ASSR: Automatic Stuttered Speech Recognition

Anshul Gupta (16305R001)
Kapil Aggarwal (16305R010)

November 27, 2017

Contents

1 Introduction **2**

2 Dataset **3**

3 Methodology and Implementation Details **4**
3.1 Data Processing 4
3.1.1 Data Pre-Processing 4
3.1.2 Feature Extraction 5
3.2 Classification 5
3.3 Audio Correction 5
3.3.1 Overlapping Segmentation 5
3.3.2 Re-assembling the segments 6
3.4 Speech-to-text 6

4 Experiments and Results **8**

5 Summary and Future work **9**

Chapter 1

Introduction

More than 70 million people worldwide are stutterers – that's one in every 100.

Past few years have seen an increase in popularity of personal voice assistants which can actually speed up the day to day tasks. But that is limited by the ability of the speaker. If the input to the voice recognition system is from a stutterer, it fails miserably with accuracy as low as 18% and as high as 73% as compared to a baseline of 92% for normal speaker [6]. So our project aims to correct the Speech-to-Text conversion for a stuttered speech.

The existing work [1] that has been done for this problem is just classification of a speech as a stuttered speech or a normal speech. The approach they use is pre-emphasis of audio and then extract audio features (most of the models used MFCC features) and training the classifier models (ANNs, HMMs, SVMs etc.) In one literature, they use RMS, standard mean, median frequency, peak to peak analysis (the difference between the highest and the lowest frequency peaks in a .wav file) and a specialized mean feature extractor as the features for training the ANN.

Chapter 2

Dataset

One of the challenges for this project was to get a time-aligned labelled dataset.

The dataset we are using is University College London Archive of Stuttered Speech (UCLASS) [2]. It contains recordings of monologues, reading and conversations of different speakers ranging from 7 years old to 20 years old. Unfortunately, most of the recordings do not have orthographic transcriptions and/or time aligned transcriptions. The database has audio files in 2 releases, release 1 and release 2. Release 1 has 16 files which have time-aligned transcriptions and release 2 has only 4 files which have time-aligned transcriptions. So we worked with the 16 files from release 1 to train our models.

The audio files are in .wav format having a sampling rate of 22050Hz and the corresponding time-aligned transcriptions are in CHILDES CHAT [4] format. The orthographic transcriptions are in plain text.

Also the orthographic transcriptions do not have the text for the corrected speech. It has the transcription for the stuttered speech. So we had to manually transcribe the the audio files.

Chapter 3

Methodology and Implementation Details

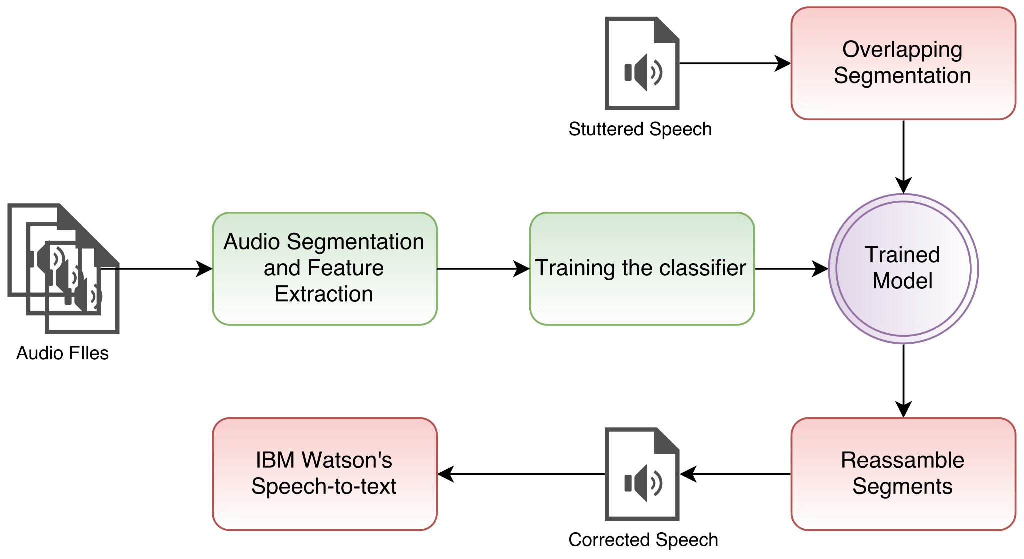

Figure 3.1: Flow Diagram

3.1 Data Processing

3.1.1 Data Pre-Processing

We used the time-aligned transcriptions to split the data files into stuttered segments and normal segments. The transcriptions had the start time and end time (in milliseconds) for each segment. So there were in all 12,633 segments that we got after splitting the audio files.

Table 3.1: Data Statistics

	ALL	STUTTER	NORMAL
COUNT	12633	2643	9990
MAX (ms)	17044	17044	14499
MIN (ms)	0	1	0
MEAN (ms)	315.0925	762.5323	196.7158
MEDIAN (ms)	192	486	168
MODE (ms)	109	201	93

From Table 3.1 we can see that the average duration of all the segments is $\sim$ 315 ms and stuttered segments are about 2.5 times longer than the average segment and the longest stutter was around 17 seconds. Training the data on such skewed data will not be useful because seeing a stuttered segment as long as 17 seconds is very unlikely.

So, instead of using the segments of variable duration, we segmented the files segments further down to less than or equal to 300 ms (figure 3.2) which is close to the average length of the segments. This segmentation created 17,545 segments which were used for training the models.

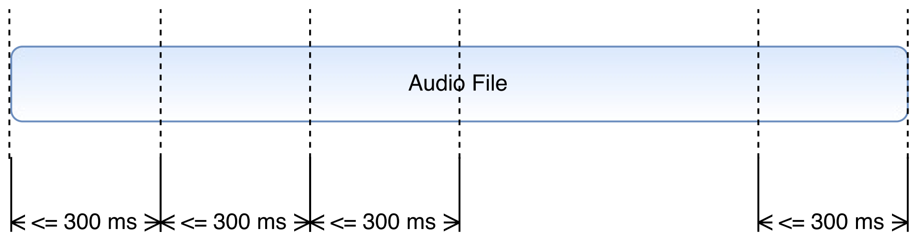

Figure 3.2: Audio segments

3.1.2 Feature Extraction

Once the audio segments were prepared, we then extracted the features from each segments which can be fed to the classification models.

We used MFCC features as they are very representative of human speech and RMSE which represents the loudness of a speech. We extracted 39 MFCC features and RMSE for each segment and stored the mean and variance of all 40 features (39 MFCC + 1 RMSE) to get a feature vector of 80 components.

In the end, our dataset had 17,545 rows each having 80 features.

3.2 Classification

Now with the dataset ready, next step was to train the classifier models. We used Deep Neural Network, Support Vector Classifier, Decision Trees, Gaussian Naïve Bayes, Bernoulli Naïve Bayes and Multinomial Naïve Bayes.

DNN model gave the highest accuracy (table 3.2) and took around ∼1 min to train as compared with SVC which took more than 1.5 hours.

The DNN model had 3 hidden layers, each having 10 neurons. Learning rate was 0.001 and training epochs were 1,200.

Table 3.2: Classification Accuracy of models

	Accuracy (%)
DNN	87.07%
SVC	85.43%
Decision Trees	76.63%
Gaussian Naïve Bayes	76.63%
Bernoulli Naïve Bayes	71.43%
Multinomial Naïve Bayes	71.43%

3.3 Audio Correction

With the classifier trained with an accuracy of ∼87%, next in the pipeline is audio correction.

3.3.1 Overlapping Segmentation

Our model is trained on audio segments of duration <= 300ms, it was only obvious that the audio to be corrected needs to be segmented with duration of 300ms. But what's less obvious was to detect the stutter boundaries. So instead of naïvely segmenting the audio in contiguous manner, we overlapped the segments as shown in figure 3.3. The overlapping was of 200ms.

With this type of segmentation, we could detect the stuttered and non-stuttered parts with the granularity of 100ms.

After the segmentation, these segments were sent to the classifier for classification.

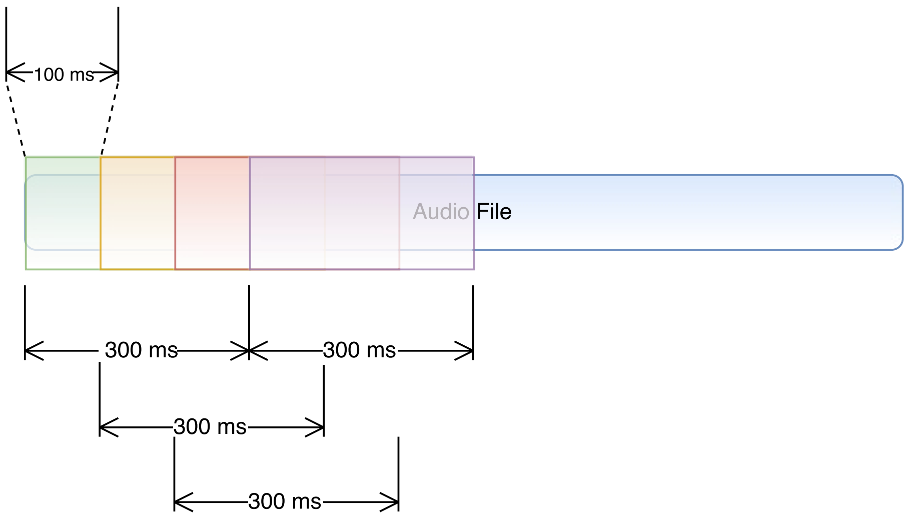

Figure 3.3: Overlapping Segmentation

3.3.2 Re-assembling the segments

The classifier gave the labels of the overlapping segments. So all we had to do was to remove the segments which were labelled as STUTTER and combine the segments labelled as NORMAL. We took a set difference to get the chunks of audio which contains only NORMAL speech.

One way of assembling the segments was to append contiguous chunks together as show in figure 3.4 but this will result in sharp interjections at the point of concatenation and will result in very artificial sounding voice.

So instead of naïvely appending the adjacent chunks, we interpolated the audio samples between the end of the previous chunk and the beginning of the current chunk as shown in figure 3.5.

The entire audio correction pipeline took 20 sec for an audio of duration 2 min 44 sec. And out of the 20 seconds, ~98% time was for feature extraction. Segmentation and classification were instantaneous.

3.4 Speech-to-text

The UCLASS dataset [2] is in British English. Instead of training our own Speech-to-text framework on British English, we used the IBM Watson's Speech-to-text [3] which already has a trained model for GB English.

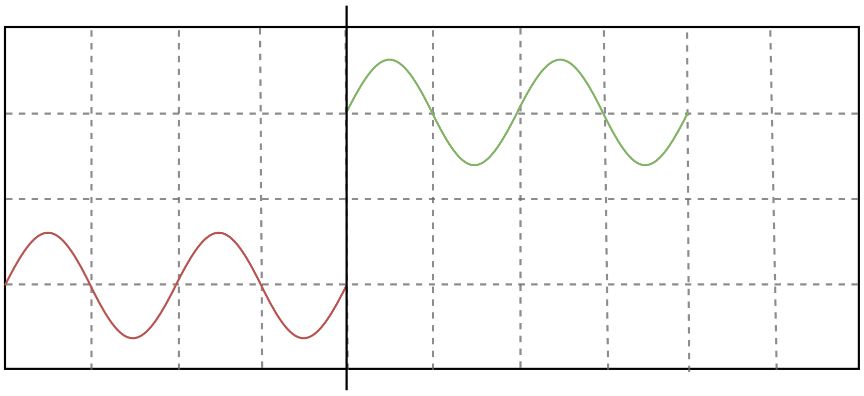

Figure 3.4: Naïve Re-assembling

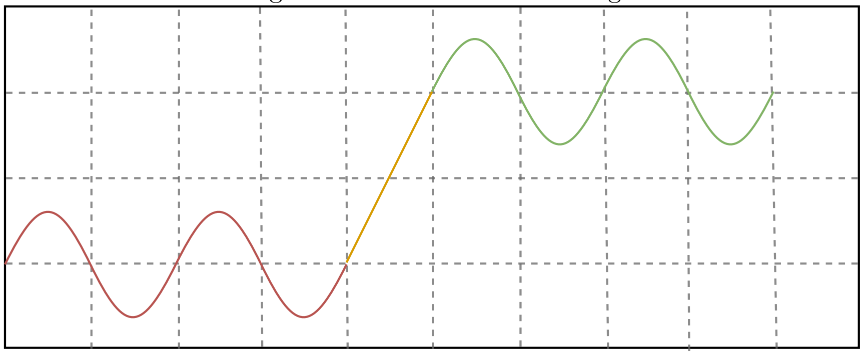

Figure 3.5: Smoothed Re-assembling

Chapter 4

Experiments and Results

As we can see from table 4.1, our model is far from perfect but we achieved some improvement over the un-modified audio files.

With the limited dataset available for training, we could only achieve this much accuracy.

There were cases where our corrected audio gave even worse WER (M_0100_12y3m_1) but we achieved significant improvement for M_1017_11y8m_1 and M_1017_13y2m_1.

Table 4.1: Comparison of WER of original audio and the corresponding corrected audio

Subject	**Original (%WER)**	**Corrected (%WER)**
M_0017_19y2m_1	74.928%	73.775%
M_0065_20y1m_1	125.000%	116.429%
M_0100_12y3m_1	84.173%	89.928%
M_1017_11y8m_1	55.396%	48.921%
M_1017_13y2m_1	59.322%	46.610%

Chapter 5

Summary and Future work

This study shows that the current state of art speech-to-text systems perform rather poorly with the speech of people having different ability for speech. With a little modification to the speech-to-text pipeline and a considerable amount of dataset, we can achieve a better result for predictions for almost 70 million people worldwide.

As mentioned in 3.3.2, around $\sim$98% time went in feature extraction, we could find some other feature extraction library (currently we are using librosa [5]) which can extract feature in less duration or even look at other features which are more representative of stuttered speech.

The source code of the project can be found at https://github.com/anshulgupta0803/stutter-speech-recoginition.

Bibliography

[1] Manu Chopra, Kevin Khieu, and Thomas Liu. Classification and recognition of stuttered speech.

[2] Peter Howell, Stephen Davis, and Jon Bartrip. The university college london archive of stuttered speech (uclass). *Journal of Speech, Language, and Hearing Research*, 52(2):556–569, 2009.

[3] IBM. Ibm watson speech to text. https://www.ibm.com/watson/services/speech-to-text.

[4] Brian MacWhinney. The childes project part 1: The chat transcription format. *Department of Psychology*, page 181, 2009.

[5] Brian McFee, Colin Raffel, Dawen Liang, Daniel PW Ellis, Matt McVicar, Eric Battenberg, and Oriol Nieto. librosa: Audio and music signal analysis in python. In *Proceedings of the 14th python in science conference*, pages 18–25, 2015.

[6] Emily Mullin. Why siri won't listen to millions of people with disabilities, May 2016. Article.