A guide to de-noising audio datasets

(Revised v3)

<u>Note: This tutorial is designed for Google Colab notebooks and uses Python language.</u>

Context

Pre-processing is an intermediate step in the Data Preparation phase of Machine Learning Lifecycle model. Large datasets are required for executing ML models. These datasets need to be cleaned and structured before it is fed into the training model. Datasets exist in various formats such as numerical databases, text documents, audios, images and videos. This article focuses on pre-processing audio datasets.

Objectives

This tutorial introduces you to audio datasets pre-processing methodology. You will learn to clean ambient noises in the audio background which reduces the audio clarity and extract the voice only from the distorted audio file and scale the process for cleaning large audio datasets.

Background

Types of ambient noises:

Stationary noise

Stationary noise refers to a steady stream of noise that is relatively unchanged in the entire audio recording. Imagine a continuous hum or steady hiss or the sound of a lost signal–a buzz sound–on a TV channel that gets recorded in the background when the user is speaking.

Non-Stationary noise

Non-Stationary noises refers to variable noises that change in frequency, intensity and pitch that occur in the background of the entire audio recording. Think of music playing inside the car or other human voices talking in the background at a party or a baby crying in the background.

Note: The audio pre-processing approach demonstrated in this guide is limited to stationary noise in an audio. You may not get the expected results for a non-stationary noise in the audio file.

Dataset

AVICAR Dataset is an open source dataset with audio recorded in a moving car. The user speaks alphabets or words or numbers with a Stationary noise recorded in the background of the audio. There are a total of 10023 audio files

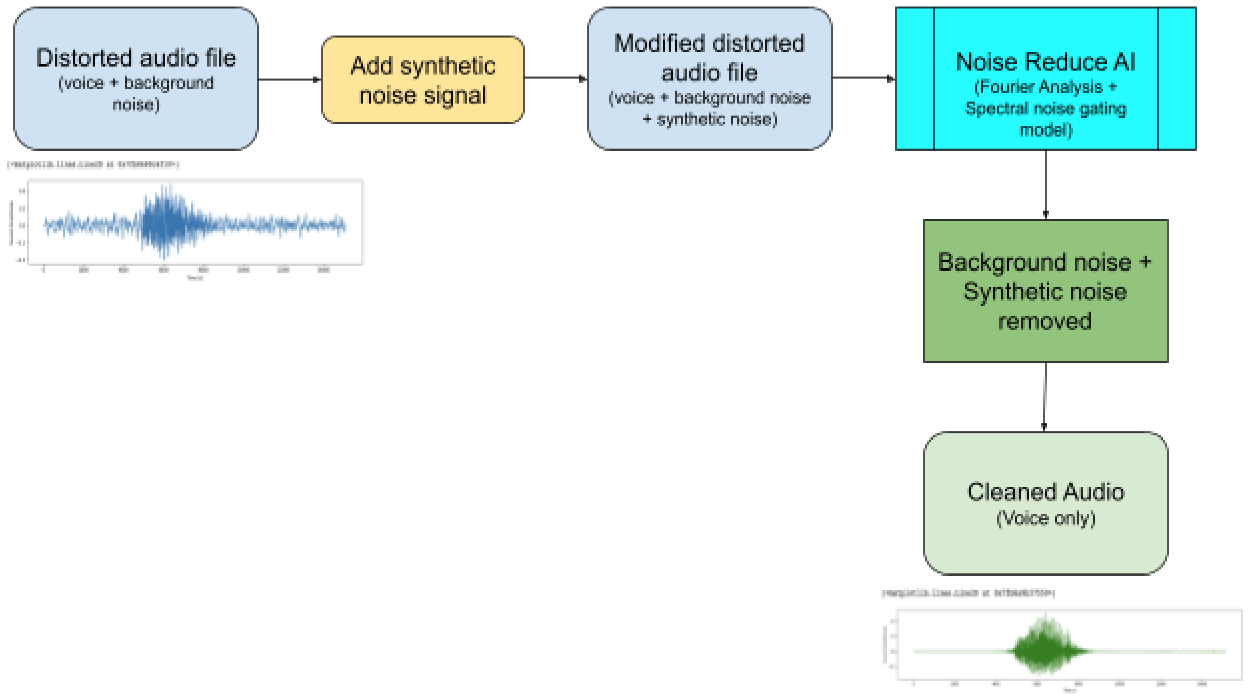
Methodology

The overall concept is to make the static raw noise file into a modified noise file that can be manipulated for CRUD(Create,Replace,Update,Delete) operations. And then, an AI model identifies and separates the noise and the voice using the frequency, intensity and pitch signals in the audio file. The result is a cleaned voice audio file.

Process

The raw audio file containing both the voice and background noise is layered with a synthetic noise signal which further distorts the signals in the audio. This modified file is fed into the Noise Reduce AI which extracts the voice separately to provide a clean audio output.

The following diagram outlines the workflow for cleaning distorted audio datasets:


Distorted audio file
(voice + background noise)
Add synthetic noise signal
Modified distorted audio file
(voice + background noise + synthetic noise)
Noise Reduce AI
(Fourier Analysis + Spectral noise gating model)
Background noise + Synthetic noise removed
Cleaned Audio
(Voice only)

Note: The curly blue lines in the distorted audio image is the noise signal and after processing those curly lines are converted to straight which indicates that the noise is removed.

Environment

You'll need the Jupyter Notebook version 6.4.5. You can find installation instructions on the Jupyter website. Alternatively, you can use Google Colab to run the notebook directly in your browser.

Pre-Requisites

The following python libraries should be installed

```
!pip install librosa   # python package for analysis of audio files
!pip install noisereduce # python package for cleaning audio background
!pip install soundfile # python package to read and write audio files
```

Dependencies

Libraries installation

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from glob import glob
from IPython.display import Audio
import librosa as lr
import librosa.display
import os
import io
import sys
from scipy.io import wavfile
import noisereduce as nr
import soundfile as sf
from noisereduce.generate_noise import band_limited_noise
```

Clean a sample Audio file

Step 1. Set directory path for loading AVICAR audio files

```
data_dir='/Downloads/avicar_somedigits'
audio_files=glob(data_dir + '/*.wav')
len(audio_files)
```

Note: This dataset is loaded from a local machine. Substitute the `data_dir` with your local path.

Step 2. Load the first audio file

```
data, rate = sf.read(audio_files[0])  # read the first audio file
np.shape(data)  # Array size is 15153
```

- `data` is the audio file that is converted into an array
- `rate` is the sampling rate

Step 3. Find Sampling rate

```
print(rate) # sampling rate is 16Khz
```

Step 4. Listen to the audio

```
Audio(data=audio_files[0], autoplay=True)
```

Step 5. Find duration of the audio

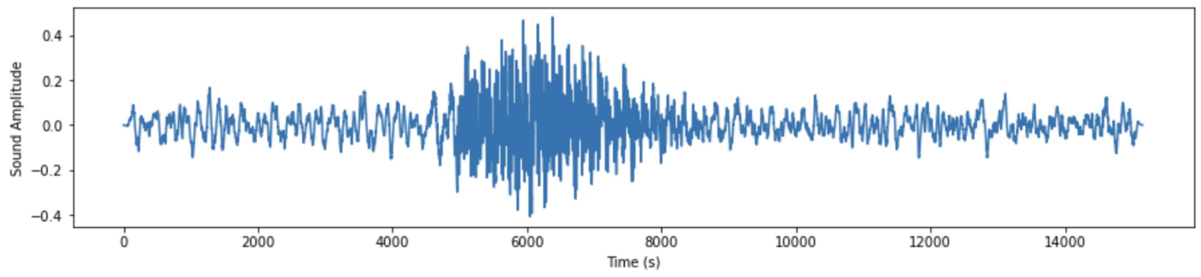
```
file_name=audio_files[0]
dur_avi=lr.get_duration(y=data, sr=rate)
print(dur_avi)  # Duration is .9470635 seconds
```

Step 6. Plot Amplitude over Time waveform

Graph is plotted using `matplotlib` library.

```
fig, ax = plt.subplots(figsize=(15,3))
ax.set(xlabel='Time (s)', ylabel='Sound Amplitude')
ax.plot(data)
```

```
: [<matplotlib.lines.Line2D at 0x7fb8a9e32130>]
```



Step 7. Dataset Inferences

The first audio file in the AVICAR dataset has a male voice speaking the word "Done" in a moving car environment.

The background noise of the moving car (humming noise) is captured along with the human voice during audio recording. This background noise is inferred as a stationary noise and it should be removed.

Duration of the first audio file = 0.9470625 seconds

The spectrum graph shows distorted amplitude wave

Spectrum shows wavy spikes throughout the length of the audio.
These wavy spikes imply background noise of the moving car.

Stationary Noise Cleaning using `noisereduce` AI

`Noisereduce` AI is a combination of Fourier Analysis algorithm and Spectral Noise Gating. The author of this algorithm and the related Github repo can be found here.

The `noisereduce` AI requires two inputs:

1. A noise audio clip containing Stationary noise.

2. A signal audio clip containing the signal and the noise intended to be removed.

Step 1. Creating the synthetic noise signal for the Audio

This code creates a synthetic noise signal and adds it to the original audio.

The length of the audio `noise_len` is standardized to 1 sec.

The frequency of the synthetic signal `noise` is set in the range 10KHz - 20Khz

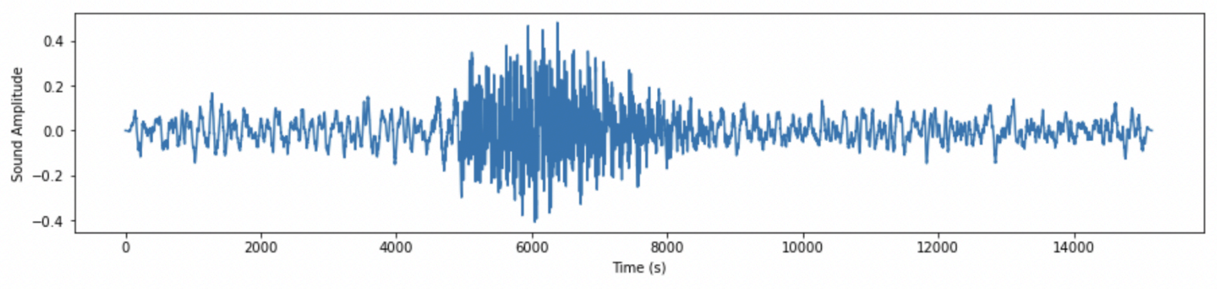
The audio file array and the noise array are combined together and stored in variable `audio_clip_band_limited`. This is a simple mathematical matrix addition technique.

```
noise_len = 1 # seconds
noise = band_limited_noise(min_freq=10000, max_freq = 20000,
samples=len(data), samplerate=rate)*10
noise_clip = noise[:rate*noise_len]
audio_clip_band_limited = data+noise
```

Step 2: Plot Graph for Modified Audio with Synthetic signal

```
fig, ax = plt.subplots(figsize=(15,3))
ax.set(xlabel='Time (s)', ylabel='Sound Amplitude')
ax.plot(audio_clip_band_limited)
```

```
[<matplotlib.lines.Line2D at 0x7fb8680c6f10>]
```



Step 3: Removing noise

This code block removes the synthetic signal that was added in Step 1.

The frequencies for both synthetic signal and background audio signal overlap each other. And, when the `noisereduce` AI is applied on the audio, the background noise is removed along with the synthetic signal that was added previously.

```
reduced_noise = nr.reduce_noise(y =
audio_clip_band_limited,n_std_thresh_stationary=1.5,sr=rate,stationary=True
)
```

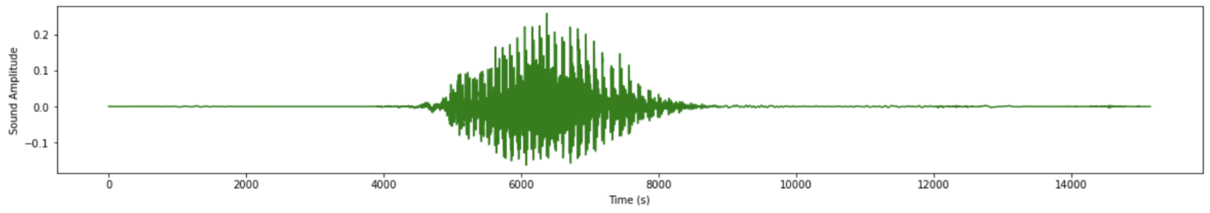
Step 4: Listen to the Cleaned audio

```
Audio(data=reduced_noise, rate=rate, autoplay=True)
```

Step 5: Plot Graph for the Cleaned Audio

```
fig, ax = plt.subplots(figsize=(20,3))
ax.set(xlabel='Time (s)', ylabel='Sound Amplitude')
ax.plot(reduced_noise, color ="green")
```

```
[<matplotlib.lines.Line2D at 0x7fb8a9b37550>]
```



The straight line implies silence in audio. The waveform is a clean audio without the background noise which represents the male voice uttering the word "Done"

Removing Stationary noise for Deep Learning Model Training

Machine Learning models require large datasets for training a model to achieve higher prediction scores (accuracy metrics) for that model. Similarly, for Deep Learning models, a larger dataset with small size is required for model training to be successful and to have higher accuracy metrics. AVICAR is designed to be used for these types of use cases.

In the previous section, a sample cleaning with a single audio file was tested. In this section, the entire dataset is cleaned using the steps discussed in the previous section. This cleaned dataset can be used for training Deep Learning models.

Steps for cleaning the AVICAR dataset

1. Create a function for the adding synthetic noise to the audio

```
S_noise_reducer(data,length,rate)
```

2. Iterate the dataset inside a loop and clean each audio.

Step 1: Create a function `S_noise_reducer()`

This function adds a signal or synthetic noise to the audio.

The audio length is not standardized. The cleaned audio length remains the same as the original audio length.

```
def S_noise_reducer(data,length,rate):
   noise_len = int(length) # seconds
   noise = band_limited_noise(min_freq=10000, max_freq = 20000,
samples=len(data), samplerate=rate)*10
   noise_clip = noise[:rate*noise_len]
   audio_clip_band_limited = data+noise

   return(audio_clip_band_limited)
```

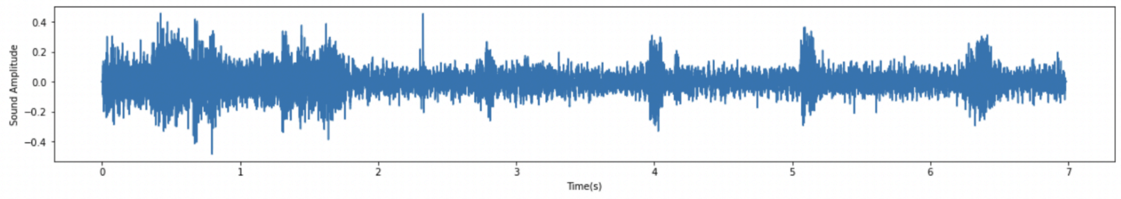
Step 2: Iterate dataset inside a loop

```
audio_dur=[]
for file in range(0, len(audio_files),1):
   file_name=os.path.basename(audio_files[file])
   print(file,file_name)

   #Read the first audio file
   audio, sfreq=lr.load(audio_files[file])
   time=np.arange(0,len(audio))/sfreq

   #find the audio duration
   duration_avi=lr.get_duration(y=audio, sr=sfreq)
   #print('Audio Duration or length:',duration_avi)
   audio_dur.append(duration_avi)

   #Plot audio over time( Original Audio)
   print('Original_',file_name)
   fig, ax=plt.subplots(figsize=(20,3))
   ax.plot(time,audio)
   ax.set(xlabel='Time(s)',ylabel='Sound Amplitude')
   plt.show()


   #Stationary noise removal
   #Step1: Add noise
   audio_clip_band_limited=S_noise_reducer(audio,duration_avi,sfreq)
   reduced_noise = nr.reduce_noise(y =
audio_clip_band_limited,n_std_thresh_stationary=1.5,sr=rate,stationary=True
)
```

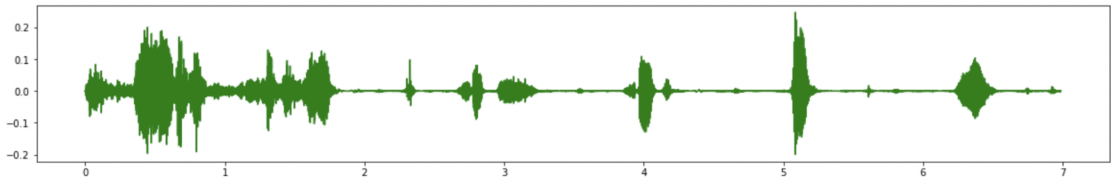
```
   #Plot Cleaned waveform
   print('Cleaned_',file_name)
   fig, ax=plt.subplots(figsize=(20,3))
   ax.plot(time,reduced_noise,color="green")
   plt.show()


print('********************************************************************
*********')

   #Save the Cleaned file
   sf.write('Cleaned_avicar_somedigits/Cleaned_'+file_name, reduced_noise,
sfreq)
```

Sample output

```
*****************************************************************************
8 GM2_55D_D9_C1_M3.wav
Original_ GM2_55D_D9_C1_M3.wav
```



```
Cleaned_ GM2_55D_D9_C1_M3.wav
```



```
*****************************************************************************
```

Future Work

This algorithm can be improvised using Speech-to-text API (Google's API) to convert the clean audio file to text.

References

1. AVICAR Dataset
   AVICAR Dataset - Stationary noise dataset (car driving noise + driver/passenger noise)

2. Noisereduce AI
   Link : timsainburg.com/noise-reduction-python.html
   Github repo: github.com/timsainb/noisereduce
   License: MIT license

Colab Notebook

Voice Assistant Stationary Noise Cleaning