Audio Noise Clustering

Dror Ayalon (dda290)

CUSP-GX-5006 Machine Learning for Cities (NYU)
Final Project

Abstract

Clustering is a major task in machine learning and many different clustering techniques are available. Very small academical effort was invested in finding the intersection between machine learning algorithms and digital signal processing (DSP) tasks. This research tried to utilize clustering algorithms, in particular spectral clustering and Independent Component Analysis, to reduce noise from speech centric audio recordings. The results were compared with a noise-reduced outcome, which was generated using existing post-production techniques, such as equalizing and frequency band-passes, that were implemented using Python. The noise-reduced audio outcome using machine learning algorithms was not as good as the audio outcome using post-production techniques, but led to some interesting conclusions and ideas for further research. All results could be heard on this url - https://dodiku.github.io/audio_noise_clustering/results .

1 Introduction

Voice interfaces become more common for day-to-day communication. Mobile devices users tend to use these interfaces on-the-go in very noisy environments. The main goal of this research was to analyze an audio recording and to find the model that could cluster the signal into two groups: ”speech” and ”noise”. Another motivation for this research was to find a computational efficient way to perform this complicated task in real-time applications, without the need for a long and costly deep-learning process.

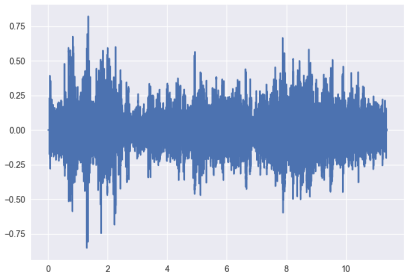

Figure 1: The audio recording represented as a time-series (amplitude over time).

2 Methods and Data Sets

1. Audio signal as a dataset - In order to run machine learning algorithms on the audio signal, there was a need to convert the digital audio time series (see figure 1) into a dataset the could be clustered. The audio signal was converted into two types of data structures:

   (a) Spectrogram - After running a short-time Fourier transform on the audio signal using Li-

bROSA [2], a Python audio analysis package, the signal could be represented as a spectrogram using a a matrix, which provides data about the intensity of each frequency range within a time frame (see figure 2). Each cell on the matrix is a data sample.

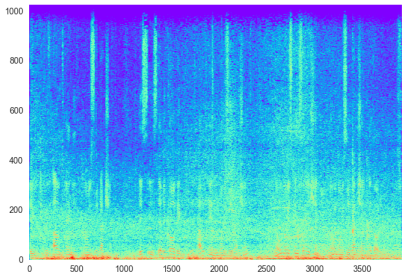

Figure 2: The audio recording represented as a spectrogram (frequency bins over time).

(b) 3D table - Another way to represent the audio signal data is using a 3-dimensional table (see figure 3), where each row is a single sample of the data. The data on each and every sample is structured as follows:

i. Column 1: A numeric value that represents the time-frame number of the sample.

ii. Column 2: A numeric value that represents the the frequency range of the sample.

iii. Column 3: The level of intensity of the sample (the frequency range in the time-frame).

Using these 2 different data structures as an input to the spectral clustering algorithm generated different results (more on that on the results section). Another motivation behind the usage of 3D table as a dataset structure was the ability to visualize the data using scatter plots, where each sample is a point on the plot. The visualization allowed a closer and cleared investigation of the clustering results (see example in figure 4).

0.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
1.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
2.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
3.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
4.0000000000000000e+00	0.0000000000000000e+00	1.0000000000000000e+00
5.0000000000000000e+00	0.0000000000000000e+00	-1.0000000000000000e+00
6.0000000000000000e+00	0.0000000000000000e+00	-2.0000000000000000e+00
7.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
8.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
9.0000000000000000e+00	0.0000000000000000e+00	0.0000000000000000e+00
1.0000000000000000e+01	0.0000000000000000e+00	0.0000000000000000e+00
1.1000000000000000e+01	0.0000000000000000e+00	0.0000000000000000e+00
1.2000000000000000e+01	0.0000000000000000e+00	0.0000000000000000e+00
1.3000000000000000e+01	0.0000000000000000e+00	0.0000000000000000e+00
1.4000000000000000e+01	0.0000000000000000e+00	0.0000000000000000e+00

Figure 3: The audio recording represented as a 3D table, where each row includes attributes on a specific sample.

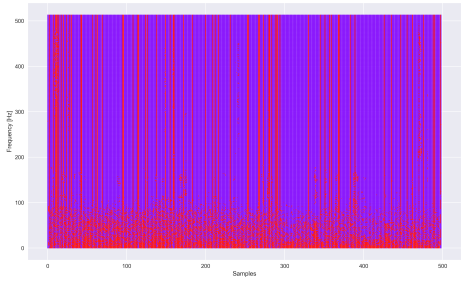

Figure 4: An example from one of the results of a spectral clustering on a 3D dataset, that represents a spectrogram of an audio signal.

2. The research was done using Python3 and the machine learning Python package scikit-learn (http://scikit-learn.org/). The following scikit-learn algorithms were used to the generate the results for this research:

- Spectral clustering (`sklearn.cluster.SpectralClustering`) [4] - To split noisy frequencies from speech frequencies, a variety of spectral clustering techniques were tested during this research.

The largest variation in the outcomes were generated using ”rbf” VS. ”nearest_neighbors” as methods to create the affinity matrix, and ”discretize” VS. ”kmeans” as methods to assign the clustering labels (more about that on the results section).
The spectral clustering algorithm was applied on both structures of the the dataset (spectrogram and 3D table) using 2 strategies:

(a) Complete approach: The entire dataset was used as an input for the algorithm.

(b) Column-by-column approach: Applying the spectral clustering algorithm on each and every time frame separately.

Each of these strategies generated very different results (more about that on the results section).

- Independent Component Analysis (`sklearn.decomposition.FastICA`) [1] [3] - An ICA algorithm was used as another method to split noise from speech (see figure 5). Since the ICA algorithm receives 2 inputs (in our case, audio signals) and generates 2 outputs based on the estimated source of the signal (speech and background noise), all possible combinations of the following audio signals were used as inputs:
  - The original recording
  - A noise reduced version of the original recording using post-production techniques
  - Audio signal that was clustered as ”noise” using the spectral clustering algorithm with the complete spectrogram matrix as a dataset
  - Audio signal that was clustered as ”speech” using the spectral clustering algorithm with the complete spectrogram matrix as a dataset
  - Audio signal that was clustered as ”noise” using the spectral clustering algorithm with the complete 3D table as a dataset
  - Audio signal that was clustered as ”speech” using the spectral clustering algorithm with the complete 3D table as a dataset
  - Audio signal that was clustered as ”noise” using the spectral clustering algorithm in a column-by-column approach on the complete spectrogram matrix as a dataset
  - Audio signal that was clustered as ”speech” using the spectral clustering algorithm in a column-by-column approach on the complete spectrogram matrix as a dataset
  - Audio signal that was clustered as ”noise” using the spectral clustering algorithm in a column-by-column approach on the complete 3D table as a dataset
  - Audio signal that was clustered as ”speech” using the spectral clustering algorithm in a column-by-column approach on the complete 3D table as a dataset

  It is important to mention that using different parameters on the spectral clustering algorithm, each of the methods specified above generated different clusters, and therefore, different signals. All these different outcomes where used as an input for the ICA algorithm. The best results are shown on the results section.

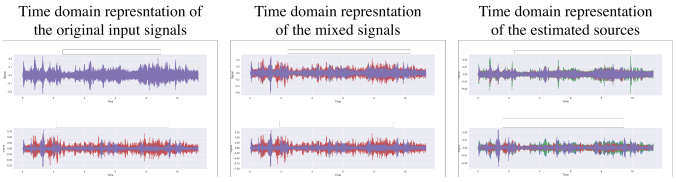

Figure 5: The process of convolving two audio signal, and splitting them again, based on their estimated sources, using ICA.

3. Data reduction process - To reduce the computation time, audio recordings were sampled in a regular sampling-rate and a low-resolution sampling-rate. The algorithmic procedures were done on the low-resolution datasets. Then, the low-resolution dataset was stretched to the size of the regular-resolution

dataset (see figure 6). The clustering results, which where generated on the the low-res dataset, were applied on the stretched regular sized dataset. This method enabled reasonable runtimes for the clustering procedures, and the ability to generate outcome wave files (which can only be generated using the high-res sampling dataset) that were used to evaluate the final results of each run.

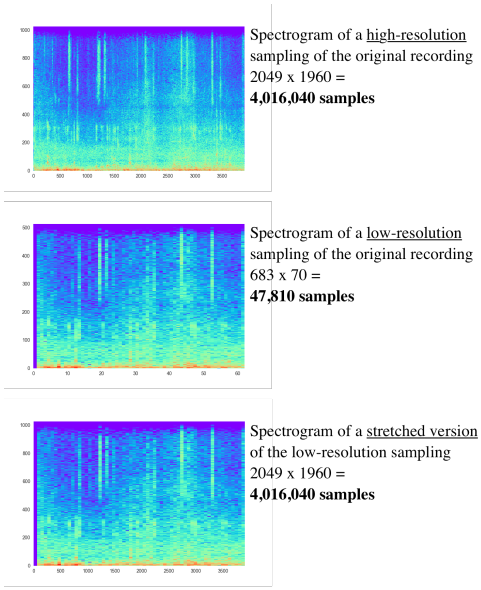

Figure 6: The process of stretching a low-res dataset (audio signal spectrogram) to the size of the regular dataset, without loosing the main features and structure of the dataset.

4. To summarize, the work process for this research was as follows:

   (a) Load audio file

   (b) Sample the audio file, using a low-res sample-rate and a regular-res sample-rate, to generate audio time series.

   (c) Convert the audio time series to a spectrogram of amplitudes (matrix, bins * frames) using a Short-time Fourier transform.

   (d) (optional) Change the structure of the data to improve clustering results

   (e) Cluster the data using the spectral clustering algorithm

   (f) Stretch the low-res dataset to the size of the regular-size dataset

   (g) Apply the clustering results on the stretched dataset

   (h) Remove (or reduce) the samples that were clustered as noise

   (i) Run Inverse short-time Fourier transform to convert the spectrogram back to an audio time series

   (j) Write a file using the new time series array- Compare results sonically

3 Results

1. All outcomes could be heard on the following url - https://dodiku.github.io/audio_noise_clustering/results/

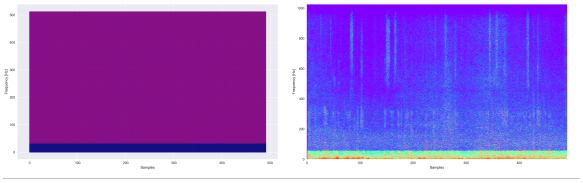

Figure 7: LEFT: Clustering results for the complete spectrogram as a dataset. RIGHT: Applying the clustering reults on the speech recording.

2. A very ”naive” result was generated using the spectrogram (matrix) as dataset, and clustering the data using the ”complete approach”, while ”nearest_neighbors” was set to create the affinity matrix (adjacency matrix) and ”arpack” as the eigen solver (see figure 7 or listen to the best results under ”spectral clustering 01 - Spectrogram” on the results webpage).
   In terms of computation time, this result was generated relatively quickly (Runtime = 4.4s for a 10s recording), but was not satisfying enough sonically.

3. Very interesting results were generated using the spectrogram (matrix) as dataset in a ”column-by-column” approach, and while ”rbf” was set as the method to create the affinity matrix. The clustering algorithm was able to pick-up the dominant parts of the recording (see figure 8), but the outcomes have many artifacts and the isolated speech is not clear enough (listen to the best results under ”spectral clustering 02 - Column-by-Column” on the results webpage).
   The main drawback of this result was the long runtime (24.2s-26.9s on a low-res dataset, 1:30h on a regular-res dataset for a 10s recording).

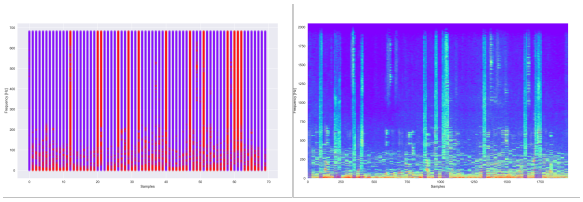

Figure 8: LEFT: Clustering results for the complete spectrogram as a dataset in a ”column-by-column” approach. RIGHT: Applying the clustering reults on the speech recording.

4. The best result from the spectral algorithm alone, was generated when the input was the 3D table as dataset using a ”column-by-column” approach (see figure 9, or listen under ”spectral clustering 04 - Column-by-Column from a 3D table” on the results webpage).
   This result was the best sonically, but was very heavy in terms of computation time (Runtime = 34.4s for low-res dataset, and 2.5h for a regular-res dataset for a 10s recording).

5. Two very interesting results were generated using the ICA algorithm. On one of these cases, the inputs to the ICA algorithm were:

   (a) The speech cluster outcome from spectral clustering using column-by-column approach on a 3D table.

   (b) The speech cluster outcome from spectral clustering using complete spectrogram.

   On the other case, the inputs to the ICA algorithm were:

   (a) The speech cluster outcome from spectral clustering using column-by-column approach on a 3D table.

   (b) ICA result using:

      i. The speech cluster outcome from spectral clustering using column-by-column approach on a 3D table and ”kmeans” as to assign the clustering labels.

      ii. A noise reduced version of the original recording using post-production techniques.

   The last described result is considered to be the best result that was found on this research (listen to these results under ”ICA: Independent component analysis” on the results webpage).

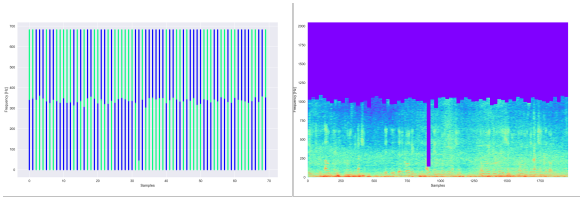

Figure 9: LEFT: Clustering results for the 3D table as a dataset in a ”column-by-column” approach. RIGHT: Applying the clustering reults on the speech recording.

4 Conclusions

1. Since human speech varies between a large range of frequencies, that in most cases overlap with background noise frequencies, it is very hard (maybe impossible) to cluster away noise.

2. The tested clustering technique could not produce better results than existing production techniques.

3. The data reduction and stretch technique was very successful in reducing the computation time of the clustering process, while not harming the outcomes of the process.

4. Using results from spectral clustering using the ”complete approach” and the ”column-by-column” approach as inputs to and ICA algorithm could produce promising results.

5. This research did not show promising reults using the noise frequencies, as clustered by the spectral clustering algorithm. This might be due to the similarity of the noise on the ”speech” cluseter and on the ”noise” cluster.

6. Using the entire spectrogram as an input to the spectral algorithm tend to generate ”naive” clusters. This behaviour makes sense since the distance between two frequency bins on different time frame (different column) is an important factor that effects algorithm”s affinity matrix.

References

[1] A. Hyvrinen and E. Oja. Independent component analysis: Algorithms and applications. *Neural Networks*, 13((4-5)):411–430, 2000.

[2] B. McFee, C. Raffel, D. Liang, D. P.W. Ellis, M. McVicar, E. Battenberg, and O. Nieto. Librosa: Audio and music signal analysis in python. *Python in Science Conf. (SCIPY)*, 2015.

[3] scikit-learn developers. scikit-learn: Fast ica. *http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.FastICA.html*, Open Source, BSD License.

[4] scikit-learn developers. scikit-learn: Spectral clustering. *http://scikit-learn.org/stable/modules/generated/sklearn.cluster.SpectralClustering.html*, Open Source, BSD License.