Proposal for automation of Derivation of MOS for the Determination of PESQ/POLQA

Problem: Current voice quality PESQ/POLQA (Perceptual Evaluation of Speech Quality/Perceptual Objective Listening Quality Analysis, respectively) derivation is manual and done by hand and leads to changes of calculation of PESQ-related metrics due to subjective differences between different observers listening and measuring the audio quality differences between signals, as well as the possibility of background noise interference further affecting the results.

Proposal: automation of the testing process to assess PESQ, as well as automation of the calculation of the MOS (Mean Opinion Score) via machine learning to eliminate as much subjectivity as possible

Methodology:

There are two ways to evaluate the PESQ.

1) Full reference (FR): transmitted signal is compared directly to the original signal as a difference analysis, but can only be used for dedicated tests in live networks
2) No reference (NR): only the transmitted signal is analyzed with no other signal for reference, far lower accuracy than FR, and usually used as part of a transport-stream analysis

POLQA as a standard has been developed as a successor to PESQ because of its expansion to take into account more modern audio signal and networking technologies and infrastructure, as well as additional physical and systems modeling.

Google has also architected an open-source perceptual sound quality model, the Virtual Speech Quality Objective Listener (ViSQOL) as an alternative to the POLQA. ViSQOL is intended as a model of human sensitivity to degradations in speech quality via direct comparisons between a reference signal and a degraded signal.

POLQA Feature selection:

- Temporal alignment
- Frequency, reverberation, and noise distortion indicator calculations
- Corrections over level variation, frame repeats, spectral flatness, noise contrast, signal power variations
- Model is re-computed based on added distortions to the signal in order to do direct comparisons between original and reconstructed signals in order to detect/apply corrections.

ViSQOL Feature selection:

- Time alignment via segmentation of the sample and application of the Neurogram Similarity Index Measure (NMIS)

- Predicting Warp via NMIS comparisons between test and reference patches
- Similarity comparison using the structural similarity index (SSIM). Treat spectrograms as images and compare directly with the SSIM via pixel comparisons
    - Intensity -> luminance
    - Variance -> contrast
    - Cross-correlation -> structure
- Sigmoid mapping function to translate NSIM similarity score onto a MOS-LQOn score for scoring speech quality

Pre-processing of the audio signal will need to be done in accordance with the respective pre-processing guidelines outlined in both the ViSQOL and POLQA algorithms/standards. Usually this will involve scaling degraded signals to match the power levels of the reference signal, then applying FFTs across recommended windows. One current limitation is that only 1 minute of audio can be processed via the current testing setup at a time.

Examples of some features the speech quality algorithms utilize:

A. Time Domain
    a. Signal Average or Mean
    b. Signal Energy
    c. Signal Power
    d. Signal Variance
    e. Input to output duration ratio
B. Frequency Domain
    a. Zero-crossings rate
    b. 1st, 2nd, 3rd, 4th Spectral Moments
    c. even/odd harmonics ratio
    d. Signal clipping

Preprocessing steps:

- audio downsampling
- frame alignment
- time-warping detection
- sample/patch window determination
- FFT

Automating either VISQOL or POLQA/PESQ:

- POLQA implementations in python: https://chromium.googlesource.com/external/webrtc/+/HEAD/modules/audio_processing/test/py_quality_assessment/README.md
- VISQOL API: https://github.com/google/visqol
- Python can be done to script and automate audio generation, preprocessing, as well as the speech quality algorithm of choice in a modular fashion.
- Audio Quality Pipeline for Python: https://github.com/JackGeraghty/AQP
    - Contains both the WARP-Q and PESQ speech quality metrics

All testing will be done with prerecorded vocal samples scraped from the opensource and open database freesound.org. In order to emulate the degradation of audio quality due to a phone call, signal processing will have to be performed on the audio. Phone calls currently use the adaptive multi rate audio codec (AMR) as an audio compression format optimized for speech coding. The AMR speech codec consists of a multi-rate narrowband speech codec that encodes narrowband (200–3400 Hz) signals at variable bit rates ranging from 4.75 to 12.2 kbps with toll quality speech starting at 7.4 kbps.

All signal processing will be done with librosa (library for music and audio analysis), in addition to pyaudio and Torchaudio.

Sound files that have a sampling rate of 44khz will first be downloaded. They will be resampled from 44khz to 8khz using librosa.

In order to block out the frequency bands according to AMR, apply the short time fourier transform to bin the resampled audio into its respective frequencies, and then 0 out the frequencies mentioned in AMR (0-200, 3400+).

Inverse short time fourier transform will then recompile the audio back into a time series signal.

This will act as the control case. To emulate signal degradation in the audio, various additional notch filters will be added to block out more frequency bands, and the audio will be further downsampled in both its sample rate and bitrate. Further glitching might be emulated via a variable pointer that will rearrange the audio buffer at either a set or random frequency, if necessary.

Week 1: voice to text testing and setup

- Python speech to text libraries: speechrecognition, pyttsx3, pyaudio, deepgram, Pysptk, Torchaudio
- Process audio files to match various phone call quality states.
- Read audio and use each respective library to perform speech to text
- Save output to file
- Scripts to batch and automate the speech to text should be written as well
  - Need to set up proper directory structure to store all of the files.

Week 2: VISQOL algorithm setup and feature testing

- Compile and build VISQOL library.
- Preprocess audio for use in VISQOL's "speech mode"
  - Resample audio to 16khz.
  - Normalized via roots-mean-squared
  - Mixed to mono
  - Audio is split into 10 second segments via librosa and pyaudio
- Single scores are not meaningful. MOS tests must be repeated and performed multiple times over numerous audio samples.

- Test with a single audio file, comparison between clean signal and degraded audio signal

Week 3: Preparing VISQOL + voice to text for batching

- Use python text to speech to do the difference comparison between clean and degraded audio signals (and potentially calculate the entropy differential as an additional feature/metric)
- Preprocess all of the audio files according to the steps previous
- Test voice to text test batching over all of the samples
- See how long it would take to do VISQOL analysis looped over all of the audio files, then test with a small number of files to debug issues in the batching code

Week 4: tuning of both voice to text and VISQOL features

- Once looping and batching scripts are finished, start changing parameters in VISQOL as well as the voice to text libraries in order to see the changes in the outputs (sample differential and MOS score).
- By adjusting feature parameters, the algorithm can be better tuned to specific groups of samples, which may represent a more accurate quality index, depending on the range of quality of the call audio signal.

Week 5: Testing with phone call audio

- Record audio from phone call between two phones in makeshift faraday cage test boxes (boxes lined with foil). Manually transfer files for preprocessing
- Test with libraries and algorithm

Week 6: Audio input hardware

- Set up input cables to record audio from both phones into digital audio interface (focusrite scarlett)
    - Digital audio interface is connected to computer via usb
    - Audio splitter cable in the phone can redirect direct microphone audio into the audio interface, to be recorded in python via pyaudio and librosa
    - Same for receiving phone
- Test with libraries and algorithm, repeating previous steps

Week 7: full pipeline and testing

- Test to see if batching for preprocessing call audio can be done while a call is in progress/after a call has recently been completed

week 8: full pipeline and testing

https://developer.android.com/guide/topics/media/sharing-audio-input

Speech Recognition:

Create a new virtual environment.

```
sudo pip install SpeechRecognition

sudo apt-get install python-pyaudio python3-pyaudio

sudo apt-get install portaudio19-dev python-all-dev python3-all-dev
&&
sudo pip install pyaudio

pip install pyaudio
```

In the case of old or broken repositories, all variants of the required libraries are listed here.

Also needed are cffi, pydub, librosa, and numpy, in addition to

```
sudo apt-get install libsndfile1
sudo apt install ffmpeg
sudo apt-get install flac
```

All code can be run from the virtual environment once all libraries are installed.

VISQOL:

Required instructions and libraries are here: https://github.com/google/visqol

In case of issues with the bazel compiler, check environment variables or other common issues:

https://docs.bazel.build/versions/0.19.1/install-ubuntu.html

VISQOL will require a specific release of bazel, nothing newer:
https://github.com/bazelbuild/bazel/releases/tag/5.3.2

https://github.com/tensorflow/tensorflow/issues/54516

There may be errors “building wheel” when attempts are made to try to compile the python VISQOL libraries. Various fixes didnt work until a brand new virtual environment was tried, even then that is not guaranteed.

https://github.com/pypa/packaging-problems/issues/648

https://stackoverflow.com/questions/53204916/what-is-the-meaning-of-failed-building-wheel-for-x-in-pip-install

https://github.com/AcademySoftwareFoundation/rez/issues/1315

https://github.com/carla-simulator/carla/issues/5542

Regarding librosa changes:

https://github.com/nestyme/Subtitles-generator/issues/4

https://pysoundfile.readthedocs.io/en/0.8.1/#soundfile.write

https://github.com/bastibe/python-soundfile/issues/227