An analysis of dimensionality reduction and music information retrieval techniques for the visual representation of large audio datasets.

Łukasz Piotrak

February 8, 2021

Contents

Abstract

Abstract goes here

1 Introduction

1.1 Background & Motivation

The current paradigm for the storage and organization of audio files is that of the classic directory tree. This results in an attribute ontology; files are grouped together into classes (usually by directory name) i.e. “Drums”, “Vocals” and assigned a range of attributes by means of file name or tags. This approach is limited, e.g. the file might be labeled incorrectly or the labels might not addequately describe the sound. Moreover, many properties inherrent to the files are hard to represent e.g. two audio samples might be in separate branches of the taxonomy but be perceptually similar. However, as shown by projects such as The Infinite Drum Machine by Google Creative Lab, collections of sounds can be explored more naturally with the help of dimensionality reduction techniques.

Advances in this area have enabled the intuitive representation of high-dimensional data. The dimension count of a Dataset can be reduced arbitrarily while still preserving information about its intrinsic properties. Commonly, this approach is used to plot Datapoints as clouds in 2d or 3d space,

allowing for a depiction of the data which can be naturally grasped by the human mind.

As with many kinds of information, audio data in its raw form is unfit for such processing. An intermediate representation must to be constructed if any insights are to be gleaned from the data. In order to obtain a representations of audio signals useful to a human observer, a selection of features have to be extracted, which might correspond to certain aspects of the human perception of sound. These can then be used as inputs to produce a visualization by means of dimensionality reduction.

The goal of this paper is to provide an evaluation of both feature extraction and dimensionality reduction techniques, apply them to a large set of audio data and determine which combination of the two gives an optimal visual representaion.

1.2 Related Work

1.2.1 The infinite drum machine

1.2.2 Klustr

1.2.3 Other Commercial products

2 Background

2.1 Machine learning and audio

2.2 Music information retreival

2.2.1 Signal processing

1. The fourier transform
2. Sampling and the dirac delta function
3. The fast fourier transform
4. Signal processing and limitations

2.2.2 Pre processing steps

1. Down mixing
2. Normalization

2.2.3 Short-term Fourier Transform (stft)

The Short-term Fourier Transform is a representation of a signal obtained by taking the Fourier transform of short, overlapping segments of time. The method used to obtain the stft is relatively straightforward: First, the signal is divided into shorter, equal-length segments over time using a window function. Multiple window functions could be used for this step, however a

Next, the DFT is computed for each segment. We can thus observe how the frequency content of the signal changes as we progress through time.

2.2.4 MFCC

2.2.5 MIR metrics

1. spectral centroid
2. spectral rollof
3. spectral bandwidth
4. spectral crest
5. specral flux
6. spectral flatness
7. RMS
8. zero-crossing rate

2.3 Dimensionality reduction

2.3.1 PCA

2.3.2 t-sne

2.3.3 umap

2.4 Evaluation metrics

2.4.1 Silhoutte score

The Silhouette Coefficient for a sample i is given by the equation

$$s(i) = \frac{a(i) - b(i)}{\max a(i), b(i)}$$

Where a is the mean distance between a sample $i \in C_i$ and all other points in the same cluster C_i. This gives us how close a point is to the corresponding cluster. It is given by the equation:

$$a(i) = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j)$$

where $d(i, j)$ is the distance between points i and j.

b is the mean distance between a sample $i \in C_i$ and all other points in the next nearest cluster C_k. Given by:

$$b(i) = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j)$$

The final Silhouette score for a clustering is the mean Silhouette Coefficient over all the datapoints:

$$\frac{1}{|I|} \sum_{i \in I} s(i)$$

Where I is the set of all datapoints. The final value is in the range $[-1, 1]$ with values closer to -1 indicating incorrect clustering and values closer to $+1$ indicating highly dense clustering. Scores around zero indicate overlapping clusters. The score is generally higher for convex well-sperated and dense clusters.

2.4.2 Roundness

The overall roundness of the plot is calculated by using the method proposed by Polsby & Popper [**?**]:

$$PP(D) = \frac{4\pi A(D)}{P(D)^2}$$

where D is the convex hull of all points of the plot, $P(D)$ is the circumference and $A(D)$ the area.

2.4.3 Overlap of cluster convex hulls

The measure of overlap of the convex hulls of each class. I calculate this by taking the ratio of the area of the unary union of convex hulls of each class to the sum of areas of the convex hulls of each class:

$$O = \frac{A(U)}{\sum_{c \in C} A(H_c)}$$

Where $A(U)$ is the unary union of all convex hulls, $A(H_c)$ is the area of the Hull for class c and C is the set of all classes.

2.4.4 Ripleys K function

Ripley's K function is sum of the number of points N within a distance r of a selected point p, per area λ surrounding p. This value is normalized by the total points:

$$K(r) = \frac{\sum_{i=1}^{n} N_{p_i}(r)}{n\lambda}$$

It may be interpreted as a meassure of deviation of a given distribution from the random Poisson distribution. In essence, this let's us measure the homogeneity of the spatial density of the data points. The expected value of $K(r)$ for a random distribution is πr^2. If the output value deviates from this value, this indicates clustering or dispersion in the data. The K-function may be normalized so that the expected value is r:

$$L(r) = \sqrt{K(r)/\pi}$$

Further normalization gives an expected value of 0, called the H-function:

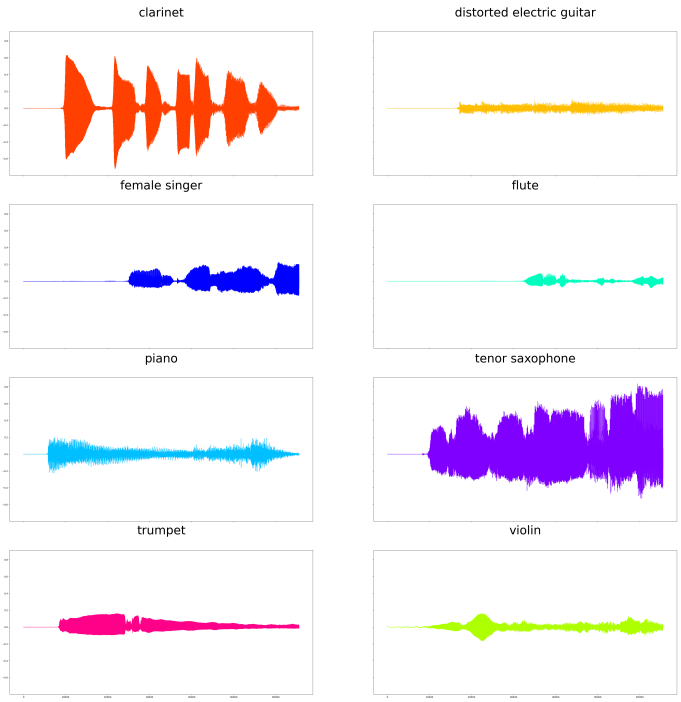
$$H(r) = L(r) - r$$

Now, a positive value of $H(r)$ indicates that the data is clustered at the scale of r. If the value is negative, the data is dispersed.

3 Experiment design and overview

3.1 Dataset

I used the Medley-Solos-db dataset assembled by Lostanlen et al. [**?**, **?**]. Downloaded through the mirdata python library [**?**]. The dataset consists of 21572 mono WAV files sampled at 44.1 kHz at a bit depth of 32. Every audio clip has a duration of 2972 milliseconds. The data is split into 3 subsets: training, validation and test. Each sample belongs to one instrument category among a taxonomy of 8. Each instrument class was given a distinct color for easier recognition on the plots:



Selected sample belonging to each of the instrument classes. Each consists of 65,536 32-bit floating point numbers.

3.2 The processing pipeline

To produce a 2d scatter plot of the dataset, the original audio files, each an array of 65,536 floating-point numbers, has to be reduced to 2 values. The process can be thought of as having 4 distinct steps:

1. Preprocessing.

   The audio files are ingested in a format which is easy to run calculations on. In this case a numpy ndarray.

2. Feature Extraction.

   Some characteristics of the sound are extracted using a selection of algorithms and mathematical tools. These are then used as an intermediate representation of the sound for further processing.

3. Feature manipulation

   Some of the features had to be further modified after extraction. The operations included reshaping feature matrices and aggregation.

4. Dimensionality Reduction.

   After selecting a set of features to serve as a representation of the original files an algorithm is applied to reduce them to two dimensions.

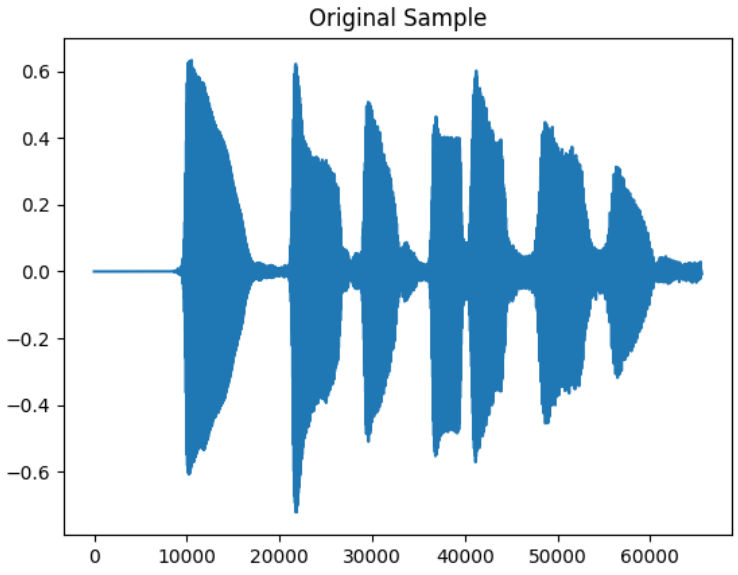
Once the plots have been generated, the one which most closely fits the defined criteria must be chosen. As such, an extra, fourth step in which plots are evaluated must be added. Each of these steps will be described in greater detail in the next section.

3.3 Preprocessing

The data is ingested using the Librosa python library [**?**] used for music and audio analysis. The "librosa.load" method was used to convert the WAV files to a float32 numpy ndarray. After loading the samples, the amplitudes were normalized to be in the range $[-1, 1]$ by dividing by the max amplitude value for the sample. The data loaded in such a way was stored in a hdf5 file.

3.4 Feature Extraction

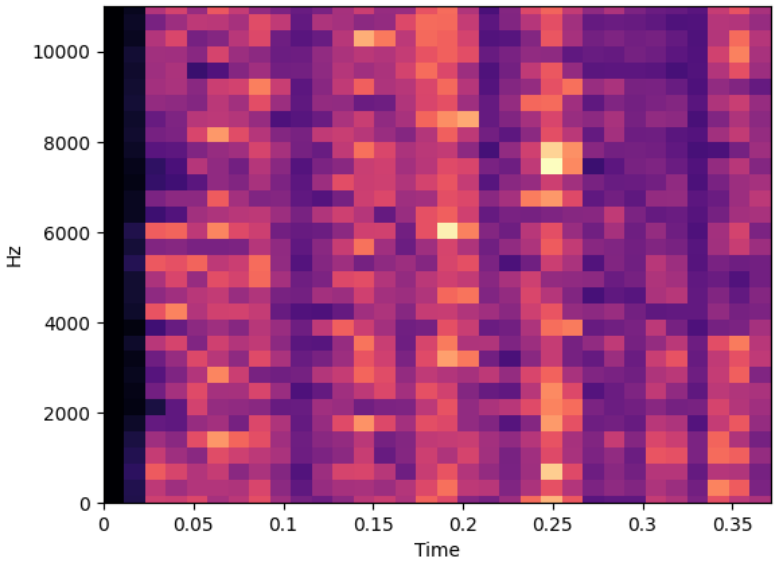
Finding a compact representation of phenomena is crucial for machine learning processess, including dimensionality reduction. To produce a meaningful representation of the raw data, useful to machines as well as humans, the step of extracting features is required. It can be even thought of as a preliminary dimensionality reduction technique as a raw signal consisting of many thousands of values to just a handful, which, with luck, provide an adequate representation of useful characteristics, innate to the signal. Most of the feature extraction steps were calculated using the implementations found in the librosa library.



An example clarinet sample.

3.4.1 STFT

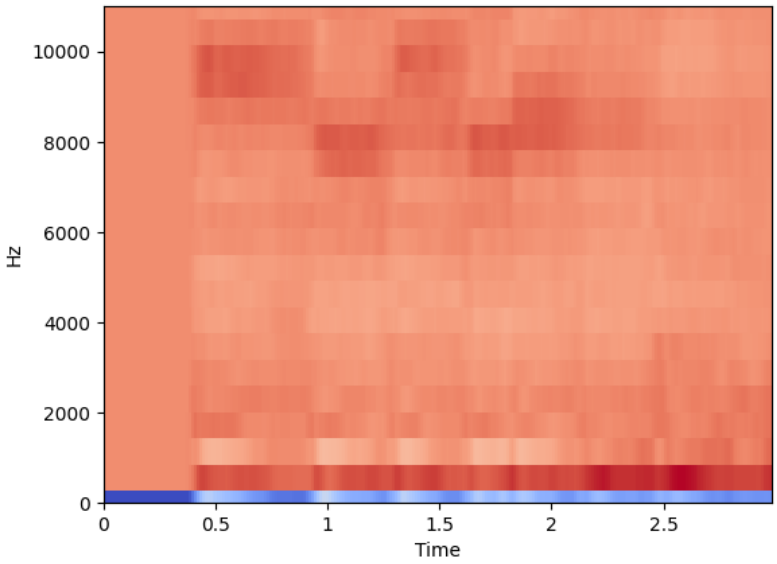
The Short-time Fourer Transform is a basic representation in signal processing, which captures the change in frequency content over time. To extract the stft, I used the librosa implentation. I decided to take the STFT over 32 windows in both the time and frequency domains, finally giving a 32x32 matrix:



STFT of the sample in figure ...

3.4.2 MFCC

The Mel-Frequency Cepstral Coefficients are a heavily used in both speech recognition and MIR. [**?**, **?**, **?**, **?**] I used the librosa implementation to calculate the mfcc's. I decided to go with a Cepstral Coefficient count of 20 and hop length of 256, resulting in a feature size of 20x257:



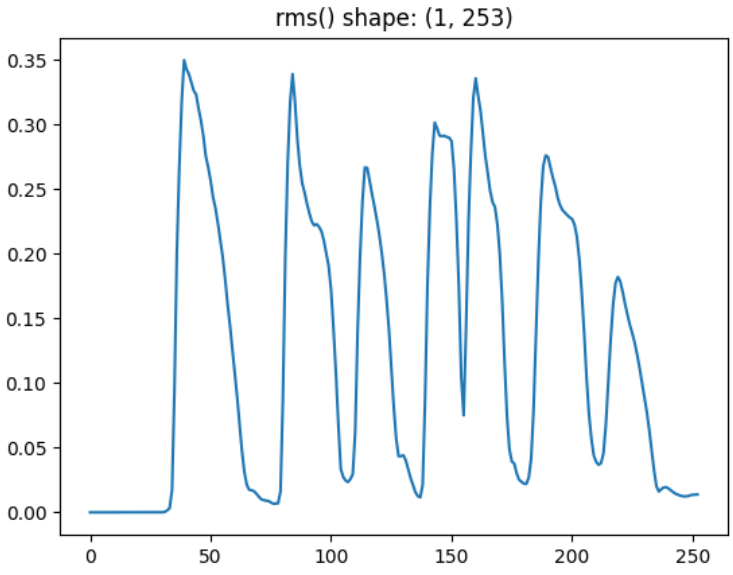
MFCC

of the sample in figure ...

3.4.3 MIR Metrics

A number of metrics from the field of audio analysis has also found to be useful when extracting timbre information from audio signals [**?**, **?**]. In my case these will include:

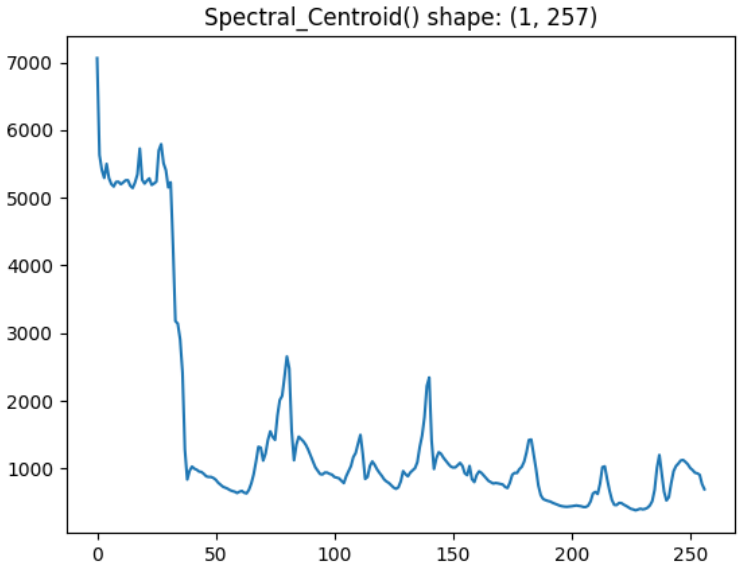
- Root mean square



RMS

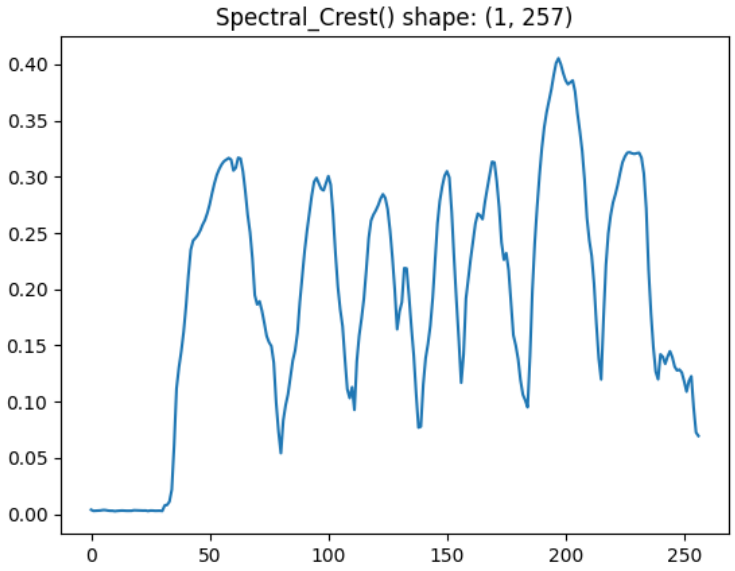
of the sample in figure ...

- Spectral Centroid



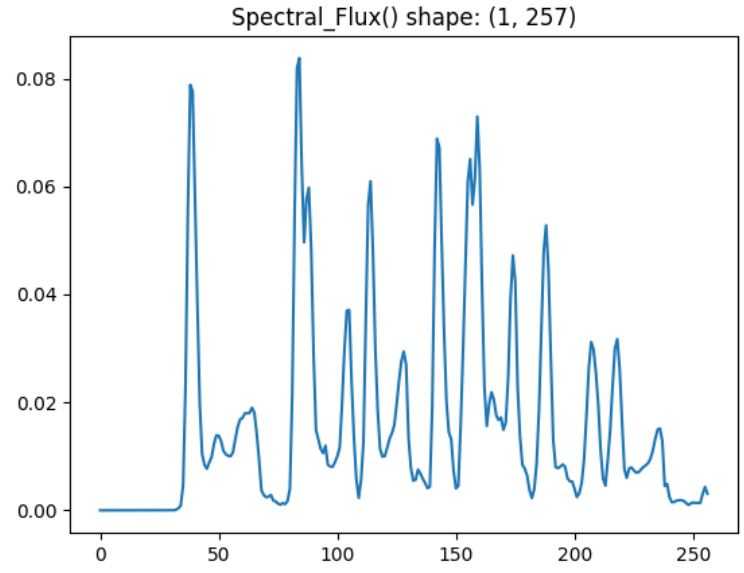
Spectral Centroid of the sample in figure ...

- Spectral Crest



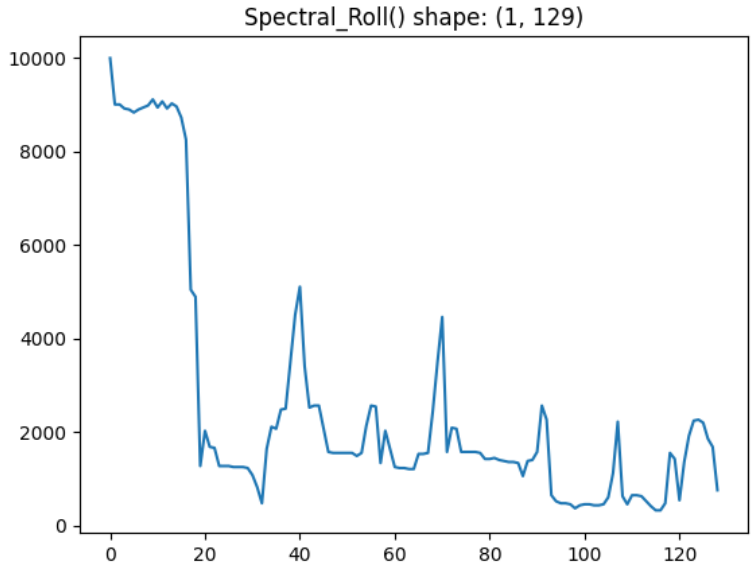
Spectral Crest of the sample in figure ...

- Spectral Flux



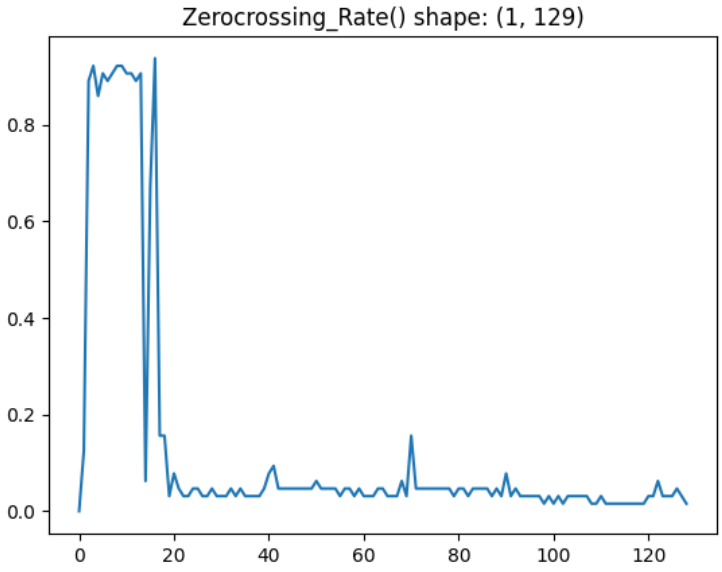
Spectral Flux of the sample in figure ...

- Spectral Roll



Spectral Roll of the sample in figure ...

- Zero crossing Rate



Zero crossing Rate of the sample in figure ...

3.5 Feature Manipulation

Many of the features are of different shapes and sizes. Since the matrix passed to the dimensionality reduction algorithm must be rectangular (i.e. a vector for each sample) a way must be found to force the features into a 1-d vector of values. I have used two approaches.

- Flattening

  If the feature matrix is 2 dimensional, we can simply concatenate consecutive rows into one feature vector transforming a (k, n) shape matrix to an array of length $k \times n$. The raw STFT and MFCC feature matrices must be flattened in this way to be used in the further steps of dimensionality reduction.

- Aggregation over selected axis

  Another approach to reducing feature dimensions is that of aggregation. We can calculate an aggregate value over a particular axis, in

essence reducing the dimensionality of the matrix by this axis. We can then treat the reduced matrix as any other feature vector. As suggested by Dupont et al. [?] and Fedden [?], I chose three aggregation functions: the average, standard deviation, and the mean of the difference between consecutive values in the vector.

In my case this procedure was applied to the following features:

- raw stft, reducing size from 32x32 -> 32x1.
- raw mfcc, reducing size from 20x257 -> 20x1.

3.6 Dimensionality Reduction

The last step in the pipeline is the final dimension reduction of the final collection of features to just 2 values. 4 different algorithms were used in this step:

- Principal Component Analysis

  The basic, tried-and-tested dimensionality reduction method. This method doesn't accept any additional parameters except the number of Principal Componenets to output. As such, for each collection of features we obtain 1 plot.

- T-stochastic neighbour embedding

  What has come to be a widespead technique in the field of machine learning for it's ability to create useful 2d maps of data. Used by McDonal et al. and Hantrakul et al. to create visualizations of audio data. T-sne's uses extend far beyond just audio data, however. It is commonly used in the field of single-cell genomics to visualize human genetic data [?] and is able to seperate samples from different continents and even reflects some local, sub-continental patterns.

  3 parameters influence the visualization in a significant way:

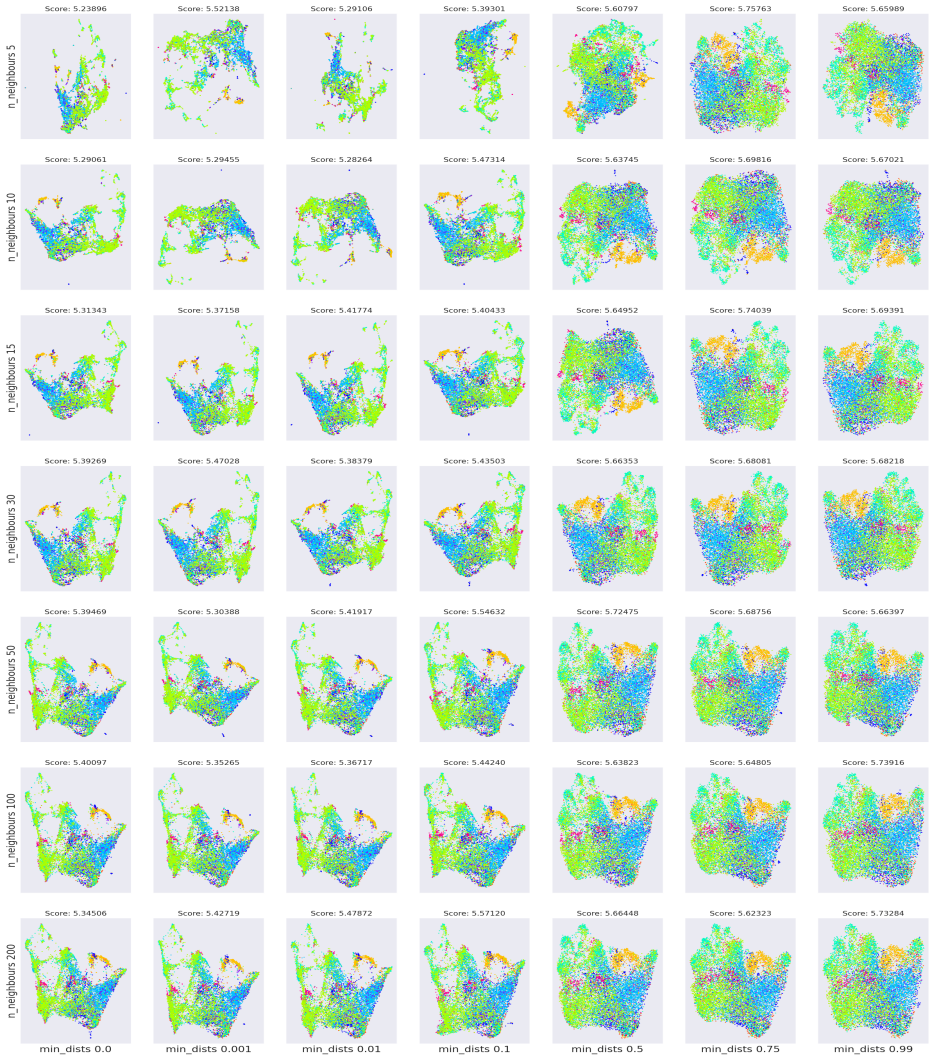
  - Perplexity, which can be interpreted as how much attention the algorithm should give to local or global structure. In the original article van der Maaten & Hinton suggested that perplexity values should generally fall in the range 5-50. In my case, smaller values tend to result in plots with less dense clusters with higher values giving more well-seperated clusters. I chose the values [5, 10, 20, 40, 60] as parameters for the t-sne plots.

- The learning rate. Usually falls in the range [10-10000]. I however determined that values higher than 1000 seemed to lose global structure. I chose the values [20, 50, 100, 200, 300]
- Iterations. I decided to go with a constant value of 3000. After heuristic tests I determined that the plots seemed to be stable for this dataset.

- Uniform Manifold Approximation and Projection

  A relatively new dimensionality method. It is very effective at preserving both the local and global structure of the original data in it's projections. Similarly to t-sne it is also based on manifold learning. The important hyperparameters are:

  - number of neighbours This parameter controls the number of approximate nearest neghbors to use to construct the initial high dimensional graph. Low values tend to focus on the fine, local structure. Higher values put an emphasis on the wider structure since they take into account a larger number of neighbour points. I chose values [5, 10, 15, 30, 50, 100, 200]
  - minimum distance is the value used by the algorithm to determine what the minimum distance points on the embedding can be from each other. I chose to sweep throught the whole range: [0.0, 0.001, 0.01, 0.1, 0.5, 0.75, 0.99]



A Umap grid search over stft.
It is clear to see that lower values of hyperparameters (top left) give more densely clustered plots with a focus on the local structure, while higher values (bottom right) are more distributed and focus more on global relationships.

Since the objective is to make a round and homogenously dispersed plot, while still keeping the clusters seperate, I suspect that values, which strike a balance between being globally spread out and having enough detailed local structure to seperate the clusters will be evaluated as teh best.

3.7 Scoring the plots

The experiment was designed with a particular goal in mind - generating plots from the original audio data which would enable an intuitive grasp of the dataset. In order to achieve this, I define several metrics for evaluating the plots:

- How well the embedding reflects the inherrent relationships between datapoints. This is measured by the Silhouette score with the class labels corresponding to cluster labels. Also, the overlap of the convex hulls of classes is an indicator of this quality.
- Readability and ease-of-navigation of the plot. In order for the plot to be readable, points should be as evenly distributed as possible, avoiding clumps, which might be hard to navigate. Ripley's function is an indicator of homogeneity of the density distribution of points on the plot. I figured, that a regular, uniform shape of the plot would increase readability, as such metric of readability is given by the Polsby-Popper method for measuring the roundness of the convex hull of the plot.

Each plot produced is scored using these metrics. Since each of these metrics had a different range of values, to compute a final score for the plot, each metric was normalized to the range $[0, 1]$. The final score for plot $p \in P$ is a weighted sum of all the normalized individual metrics given by:

$$T(p) = 2silhouette(p) + 2ripley(p) + overlap(p) + roundness(p)$$

Where:

- $silhouette(p)$ is the normalized Silhouette score. Because the silhouette metric is in the range $[-1, 1]$ the value must be shifted to be in range $[0, 1]$: $s(p) + 1$. Where $s(p)$ is the Silhouette score for the plot. The shifted silhouette score is then normalized relative to the max silhouette score, finally giving:

$$silhouette(p) = \frac{s(p) + 1}{\max\{s(p)|p \in P\}}$$

- $ripley(p)$ is a metric based on the Ripley H-function. First, the average Ripley H-function is taken for plot p for radii in the set: $R = (0.05, 0.1, 0.25, 0.5)$ $\sum_{r \in R} H_p(r)$ Since the H-function can assume values both negative and positive, I decided to take the absolute value.

This causes a loss of information, since negative values indicate dispersion and positive ones indicate clustering. However, this distinction is not important for the purposes of the experiment. The only information important to us is how much the plot deviates from being uniformly dense. Since we want values close to 1 to indicate a better score I take the inverse, giving:

$$H(p) = abs(\frac{\sum_{r \in R} H_p(r)}{|R|})^{-1}$$

With $H_p(r)$ being Ripley's H-function for plot p taken for radius r. Finally, the value is normalized relative to the max value for all plots.

$$ripley(p) = \frac{H(p)}{\max\{H(p)|p \in P\}}$$

- $overlap(p)$ is the ratio of overlap of convex hulls for the clusters to the area of the whole plot. Normalized relative to the max value for all plots:

$$overlap(p) = \frac{O(p)}{\max\{O(p)|p \in P\}}$$

- $roundness(p)$ is calculated using the Polsby-Popper method. Also normalized relative to the max value for all plots:

$$roundness(p) = \frac{PP(p)}{\max\{PP(p)|p \in P\}}$$

4 Experiment Evaluation

4.1 Describe the results and how they apply to the research question

4.2 What could be improved

4.3 limitations

4.4 suggestions for future work

4.5 Conclusion