Digital Filter Speech Enhancement

Aidan Sharpe and Elise Heim

April 2024

Contents

1 Introduction

The purpose of speech enhancement is to improve the quality of degraded speech signals. Speech signals can be corrupted through different types of noise, interferences, echoes, and reverberations. Such degradations reduce sound quality to listeners. This can also deteriorate the performance of speech recognition systems. Speech enhancement techniques are necessary for almost all kinds of speech communication systems [1].

2 List of Objectives

1. Add various noises to various speech files with various SNRs
2. Evaluate the quality of sound files
3. Remove the noise from noisy speech files using a Weiner filter
4. Determine effectiveness of the Weiner filter

3 Background on Speech Enhancement

Spectral Subtractive Methods for filtering speech are based on the principle that by using an estimate of the noise spectrum, an estimate of the clean speech spectrum can be attained by subtracting the noise estimate from the noisy speech spectrum. These methods are based on the assumption that one can obtain an estimate of clean speech spectrum by subtracting the estimated noise spectrum from the observations spectrum. Despite being intuitive and computationally simple, these methods cannot make any claim of being perfectly optimal. In contrast to this, in Weiner Filtering Based Methods of filtering the estimated speech spectrum is obtained by estimating the power of the SNR (Signal to Noise Ratio) in an effort to minimize mean square error [1].

4 Adding Noise to the Speech

The speech dataset consisted of 30 IEEE sentences from 3 male and 3 female speakers These initial speech data were clean (without noise). The dataset also included four noise samples. Both the speech data and the noise data were sampled at 8[kHz].

The first step to adding noise was to load all files into memory. The speech files, which were encoded as Waveform Audio (.wav) files, could be loaded using `scipy.io.wavfile.read(path)`. This function returns the sound data in a `numpy.ndarray`, making calculations very easy. To load the noise, the files were read as a text file, and each line was typecast to a float. These floats were also put in a `numpy.ndarray`.

With the audio data loaded in this way, adding the noise to the signal at the desired SNR was quite simple. First, a random segment from the noise was chosen such that the number of samples in the noise segment matched the number of samples in the speech array. Next, the power of the noise segment and the speech were calculated. This was done by taking the L2 norm of each array. With the power of each signal in mind, the amplitude of the noise was scaled to the desired SNR. Finally, the noisy signal was produced by the vector addition of the scaled noise segment and the initial speech signal.

Noise was added at SNRs of 0[dB], 10[dB], 20[dB], and 30[dB]. At 0[dB], the speech becomes very hard to parse, because the power of the noise is the same as the power of the speech. At 30[dB], however, the noise is still noticeably there, but it its far less disruptive.

5 Perceptual Evaluation of Speech Quality (PESQ)

PESQ is a standardized and popular method for measuring the perceived speech quality of an audio signal. Importantly, it measures, or more correctly, attempts to estimate, how easy it is to understand speech in an audio signal. It is measured on a scale from -0.5 to 4.5. A PESQ under 2.8 is considered difficult to understand.

The metric used in this experiment is a full-reference (FR) PESQ. Whereas a no reference (NR) PESQ measurement only has access to a noisy signal, a FR PESQ has access to the original, clean signal. The FR PESQ yields much higher accuracy than the NR PESQ.

To calculate the PESQ of the signal with Python, a helper function was installed. The function, `pesq()`, used the corrupted and the original signals, and their sample rates to calculate the FR PESQ of the signal. This function provided a "narrow band" mode for all signals, and a "wide band" mode for signals sampled at over 8[kHz]. Since our data was sampled at 8[kHz], only the narrow band PESQ was calculated.

6 Wiener Filter

To enhance the noisy speech data, a Wiener Filter was applied by using the `scipy.signal.wiener(im)` function. After applying the filter to noisy speech, the noise and the speech become more muffled. The speech is a little more clear, but overall, the sound quality was not great.

As seen in figure 1, when increasing the SNR, the PESQ also increased. This makes sense, because as the SNR increases, the amount of noise in the signal decreases. However, the filter does not appear to be helping the PESQ. At the highest SNR, the PESQ actually decreased slightly after filtering.

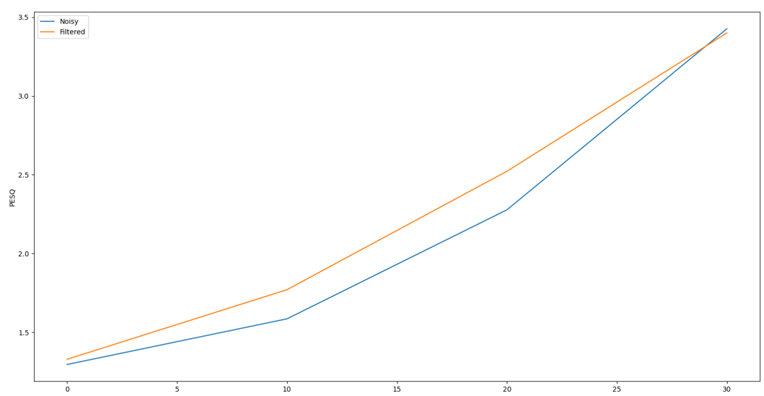

Figure 1: PESQ of noisy and filtered speech at different SNRs

7 95% Confidence Interval of PESQ

While the PESQ may have improved slightly when applying the Wiener filter to `sp01.wav` with white noise, only having one sample at each SNR does not provide much information about the efficacy of the filter. To make such claims requires an array of PESQs for different added noise and different speech clips. To accommodate this need, 30 speech clips, each with four different added noises, were examined. A 95% confidence interval for the PESQ was created for the noisy signal as well as the filtered signal in each case to determine if the filter made a statistically significant improvement.

As clearly seen in figure 2, there is overlap in the confidence intervals for SNRs of 0[dB], 10[dB], and 20[dB]. Therefore, we are 95% confident that at these SNRs, the Wiener filter used makes no change to the perceived speech quality. Even more surprisingly, at an SNR of 30[dB], we are 95% confident that the Wiener filter used makes the perceived speech quality worse.

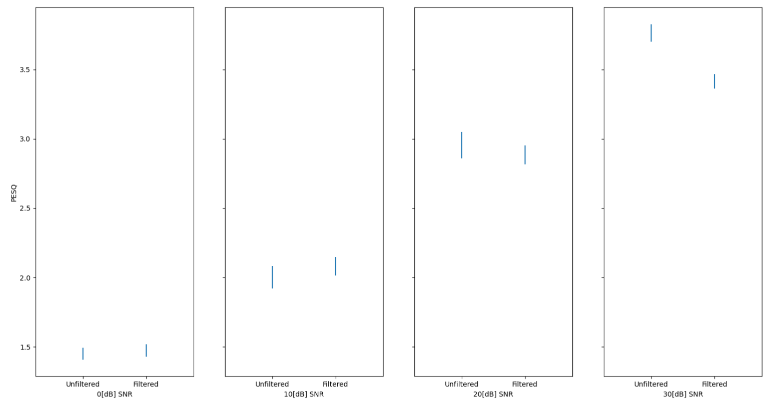

Figure 2: 95% confidence intervals for filtered and unfiltered PESQs at different SNRs

These results are unexpected, as the purpose of the filter is to improve the quality of the speech. The implementation of the Wiener filter in the SciPy library is quite different than the supplied method. Similar results were achieved by using the noise suppression algorithm implemented in [6], which uses *a priori* SNR estimation and cites Hu and Loizou as a reference, just like `wiener_as.m`. However, the implementation used in [6] used a frame size of 160 samples as opposed to the 20 samples per frame in `wiener_as.m`, and modifying [6] to fully support 8[kHz] signals was beyond the scope of this experiment.

8 Summary and Conclusions

Both the SciPy Wiener filter and the Python Speech Enhancement Library could not significantly improve the PESQ of the noisy speech data. While upon listening the filters reduced the noise level, the speech became muffled and difficult to understand at times. Since the PESQ was below 2.8 for SNRs of less than 30[dB], the inability to comprehend the speech is in line with the findings of [3].

Using Python's powerful built-in string manipulation tools made loading the list of filenames very easy. Additionally, Python's wide array of community-built libraries made finding signal processing functions a very straightforward process. For this reason, re-implementing `wiener_as.m` in Python may prove to be worthwhile future work.

9 Acknowledgement

The MATLAB code for the Wiener filter implementation and the PESQ calculation was taken from [5].

10 References

1. M. Parchami, W.-P. Zhu, B. Champagne, and E. Plourde, "Recent Developments in Speech Enhancement in the Short Time Fourier Transform Domain", IEEE Circuits and Systems Magazine, pp. 45—77, September 2016.
2. https://en.wikipedia.org/wiki/PESQ
3. https://www.spearline.com/blog/what-is-pesq/
4. https://www.mathsisfun.com/data/confidence-interval.html
5. P. C Loizou, Speech Enhancement: Theory and Practice, CRC Press, 2013.
6. https://github.com/chenwj1989/python-speech-enhancement/tree/master
7. https://github.com/ludlows/PESQ
8. https://en.wikipedia.org/wiki/Wiener_filter