.

[49] Laurens Van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. *Journal of machine learning research*, 9(11), 2008.

[50] Joseph Keshet, Shai Shalev-Shwartz, Yoram Singer, and Dan Chazan. Phoneme alignment based on discriminative learning. In *Ninth European Conference on Speech Communication and Technology*, 2005.

[51] Benoit Favre, Kyla Cheung, Siavash Kazemian, Adam Lee, Yang Liu, Cosmin Munteanu, Ani Nenkova, Dennis Ochei, Gerald Penn, Stephen Tratz, et al. Automatic human utility evaluation of asr systems: Does wer really predict performance? In *INTERSPEECH*, pages 3463–3467, 2013.

A1 Appendix

A1.1 Sample aeneas JSON Output

```
{
 "fragments": [
  {
   "begin": "0.000",
   "children": [],
   "end": "5.280",
   "id": "f000001",
   "language": "gle",
   "lines": [
    "C\u00farsa\u00ed sl\u00e1inte anois agus t\u00e1 an docht\u00fair Seamus \u00d3
     ↪ Beirn liom, eh, beo ar\u00eds san sti\u00faide\u00f3 anseo."
   ]
  },
  {
   "begin": "5.280",
   "children": [],
   "end": "6.320",
   "id": "f000002",
   "language": "gle",
   "lines": [
    "F\u00e1ilte romhat ar ais a dhocht\u00fair."
   ]
  }
 ]
}
```

A1.2 RTTM File Format

RTTM stands for *Rich Transcription Time Marked* files. They are space separated that contain meta-data and annotations of audio recording files. The format is as follows, adapted from a pyannote tutorial[1]:

[1] https://github.com/pyannote/pyannote-audio/tree/master/tutorials/data_preparation

```
SPEAKER {uri} 1 {start} {duration} <NA> <NA> {identifier} <NA> <NA>
```

- {uri} stands for "unique resource identifier" (think of it as the filename).
- {start} is the start time (elapsed time since the beginning of the file, in seconds) of the speech turn.
- {duration} is its duration (in seconds), and {identifier} is the unique speaker identifier.

A1.3 Python Notebooks

All of the models, algorithms, and evaluation procedures described throughout this project were implemented across a number of python notebooks. Google's Colab[2] platform was used as a development environment as it provides free GPU machine usage, which was enormously helpful in running experiments involving deep neural networks. A secondary benefit of using Colab notebooks is that they are openly accessible by default. Here are links to a variety of notebooks used throughout this project:

- **Forced Alignment Filtering Algorithm**:

  https://colab.research.google.com/drive/1IgcJJz1PsdClBN46qxeWw4Ao1xBWcTd-?usp=sharing

- **Speaker Diarization**:

  https://colab.research.google.com/drive/18YtH_buEFdhCzem7KRr-fvFxFF9i-Wlv?usp=sharing

- **Overlapped Speech Detection**:

  https://colab.research.google.com/drive/1R7MbJj1l45bYlpd8rT9S4O-VhbwMMv-N?usp=sharing

- **Guided Speaker Clustering**:

  https://colab.research.google.com/drive/1rBw12oN61bbwDJ4QQD9l0Qnr9wgwbfEb?usp=sharing

[2] https://colab.research.google.com/