POLISH-JAPANESE ACADEMY
OF INFORMATION TECHNOLOGY

Department of Computer Science
Specialization of Intelligent Data Processing Systems

Łukasz Piotrak
s18002

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

Bachelor's Thesis
Supervisor: Dr Sinh Hoa Nguyen

Warsaw, Poland, September 2021

Abstract

Navigation and search has long been a bottleneck activity when working with sound. Recent advancements in dimensionality reduction techniques have made it possible to replace the classic directory tree with a spacial map of audio files, resulting in a more intuitive representation. I provide a method of evaluating visualizations using methods from spatial statistics and find that a combination of PCA + STFT + UMAP methods gives the best scoring, although unexpected 2D embedding on a dataset of instrument recordings.

Abstract

Przeszukiwanie zbiorów plików od dawna sprawia trudność w efektywnej pracy z dźwiękiem. Wraz z postępem w dziedzinie redukcji wymiarowości danych, powstała możliwość zastąpienia klasycznej przeglądarki plików na bardziej intuicyjną reprezentację, tj. przestrzenną mapę plików audio. W tej pracy przedstawiam metodę na ocenę wygenerowanych przestrzennych reprezentacji poprzez zastosowanie metod ze statystyki przestrzennej. Przeprowadzam także analizę porównawczą metod ekstrakcji cech oraz redukcji wymiarowości i odkrywam, że połączenie metod PCA + STFT + UMAP daję najlepszą, mimo nieregularnych reprezentacji dla zbioru danych składających się z krótkich nagrań gry na instrumentach.

Contents

Introduction

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 [9], collections of sounds can be explored more naturally with the help of dimensionality reduction techniques.

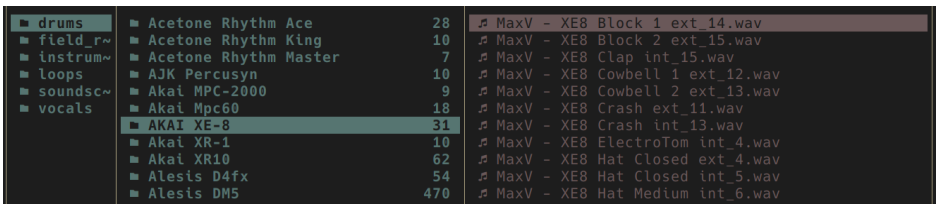


Figure 1.1: Browsing sound samples using a directory structure.

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.

1.1 Related Work

There have been a number of papers and software projects with a focus on dimensionality reduction as applied to audio datasets. This work is based on previous efforts in this domain, such as those by Hantrakul & Sarwate [16] and McDonald et. al [9] which I will describe in more detail below. I expand upon their findings by using a dataset of samples containing harmonic data, widening the pool of feature extraction methods and introducing a novel method of assessing the visualizations using Ripley's H function to measure the homogeneity of density distribution of the plots.

1.1.1 The infinite drum machine

The idea for this work originally came from the excellent web app by McDonald et. al [9]. The app organized thousands of foley sounds on a 2D plane using the t-SNE technique introduced by van Der Maaten & Hinton [18]. Similar sounding samples were placed close together. Sounds could be navigated and played back by hovering over individual points with the mouse. A sequencer was implemented to enable the arrangement of those sounds into audio loops. As far as my research showed this was one of the first applications introducing the excellent idea of using dimensionality reduction techniques to make intuitive "maps" of large audio collections. Since the app was meant more as a proof of concept I immediately saw areas in which it might be extended. Firstly, it only worked on a static dataset, meaning there was no way to make visualizations for other datasets or even add new samples to the existing visualization. Secondly, the functionality was extremely limited by the intended use case. A user is able to select several sounds and use them as samples to program a musical loop and not much besides.

1.1.2 Klustr

Klustr, by Hantrakul & Sarwate [16] is a tool for dimensionality reduction and visualization of large audio dataset. The authors used a dataset of 10,000 drum samples, shortened to a length of 0.25 seconds. The feature sets used were: Short-time-Fourier-Transform, Mel-Frequency Cepstral Coefficients, various Music Information Retrieval features (RMS, Spectral Cen-

troid, Spectral Crest, Spectral Flux, Spectral Rolloff, Zero-Crossing Rate) and finally features extracted by a Wavenet autoencoder. These features were then applied to a dimensionality reduction step using 3 different dimenionality reduction techniques: PCA, t-SNE and UMAP. The scoring function is a combination of the silhouette coefficient [1], roundness of plots measured by the Polsby-Popper test [14] and the ratio of overlap of convex hulls of the different drum classes. I have used a similar scoring system, though changing the calculation of the ratio of overlap of convex hulls and adding another scoring method. The authors determined that, for the dataset used, a combination of STFT, PCA and UMAP yielded the best 2D visualization.

Background

2.1 Music Information Retrieval

With the recent rise of music streaming platforms, which serve millions of users each day, extracting information from music to classify, categorize and build effective recommender systems seems more relevant than ever. The objective of Music information retrieval (MIR) is the extraction of such information. It is a broad field of study, which lies on the intersection of many different research domains. It uses knowledge from musicology and music theory, (music) psychology, psycho-acoustics, audio engineering, computer science and machine learning. It is rapidly growing in scope, with a rapid increase in published papers in the last few years.
MIR may be subdivided into two distinct subdomains. The first focused mainly on the analysis of non-audio music formats such as musical notation, song lyrics and even user reviews and bibliographical information (publication date, title etc). And the second, Audio Content Analysis (ACA), which uses various techniques from digital signal processing, machine learning, statistics and psychoacoustics, among others, to extract meaningful information from audio signals. Many techniques from ACA are used in machine learning, notably speech recognition and other fields, to extract psychoacoustic features from raw pulse-code modulated signals.
There are a wide range of features which describe different statistical, physical and perceptual aspects of sound. Some of the most commonly used in the field of MIR are described bellow.

2.1.1 Short-time Fourier Transform (STFT)

The Short-time Fourier Transform is a representation of a signal obtained by taking the Fourier transform of short segments of time. The method used to obtain the STFT is relatively straightforward. First, the signal is divided

into shorter, equal-length segments using a window function. These segments usually have some overlap in order to prevent artifacts of the original signal (so-called spectral leakage) Multiple window functions could be used for this step, however a Hann window is usually used. A window length of 10 to 300ms is common. Smith [17] points out three reasons for doing so:

- The human ear analyzes short fragments of signals at a time (10-20ms).
- Signals change over time. It is best to analyze over a time frame where the spectral content stays relatively constant.
- Computation of the fourier transform is costly and computing a whole signal at once may inefficient and undesirable.

Next, the Discrete Fourier Transform (DFT) is computed for each STFT segment S_i:

$$S_i(k) = \Sigma_{n=1}^{N} x(n) h_i(n) e^{-j2\pi kn/N} 1 \leq k \leq K \tag{2.1}$$

Where $x(n)$ is the original signal
$h_i(n)$ is the i-th analysis window,
N is the number of samples in the signal and K is the number of frequency bins in the DFT.
We can thus observe how the frequency content of the signal changes as we progress through time.

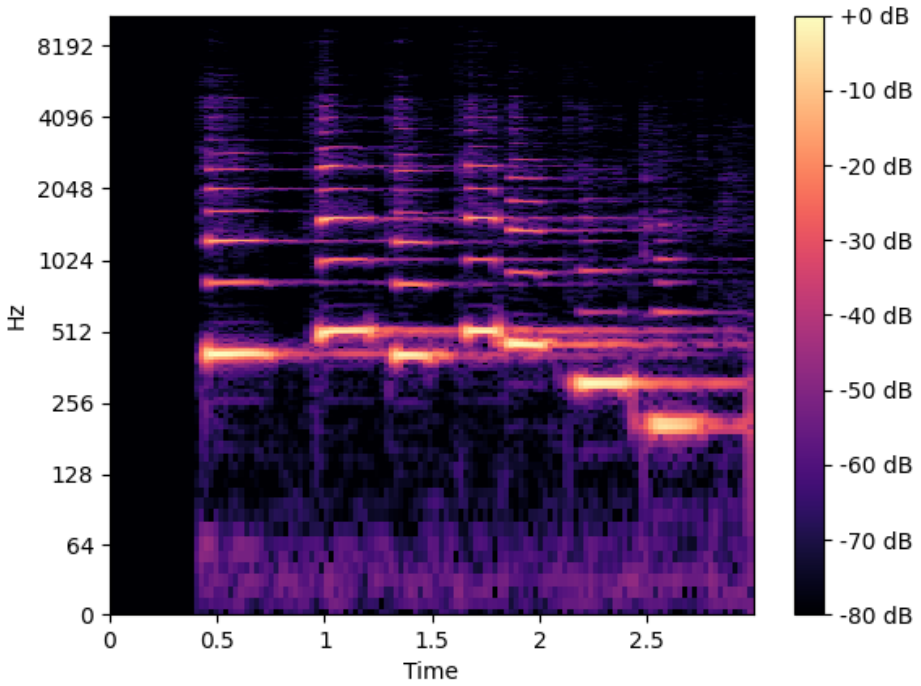


Figure 2.1: An STFT spectrogram

2.1.2 Mel Frequency Cepstral Coefficients (MFCC)

Davis and Mermelstein [4] first introduced the concept of MFCC in 1980. Several parametric representations of acoustic signals were compared with an emphasis placed on providing a way of distinguishing between phonetically similar monosyllabic words. This resulted in the development of a compact representation, which could provide an efficient way of reflecting phonetic differences between phonemes. Inspired by the observation that humans percieve sound in a logarithmic, not linear scale. A key component to the success of the MFCC representation is the initial transformation of the signal using the Mel-filterbank seen in 2.3. Earlier work by Pols [13], suggested that a cosine series expansion on these Mel-filtered frequency energies which might yield an efficient representation of audio signals for speech recognition systems. An example MFCC spectrogram can be seen in 3.4. The main steps for calculating MFCC may be summarized as follows:

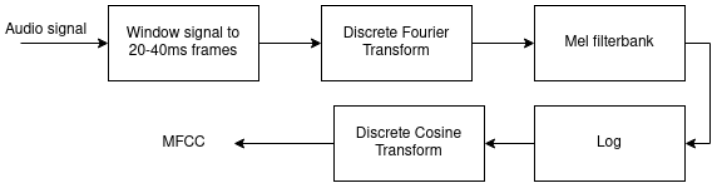


Figure 2.2: A high-level diagram of the MFCC calculation algorithm.

1. Split the signal into windowed fragments using a window function, typically a Hamming window is used [3] with a period of 20-40ms. This period allows us to obtain a reliable estimate of the spectral features at the particular window while also roughly corresponding to the temporal resolution of human hearing. A longer period would mean losing information about the spectral variation.

2. Similarly to the calculation of STFT, the Discrete Fourier Transform 2.1 is applied to each window Usually, a 512 or 256 frequency bin DFT is calculated.

3. Next, the Mel filterbank is applied to each window (shown in 2.3) to convert the frequencies to the Mel scale. The Mel scale is used because the human perception of sound is more sensitive to differences in pitch at low rather than high frequencies. The formula for converting from the frequency scale to the Mel scale is:

$$M(f) = 2595 \log_{10}(1 + f/700)$$

   Commonly, 26 filters are used and applied to the result of the previous step. To calculate the filterbank energies, each filterbank (the length of which corresponds to the length of the periodogram) is multiplied with the power spectrum. The result is summed, the result of which indicates the amount of energy in the periodogram power spectrum for that particular filterbank. The result of this step is 26 coefficients corresponding to the 26 filterbank energies. The output filterbank energies may be calculated as:

$$e(i) = \Sigma_{k=1}^{M} |Y(k)|^2$$

4. The log for each filterbank energy is taken.

5. The Discrete Cosine Transform (DCT) is taken for each filterbank energy log, giving 26 cepstral coefficients. In effect this gives the formula:

$$MFCC_m = \sqrt{\frac{2}{Q}} \Sigma_{l=0}^{Q-1} \log[e(l+1)] \cdot \cos\left[m \cdot (\frac{2l-1}{2}) \cdot \frac{\pi}{Q}\right]$$

Where $0 \leq m \leq R-1, R$ is the number of cepstrum coefficients and Q, the number of filterbanks used.

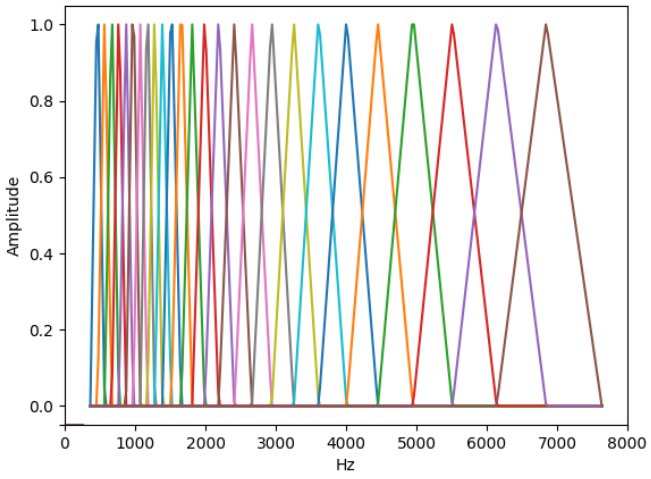


Figure 2.3: The mel filterbank

2.1.3 MIR metrics

Spectral Centroid

The mean of the normalized distribution over frequency per frame of a spectrogram.

$$centroid(t) = \frac{\sum_{k=b_1}^{b_2} f_k s_k}{\sum_{k=b_1}^{b_2} s_k}$$

where:

- s_k is the mean amplitude at bin k
- f_k is the center frequency value at bin k.
- b_1 and b_2 are the bin boundaries between which to calculate the centroid.

Spectral Rolloff

The frequency for each window such that at least k percent of the energy of the spectrum is contained in this frequency bin and the bins bellow. The

rolloff point is i such that:

$$\sum_{k=b_1}^{i} s_k = k \sum_{k=b_1}^{b_2} s_k$$

where:

- s_k is the mean amplitude at bin k.
- k is the percentage of energy contained between b_1 and i.
- b_1 and b_2 are the bin boundaries between which to calculate the spread.

Spectral Bandwidth

Given by:

$$(\sum_{k} s_k(f_k - f_c)^p)^{\frac{1}{p}}$$

where:

- s_k is the mean amplitude at bin k.
- f_k is the bin at k.
- f_c is the spectral centriod at k.

Spectral Crest

The crest of the power spectrum over time. Given by:

$$crest = \frac{max(s_k \in [b_1, b_2])}{\frac{1}{b_2 - b_1} \sum_{k=b_1}^{b_2} s_k}$$

where:

- s_k is the mean amplitude at bin k.
- b_1 and b_2 are the bin boundaries between which to calculate the crest.

Spectral Flux

Can intuitively be thought of as how much the signal changes over time. Given by:

$$flux(t) = (\sum_{k=b_1}^{b_2} |s_k(t) - s_k(t-1)|^p)^{\frac{1}{p}}$$

where:

- s_k is the amplitude at time t of bin k.
- b_1 and b_2 are the bin boundaries between which to calculate the flux.
- p is the normalization type.

Spectral Flatness

Measures how much noise-like a sound is. Given by:

$$flatness = \frac{(\prod_{k=b_1}^{b_2} s_k)^{\frac{1}{b_2-b_1}}}{\frac{1}{b_2-b_1}\sum_{k=b_1}^{b_2} s_k}$$

where:

- s_k is the amplitude at bin k.
- b_1 and b_2 are the bin boundaries between which to calculate the flux.

Root-Mean-Square (RMS)

The root-mean-square. Given by:

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1}^{N} |x_n|^2}$$

Where x_n is sample at position n and N is the number of samples.

Zero-Crossing Rate

How many times a signal crosses zero a given time frame.

2.2 Dimensionality reduction

2.2.1 Principal Component Analysis (PCA)

PCA is an orthogonal linear transformation, which projects data onto a new system of axes. The projection is such that the first axis accounts for the greatest variance in the data, the second axes for the second greatest, and so on. The computation is described as follows:

1. **Normalization**.

   By transforming each variable to be in the same scale.

   $$Norm(x) = \frac{x - \mu(X)}{\sigma(X)}$$

2. **Computing the covariance matrix**

   If we have a vector of random variables $(X_1, \ldots, X_n)$ then the covariance matrix is defined as:

   $$K_{XX} = \begin{bmatrix} \mathrm{Cov}(X_1, X_1) & \ldots & \mathrm{Cov}(X_1, X_n) \\ \vdots & \ddots & \\ \mathrm{Cov}(X_n, X_1) & \ldots & \mathrm{Cov}(X_n, X_n) \end{bmatrix}$$

   Where $\mathrm{Cov}(X_i, X_h)$ is defined as:

   $$\mathrm{Cov}(X_i, X_j) = \mathrm{E}(X_i X_j) - \mathrm{E}(X_i)\mathrm{E}(X_j)$$

   $\mathrm{E}(X_i)$ is the expected value given as:

   $$\mathrm{E}(X_i) = \sum_{j=1}^{k} x_j p_j$$

   and p_j is the probability of x_j.

   Since $\mathrm{Cov}(X_i, X_i) = \mathrm{Var}(X_i)$, the diagonal is composed of the variances of each variable. Also, since covariance is commutative ($\mathrm{Cov}(X_i, X_j) = \mathrm{Cov}(X_j, X_i)$), the matrix is symmetric with respect to the diagonal.

3. **Calculate the Principal Components of the covariance matrix**

   Principal Components (or eigenvectors) for a linear transformation are the vectors which stay on their own span. Knowing this property we

can define it as an equation for a transformation A, eigenvector v and eigenvalue λ:

$$Av = \lambda v \tag{2.2}$$

After inserting an identity matrix (I) and bringing all to the left side:

$$Av - \lambda Iv = 0$$

Since v must be non-zero we can use the determinant. From this equation we should get a polynomial. The roots of this polynomial give us the eigenvalues.

$$\det(A - \lambda I) = 0$$

Once the eigenvalues are known we can easily determine the corresponding eigenvector by solving for v in 2.2.

Now, the eigenvectors can be ordered by their eigenvalues. We can compute the percentage of variance accounted for by each component by dividing the eigenvalue of a component by the sum of eigenvalues.

4. **Project the standardized data onto axes represented by the Principal Components**

$$P = C^T D^T$$

Where P is the projected data, C the matrix containing the Principal Components and D the original, standardized data.

2.2.2 t-Distributed Stochastic Neighbour Embedding (t-SNE)

t-Distributed Stochastic Neighbour Embedding is a nonlinear dimensionality reduction technique used for embedding high-dimensional datasets into 2 or 3 dimensions. It is based on the original Stochastic Neighbour Embedding (SNE), developed by Roweis and Hinton [8] in 2002 and improved by Maaten and Hinton [18] in 2008 by adding the use of the t-distribution. The goal of the algorithm is to minimize the divergence between two distributions: a distribution that measures pairwise similarities of the input objects (which we will refer to as x_i) and a distribution that measures pairwise similarities of the corresponding low-dimensional points of the resulting embedding (referred to as y_i). The steps for calculating t-SNE may be described as follows:

1. Begin by converting of the Euclidean distances between points in the original dataset into conditional probabilities that represent how similar points are to each other. From the original t-SNE paper: "The similarity of datapoint x_j to datapoint x_i is the conditional probability, $p_{j|i}$, that x_i would pick x_j as its neighbor if neighbours were picked in proportion to their probability density under a Gaussian centered at x_i." In essence, this means that points which are close together in the original dataset have a high value $p_{j|i}$ while those which are further away obtain lower values in proportion to their distance. The condition $p_{j|i}$ is given by:

$$p_{j|i} = \frac{\exp(\frac{-||x_i - x_j||^2}{2\sigma_i^2})}{\Sigma_{k \neq i} \exp(\frac{-||x_i - x_k||^2}{2\sigma_i^2})}$$

Where σ_i is the variance of the Gaussian that is centered on datapoint x_i. There is most likely no single σ_i optimal for all datapoints as the density of the data will often vary depending upon the point in question. A particular value of the variance gives a probability distribution P_i over all the other datapoints. A binary search is performed as seen in 1 to find the value of σ_i, that produces a P_i with a perplexity value, which is given as a parameter to the t-SNE algorithm, defined as:

$$Perp(P_i) = 2^{H(P_i)}$$

where $H(P_i)$ is the entropy of P_i:

$$H(P_i) = -\Sigma_j p_{j|i} \log p_{j|i}$$

The probability is defined as:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N}$$

with $p_{ij} = p_{ji}$, $p_{ii} = 0$ and $\Sigma_{i,j} p_{ij} = 1$.

The Implementation uses the precision β $(1/2\sigma^2)$ instead of the variance. Now, the entropy is:

$$H = \sum_j p_{j|i}(log(S_i) + ||x_i - x_j||^2 \beta_i$$

Algorithm 1: Binary search for the precision value

Input:
Perplexity: The input perplexity to use
Tolerance: The tolerance to use when testing if a given precision is close enough to the perplexity value
Output: β: Precision values for each point x_i

$\beta \leftarrow$A vector of length N with entries equal to 1
for $\beta_i \in \beta$ **do**
  $\beta_{min} \leftarrow -\infty$;
  $\beta_{max} \leftarrow \infty$;
  Hdiff $\leftarrow H(\beta_i) - \log_2(Perplexity)$;
  while $\mid Hdiff \mid - Tolerance > 0$ **do**
    if $Hdiff > 0$ **then**
      $\beta min \leftarrow \beta_i$;
      if $\beta_{max} = \infty \vee \beta_{max} = -\infty$ **then**
        $\beta_i \leftarrow \beta_i^2$;
      else
        $\beta_i \leftarrow \frac{\beta_i + \beta max}{2}$;
    else
      $\beta max \leftarrow \beta_i$;
      if $\beta min = \infty \vee \beta max = -\infty$ **then**
        $\beta_i \leftarrow \beta_i / 2$;
      else
        $\beta_i \leftarrow \frac{\beta_i + \beta min}{2}$;
    Hdiff $\leftarrow H(\beta_i) - \log_2$Perplexity;
  end
end
return Y;

2. Next, a measure of similarity, q_{ij} is defined between two points y_i and y_j (these represent the low-dimensional models of x_i and x_j). If d is the number of dimensions, then $y_i \in \mathbb{R}^d$:

$$q_{ij} = \frac{\exp(1 + ||y_i - y_j||^2)^{-1}}{\Sigma_k \Sigma_{l \neq k} \exp(1 + ||y_k - y_l||^2)^{-1}}$$

Here, the distribution is Student's t-distribution with one degree of freedom. The distribution is chosen because $(1 + ||y_i - y_j||^2)^{-1}$ approaches an inverse square law for large distances between points in the low-dimensional map. This allows dissimilar datapoints to be modeled as far apart in the low-dimensional embedding.

The points are then distributed randomly throughout the low-dimensional space.

3. Gradient descent 2 is performed on the low-dimensional embedding of the data in such a way as to minimize the Kullback-Leibler divergence between p and q. The divergence is defined as:

$$KL(P||Q) = \Sigma_i \Sigma_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

The gradient descent minimizes a Kullback-Leibler divergence between a joint probability distribution P and the joint probability distribution Q:

$$C = KL(P||Q)$$

In general, the cost is large if widely seperated points in the embedding are used to represent points which are nearby in the original data (using a small $q_{i|j}$ to model a large $p_{j|i}$). Conversely using nearby embedding points to represent widely seperated datapoints has a small cost. This gives a focus on retaining local structure in the embedding. The final gradient function is:

$$\frac{\delta C}{\delta y_i} = 4\Sigma_j(p_{ij} - q_{ij})(y_i - y_j)(1 + ||y_i - y_j||^2)^{-1}$$

Algorithm 2: Gradient descent

Input:
α: The learning rate
f: The cost function
x: Initial value
t: Tolerance
Output: x The argument for which the cost function has been minimized

```
while true do
    cost ←derivative(f(x));
    x ← x − α· cost;
    if cost < t then
        break;
    end
end
return x
```

2.2.3 Uniform Manifold Approximation and Projection (UMAP)

Similar to t-SNE, UMAP is a nonlinear technique for dimension reduction. It was developed by McInnes et. al. in 2018. Advertised as being able to preserve more global structure and having a superior run time than t-SNE, UMAP has seen a giant growth in popularity in the last few years. Unlike t-SNE it has no restrictions on embedding dimensions, which means it can be used as a general-purpose dimensionality reduction technique for machine learning.
An overview of the UMAP algorithm is presented below:

Algorithm 3: High-level UMAP algorithm

Input: X: Set of datapoints
n: the neighborhood size to use for local metric approximation
d: the dimension count of the reduction
min-dist: a parameter specifying the minimum distance between points in the embedding
n-epochs: a parameter controlling how much work to perform
Output: Y: Dataset with reduced dimensions

for $x \in X$ **do**
| $SS \leftarrow \bigcup_{x \in X} \text{localFuzzySimplicialSet}(X, x, n)$
end
$Y \leftarrow$ spectralEmbedding(SS, d)
$Y \leftarrow$ optimizeEmbedding(SS, Y, min-dist, n-epochs)
return Y

This process starts by the construction of local fuzzy simplicial sets for each datapoint. These are constructed by finding the n nearest neighbors according to a selected distance metric. Next a normalized, approximate distance between the neighbors on the manifold is calculated as seen in algorithm 5. The next step is the construction of the union of all local fuzzy simplicial sets after which Spectral embedding is performed as shown in algorithm 6. The last step is the optimization of the embedding by minimizing the fuzzy set cross entropy by using stochastic gradient descent.

Algorithm 4: localFuzzySimplicialSet

Input: X: Set of datapoints
x: Datapoint to calculate the local simplicial set for
n: the neighborhood size to used for local metric approximation
Output: fs-set: output fuzzy simplicial set

knn, knn-dists $\leftarrow$ approxNN(X, x, n);
$\rho \leftarrow$ knn-dists[1];
$\sigma \leftarrow$ smoothKNNDist(knn-dists, n, ρ);
fs-set$_0 \leftarrow X$ `// Fuzzy set image of 0-simplices`
fs-set$_1 \leftarrow \{([x, y], 0) | y \in X\}$ `// Fuzzy set image of 1-simplices`
for $y \in knn$ **do**
  $d_{x,y} \leftarrow \max(0, \text{dist}(x, y) - \rho)/\sigma$;
  fs-set$_1 \leftarrow$ fs-set$_1 \bigcup([x, y], \exp(-d_{x,y}))$;
end
return fs-set

Algorithm 5: smoothKNNDist

Input: knn-dists: distances of k-nearest neighbors
n: the neighborhood size to used for local metric approximation
ρ: Distance to nearest neighbor
Output: σ: normalizing factor for distances

binSearch(knn-dists) such that
$\Sigma_{i=1}^{n} \exp(-(\text{knn-dists}_i - \rho)/\sigma) = \log_2(n)$ **return** σ

Algorithm 6: spectralEmbedding

Input: top-rep: Fuzzy simplicial set representation of the data
d: the neighborhood size to used for local metric approximation
Output: Y: Spectral embedding

$A \leftarrow$ 1-skeleton of top-rep expressed as a weighted adjacency matrix
$D \leftarrow$ degree matrix for the graph A
$L \leftarrow D^{1/2}(D - A)D^{1/2}$
evec $\leftarrow$ sorted Eigenvectors of L
$Y \leftarrow$evec$[1 \dots d + 1]$
return Y

2.3 Evaluation metrics

After reducing the dimensionality of the dataset to 2 dimensions, a scatter plot may be constructed with each audio file being represented as a single point. Below are detailed description of measures I chose to describe the correctness and usability of plots.

2.3.1 Silhouette Score

The Silhouette Score is a measure of how well an algorithm clusters data. More precisely, it measures how closely a datapoint resembles other points from it's own cluster compared with other clusters. The score is generally higher for convex, well-seperated and dense clusters.

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

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

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

where $d(i, j)$ is the distance between points i and j.
b is the mean distance between a sample $i \in C_k$ and all other points in the nearest cluster C_l. Given by:

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

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 clusters and values closer to $+1$ indicating highly dense clusters. Scores around zero indicate overlapping clusters.

2.3.2 Roundness

The overall roundness of the plot is calculated by using the method proposed by Polsby & Popper [14]:

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

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

2.3.3 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 union of convex hulls of each class to the sum of areas of the convex hulls of each cluster:

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

Where $A(U)$ is the 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.

2.3.4 Ripley's K function

Ripley's K function is a method used in spatial analysis to describe how a collection of points are distributed over a given area. It provides a measure of how clustered, dispersed or randomly distributed the measured phenomenon appears to be in the area of interest. The 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 number of 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.

Experiment design and overview

3.1 Dataset

I used the Medley-Solos-db dataset assembled by Lostanlen et al. [11]. Downloaded through the *mirdata* Python library [2]. 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 as seen in 3.1. I use the training subset of this dataset. The distribution of sounds is summarized in table

clarinet	elec. guitar	f. singer	flute	piano	tenor sax	trumpet	violin
732	955	1142	3167	2609	325	406	2899

Table 3.1: The distribution of instrument classes

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. A short overview of the pipeline is given below. Each step is then described in greater detail.

1. **Preprocessing.**

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

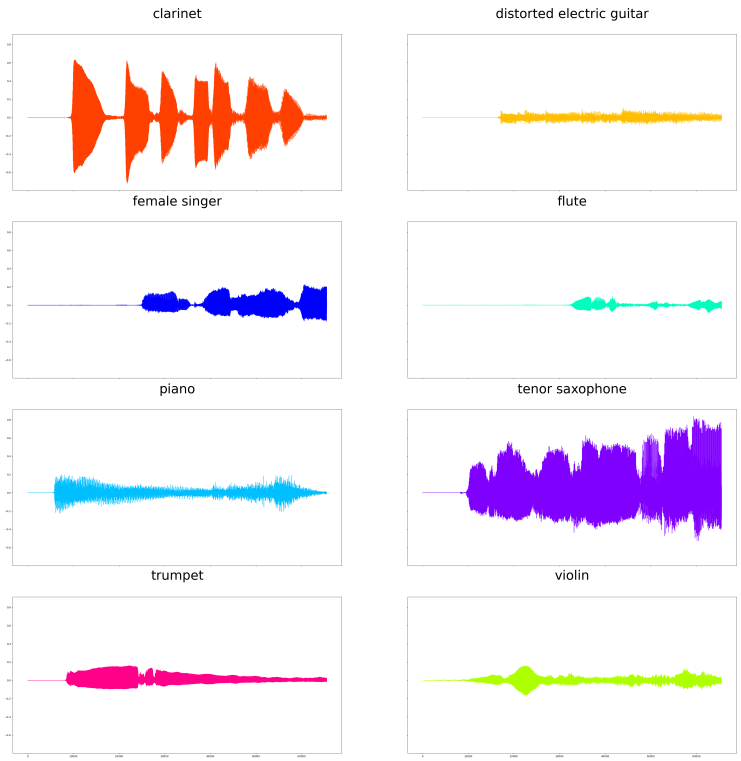


Figure 3.1: Selected sample belonging to each of the instrument classes. Each consists of 65,536 32-bit floating point numbers.

2. **Feature Extraction.**

   Characteristics of the sound are extracted using a selection of algorithms and mathematical tools found in 3.2. 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 include reshaping feature matrices and aggregation.

4. **Dimensionality Reduction.**

   After selecting a set of features to serve as a representation of the original files PCA, t-SNE or UMAP is used 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 is added.

3.3 Preprocessing

The data is ingested using the Librosa python library [12] used for music and audio analysis. The "librosa.load" method was used to convert the WAV files to a float32 numpy ndarray.

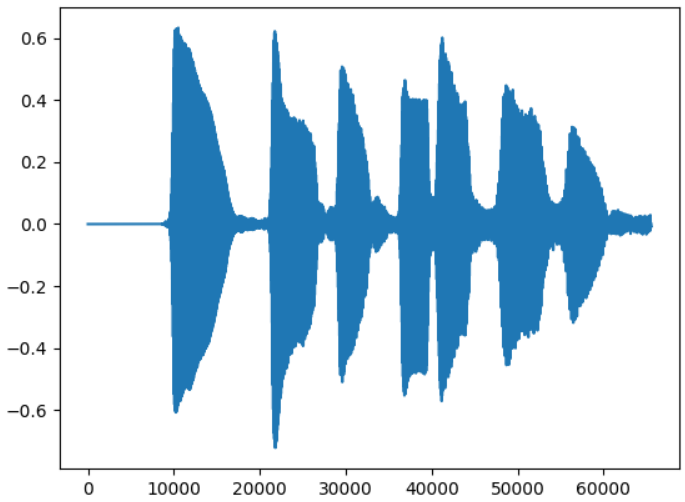


Figure 3.2: A sample from the clarinet class

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.

3.4.1 Short-time Fourier Transform (STFT)

The Short-time Fourier 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 implementation. I decided to take the STFT over 32 windows in both the time and frequency domains, finally giving a 32x32 matrix:

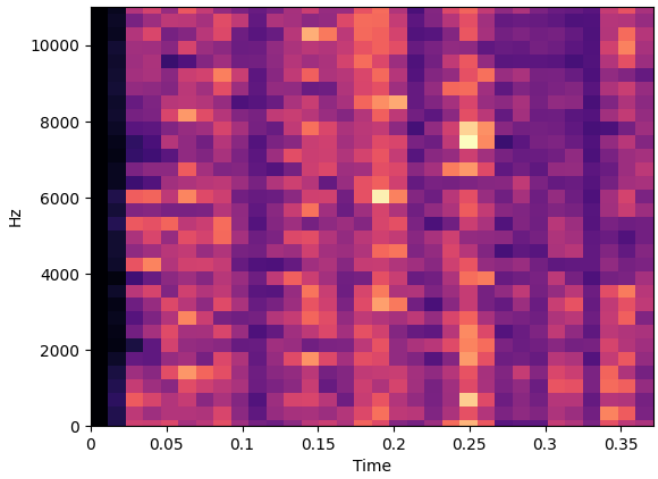


Figure 3.3: STFT of the sample in figure 3.2

3.4.2 MFCC

The Mel-Frequency Cepstral Coefficients are a heavily used in both speech recognition and Music Information Retrieval [7, 16, 15, 5]. I used the librosa implementation to calculate the MFCC's. I decided to go with 20 Cepstral Coefficient and a hop length of 256, resulting in a feature size of 20x257:

3.4.3 Music Information Retrieval Metrics

A number of metrics from the field of audio analysis has also found to be useful when extracting timbre information from audio signals [5, 16]. In my

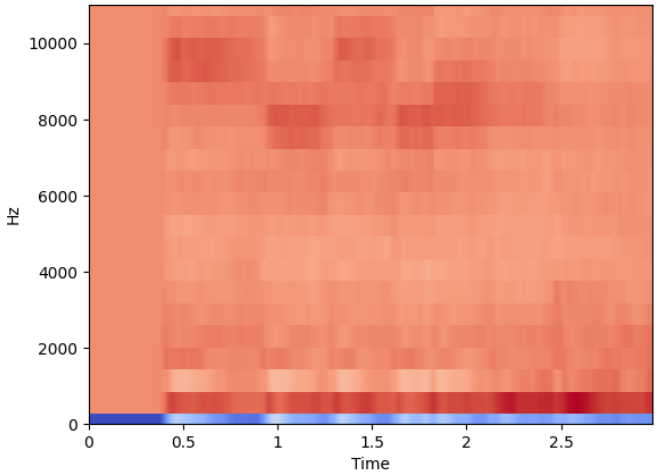


Figure 3.4: MFCC of the sample in figure 3.2

case these will include all those described in 2.1.3

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 1D 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

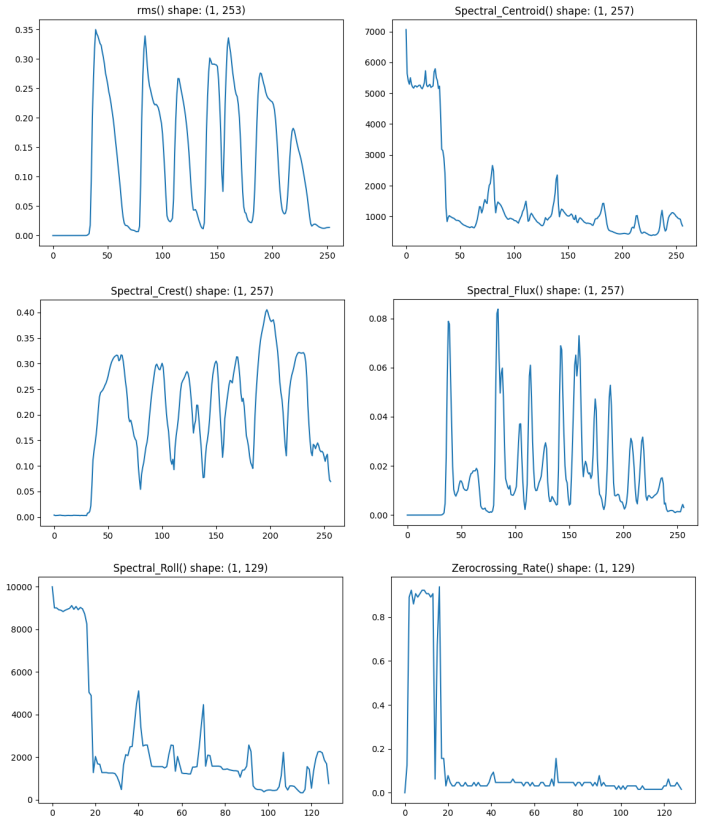


Figure 3.5: Graphs of MIR features for the sample in figure 3.2

by Dupont et al. [6] and Fedden [7], 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 obtain the following four features:

- Raw STFT, reducing size from 32x32 -> 32x3. I call this "statistically shortened STFT"
- Raw MFCC, reducing size from 20x257 -> 20x3. I call this "statistically shortened MFCC"
- A second time to the statistically shortened STFT, reducing size from 32x3 -> 3. I call this "vertically statistically shortened STFT"
- A second time to the statistically shortened MFCC, reducing size from 20x3 -> 3. I call this "vertically statistically shortened

MFCC"

Inspired by the successes of Klustr [16], with using a preliminary dimensionality reduction step on raw features using PCA before applying dimensionality reduction proper. I also tried this approach on the raw STFT and MFCC features. I decided to make features out of the top n principal components, where n took the values: [3, 5, 8, 12, 20].
The final list of features is shown in table 3.2

Metric Name	Dimensions	Description
STFT	32x32	Original STFT
stat STFT	32x3	statistically shortened STFT
vert stat STFT	3	vertically statistically shortened STFT
PCA 3 STFT	3	First 3 principal components of the original STFT
PCA 5 STFT	5	First 5 principal components of the original STFT
PCA 8 STFT	8	First 8 principal components of the original STFT
PCA 12 STFT	12	First 12 principal components of the original STFT
PCA 20 STFT	20	First 20 principal components of the original STFT
MFCC	20x257	Original MFCC
stat MFCC	20x3	statistically shortened MFCC
vert stat MFCC	3	vertically statistically shortened MFCC
PCA 3 MFCC	3	First 3 principal components of the original MFCC
PCA 5 MFCC	5	First 5 principal components of the original MFCC
PCA 8 MFCC	8	First 8 principal components of the original MFCC
PCA 12 MFCC	12	First 12 principal components of the original MFCC
PCA 20 MFCC	20	First 20 principal components of the original MFCC
stat STFT + stat MFCC	52x3	stat MFCC appended to stat STFT
all mir	8x3	statistically shortened mir metrics
reduced mir	6x3	mir metrics without rms and centroid
PCA 3 MFCC + reduced mir	7x3	reduced mir with PCA 3 MFCC
PCA 3 STFT + reduced mir	7x3	reduced mir with PCA 3 STFT

Table 3.2: All feature sets used for the dimensionality reduction step

3.6 Dimensionality Reduction

The last step in the pipeline is the final dimension reduction of the final collection of features to just 2 dimensions. 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 [10] 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.

  - **Learning rate**

    Usually falls in the range [10-1000]. I however determined that values higher than 1000 seemed to lose global structure. I chose the values [20, 50, 100, 200, 300]

  - **Number of 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 reduction 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 neighbors 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]

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 the 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 them had a different range of values, to compute a final score for the plot, each metric was normalized to the range $[0, 1]$. After experimentally selecting the weights, 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:

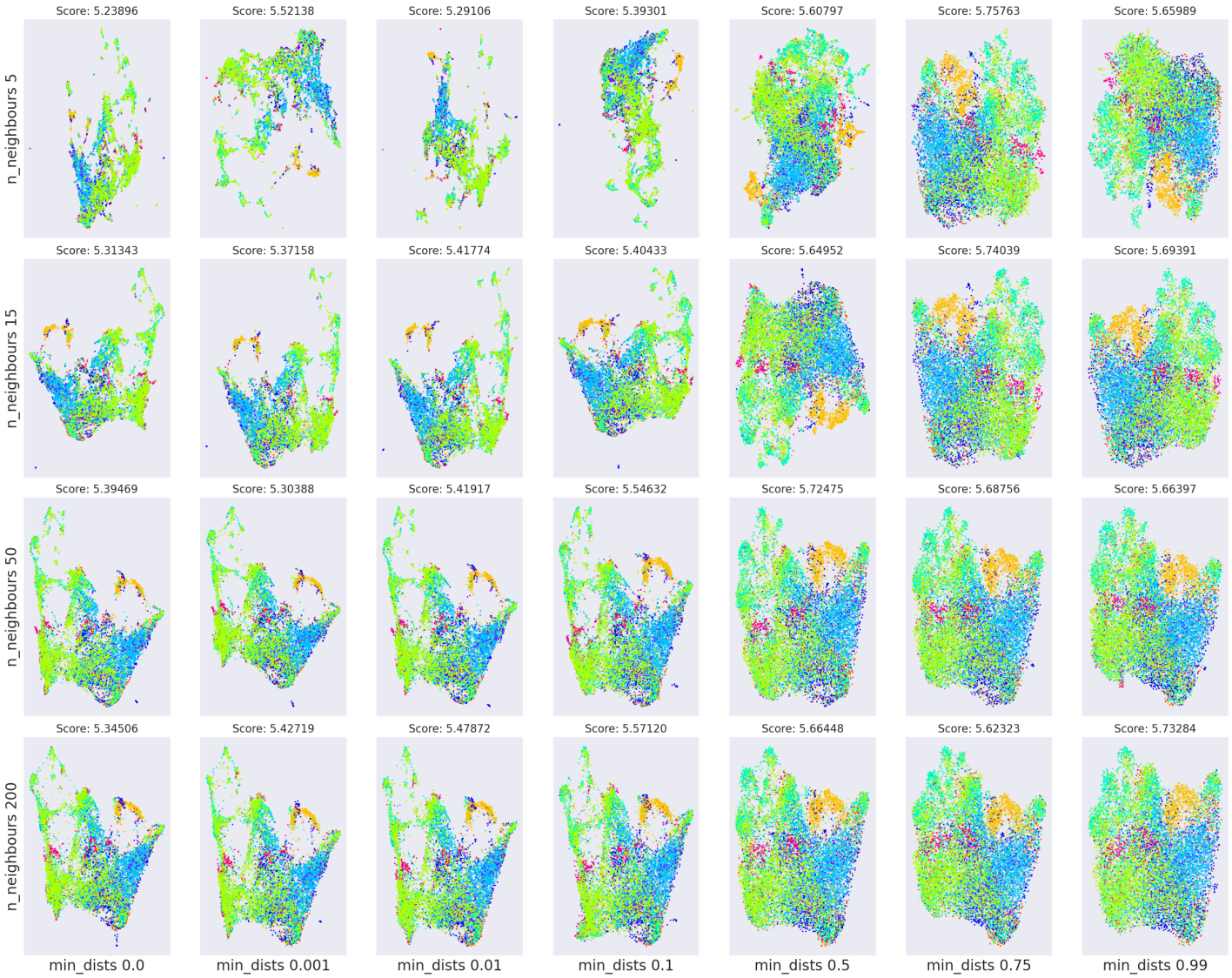


Figure 3.6: A UMAP parameter sweep with STFT as the feature.
It is clear to see that lower values of hyperparameters (top left) give more densely clustered plots with a focus on the local structure, while higher values (bottom right) are more distributed and focus more on global relationships.

- $silhouette(p)$ 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. It is interpreted as a measure of how uniformly distributed points are on a spacial plane. In order to calculate it, 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:

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

Experiment Evaluation

4.1 Overall Results

The final, best scoring plots for different feature sets are shown in 4.1, 4.2, 4.3. Most of the plots seperate the instrument classes relatively well. They also seem to exhibit a round and fairly uniform structure. The exceptions are plots which have gone through a preliminary PCA reduction step, leaving 3 prinicipal components. These are labeled *PCA 3 STFT* and *PCA 3 MFCC*. The interesting, string-shaped plots only appear with the combination of *PCA 3 STFT* and *PCA 3 MFCC* with *UMAP* . This is most probably a side-effect of the way in which UMAP builds it's high dimensional representation of the data by using a fuzzy simplical complex. UMAP always connects points to their nearest neighbour in the high-dimensional representation. I hypothesize that the string-like shapes are the joined nearest neighbours. UMAP then decides that the global structure contained in the 3d PCA plot shown in 4.4 is best represented by spacing the strings as visible in the plots. A closer look at the UMAP grid search (4.5) reveals that this is most probably the case. The more nearest neighbours the UMAP algorithm uses to determine connectedness in the high-dimensional representation, the more pronounced becomes the pattern.
This phenomenon is all the more interesting when we take a look at the top scoring plots overall, shown in 6. All the best scoring plots are from this particular feature set.
This result is counterintuitive. It seems that many of the other plots would score better on each metric. The plots seem to be less clustered and less uniformly distributed. When looking closely at the values, we can see that the silhouette and and overlap scores for the *PCA 3* plots are very close the scores of the other plots. However, when we look at the Ripley score, we see that these plots have values significantly closer to zero (meaning a higher score after normalization).

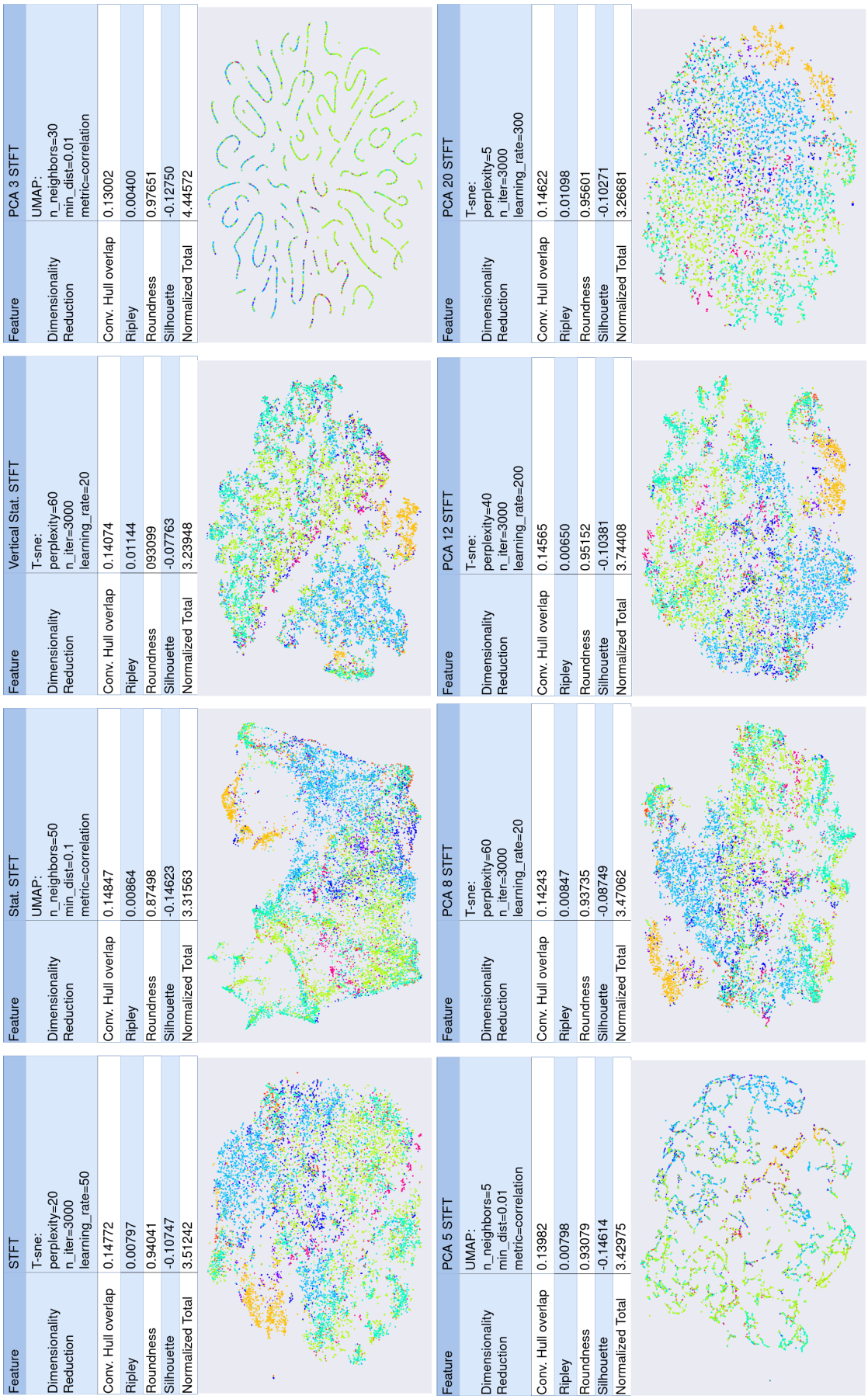


Figure 4.1: The best scoring plots for STFT features.

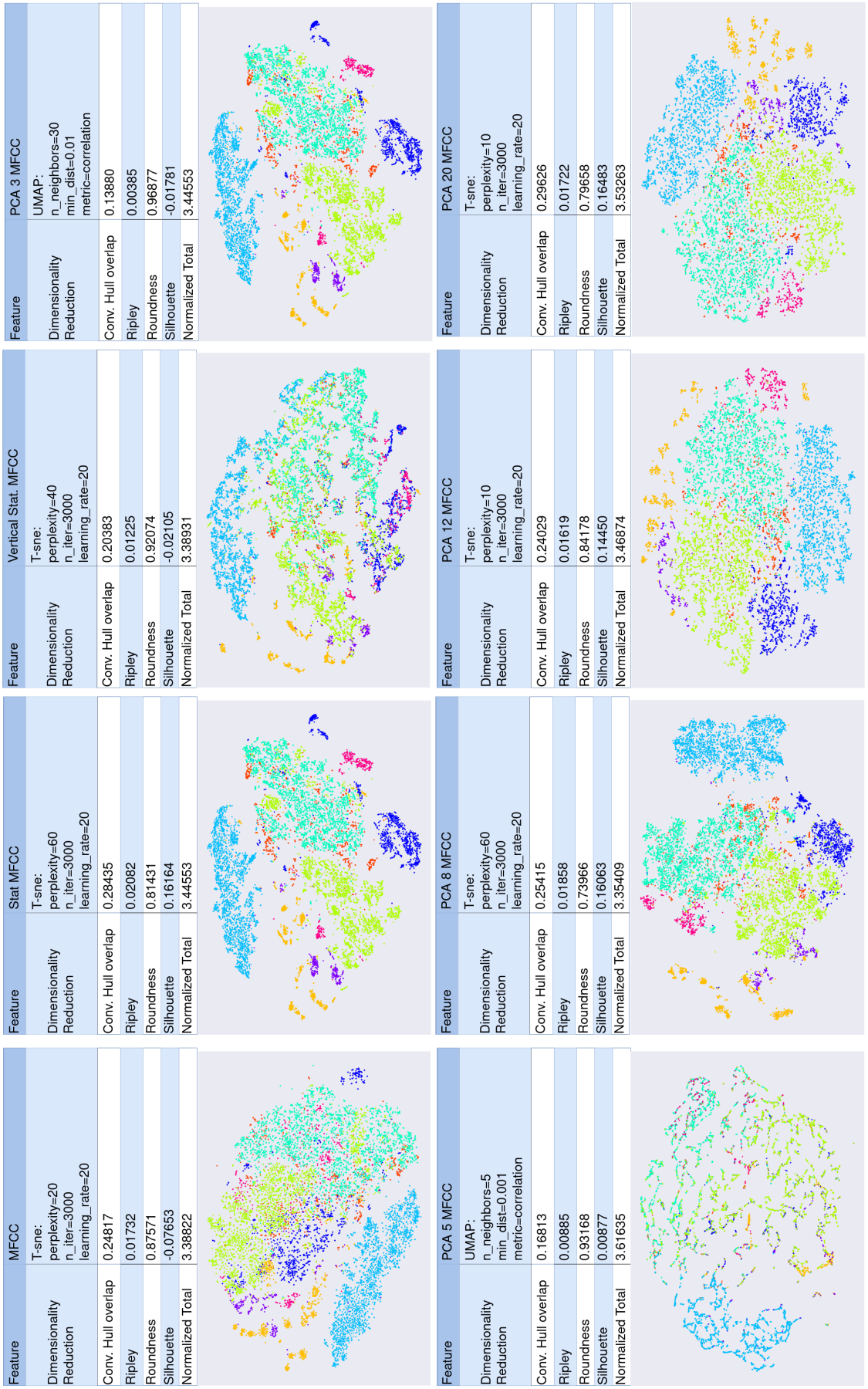


Figure 4.2: The best scoring plots for MFCC features.

Feature	Reduced MIR
Dimensionality Reduction	T-sne: perplexity=20 n_iter=3000 learning_rate=100
Conv. Hull overlap	0.16606
Ripley	0.00734
Roundness	0.95173
Silhouette	-0.01415
Normalized Total	3.77854

Feature	PCA 3 MFCC + Reduced MIR
Dimensionality Reduction	UMAP: n_neighbors=15 min_dist=0.75 metric=correlation
Conv. Hull overlap	0.13566
Ripley	0.00536
Roundness	0.94146
Silhouette	-0.10576
Normalized Total	3.96195

Feature	Stat. STFT + Stat. MFCC
Dimensionality Reduction	T-sne: perplexity=60 n_iter=3000 learning_rate=100
Conv. Hull overlap	0.27388
Ripley	0.01730
Roundness	0.84680
Silhouette	0.13488
Normalized Total	3.49438

Feature	All MIR
Dimensionality Reduction	T-sne: perplexity=40 n_iter=3000 learning_rate=200
Conv. Hull overlap	0.16571
Ripley	0.01023
Roundness	0.91621
Silhouette	0.02303
Normalized Total	3.49944

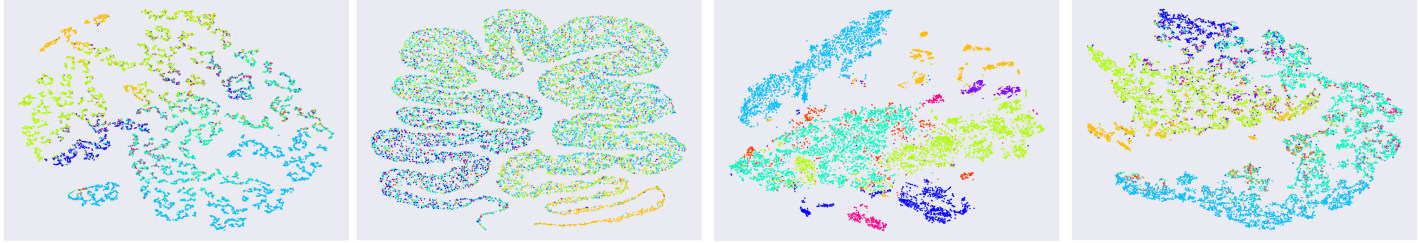


Figure 4.3: The best scoring plots for MIR features and combined feature sets.

4.2 The Ripley metric

Looking at the top scoring plots for the ripley metric presented in 7, it is clear that the PCA 3 plots score very highly. The first 8 top scoring ripley metrics come from a combination of PCA 3 + UMAP. My hypothesis is that each grouping of points is composed in such a way that the distance between groupings of points is equal to that of the min dist UMAP parameter. 4.5 seems to show this as the points on the strings get slightly farther apart as the *min dists* parameter increases. The clusters themselves seem to be roughly equally spaced from each other. Thanks to this, if we take a circle from any point, we obtain an approximately equally distributed number of points. Hence, the high score of these plots for the Ripley metric.
Let's see what the best plots are if we exclude the PCA 3 features. See 8. The plots are drastically different. A lot more inline with what might have been expected. It is interesting to see that the similar behaviour is present here also, although this time coming from a different feature, namely *reduced mir*. All these plots are also a result of UMAP.

4.3 The Silhouette Measure

In 7 we can see the top 9 plots which scored highest without the Ripley metric. Since the points in the clusters are tightly packed the Silhouette intra-cluster distance is low, and the silhouette measure interprets as relatively well-clustered. It is clear see why this metric is so crucial for good-looking plots as they exhibit very clear and cleanly seperated clusters. However, they would not be useful for the intended use case. The clusters are *too* dense and would be hard to navigate. This is why the Ripley metric is important.

4.4 Conclusion

This thesis explored methods of visualizing audio files in two-dimensional space. I explored techniques for reducing the raw, high-dimensional audio data to two dimensions. These included domain-specific (Music Information Retreival) metrics which intend to represent aspects of the human perception of sound but also statistical methods for dimensionality reduction (PCA, t-SNE, UMAP). To correctly analyze which combination of methods results in a high-quality visualization, I explored methods of scoring the resulting two dimensional embeddings using various methods from spacial statistics.

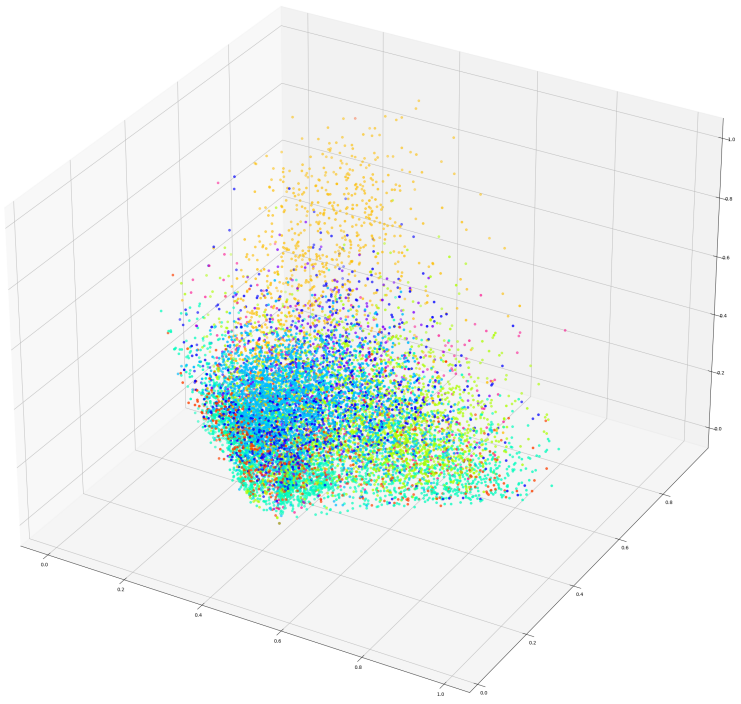


Figure 4.4: A plot of the PCA 3 MFCC feature

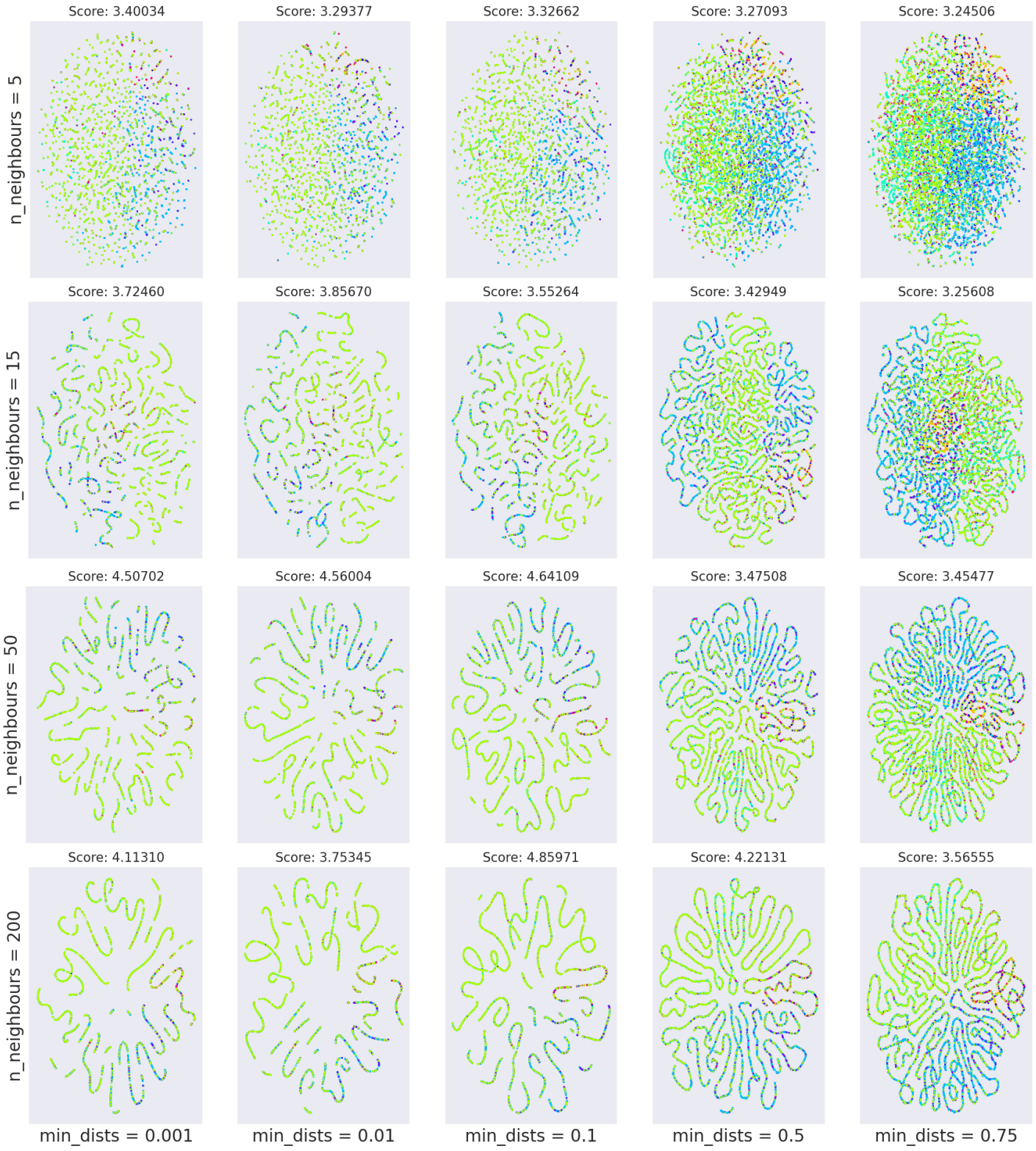


Figure 4.5: A grid search on the UMAP algorithm with the PCA 3 MFCC feature. As the value of n neighbours increases, the string-like pattern gets more pronounced.

The overall, best scoring plots 6 are not what I expected. These plots would not be useful for the intended use case, i.e. an audio file manager program. This leads me to the conclusion that more work is needed to develop an effective method of scoring.
All 9 of the highest scoring plots were some combination of STFT or MFCC as the extracted feature set, with a preliminary dimensionality reduction step of PCA to 3 dimensions, after which UMAP was applied. If we remove the plots which had a preliminary PCA dimensionality reduction step to 3 dimensions, the plots look a lot more as expected 8. The Ripley metric is also partly responsible for this result as plots with clearly separated "strings" of points score highly as seen in 7. If we remove the ripley metric altogether, the plots exhibit very dense, highly seperated clusters 9. So a combination of the preliminary reduction with PCA 3 and the way the Ripley metric evaluates plots is the reason for the unexpected results.
Besides not scoring particularly well, plots which would be intuitively useful as visualizations of audio collections were generated. For example, the best-scoring plots for the MFCC feature 4.2 (especially those for the PCA 12 MFCC and PCA 20 MFCC features) are what I would have expected to score most highly. They exhibit, homogenous clusters, while keeping the overall layout of points fairly uniformly dense. These plots seem to score high when it comes to the roundness metric, leading to the conclusion that this is an important aspect of a good plot.

4.4.1 Future work

Finding the correct metrics is crucial to correctly solving this problem. While it is probable that a combination of clustering and spatial measurements is the correct way to approach the problem, more work is still needed to find a set of metrics which would provide an optimal way of evaluating the plots in terms of both how well the timbral characteristics of the dataset are reflected (i.e how well the sounds are clustered) and how comfortable the plot would be to use as a file browser (shape and density distribution).
In addition to this even more feature extraction methods such as wavenet autoencoders and methods from the field of MIR could be tested. There are also many more dimensionality reduction methods to experiment with.

Bibliography

[1] sklearn metrics silhouette score.

[2] Rachel M Bittner, Magdalena Fuentes, David Rubinstein, Andreas Jansson, Keunwoo Choi, and Thor Kell. mirdata: Software for reproducible usage of datasets. In *International Society for Music Information Retrieval (ISMIR) Conference*, 2019.

[3] S. Chakroborty, A. Roy, and G. Saha. Fusion of a complementary feature set with mfcc for improved closed set text-independent speaker identification. *2006 IEEE International Conference on Industrial Technology*, pages 387–390, 2006.

[4] S. Davis and P. Mermelstein. Comparison of parametric representations for monosyllabic word recognition in continuously spoken sentences. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 28(4):357–366, 1980.

[5] Jeremiah Deng, Christian Simmermacher, and Stephen Cranefield. A study on feature analysis for musical instrument classification. *IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 38:429–38, 05 2008.

[6] Stephane Dupont, Thierry Ravet, Cecile Picard-Limpens, and Christian Frisson. Nonlinear dimensionality reduction approaches applied to music and textural sounds. *2013 IEEE International Conference on Multimedia and Expo (ICME)*, Jul 2013.

[7] Leon Fedden. Comparative audio analysis with wavenet, mfccs, umap, t-sne and pca, 2017.

[8] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. 15, 06 2003.

[9] Yotam Mann Kyle McDonald, Manny Tan. The infinite drum machine. 2016.

[10] Wentian Li, Jane E. Cerise, Yaning Yang, and Henry Han. Application of t-sne to human genetic data. *Journal of Bioinformatics and Computational Biology*, 15(04):1750017, 2017. PMID: 28718343.

[11] Vincent Lostanlen, Carmine-Emanuele Cella, Rachel Bittner, and Slim Essid. Medley-solos-DB: a cross-collection dataset for musical instrument recognition. September 2019.

[12] Brian McFee, Vincent Lostanlen, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Jack Mason, Dan Ellis, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, Keunwoo Choi, viktorandreevichmorozov, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao Wei, nullmightybofo, Darío Hereñú, Fabian-Robert Stöter, Pius Friesch, Adam Weiss, Matt Vollrath, and Taewoon Kim. librosa/librosa: 0.8.0. July 2020.

[13] L. Pols. Spectral analysis and identification of dutch vowels in monosyllabic words. 1977.

[14] Popper Polsby, Daniel. The third criterion: Compactness as a procedural safeguard against partisan gerrymandering. *Yale Law & policy Review 9.2 (1991): 301-353.*, 9:2579–2605, 2008.

[15] Karthikeya Racharla, Vineet Kumar, Chaudhari Bhushan Jayant, Ankit Khairkar, and Paturu Harish. Predominant musical instrument classification based on spectral features. *2020 7th International Conference on Signal Processing and Integrated Networks (SPIN)*, Feb 2020.

[16] Lamtharn (Hanoi) Hantrakul & Avneesh Sarwate. Klustr: A tool for dimensionality reduction and visualization of large audio datasets. `https://github.com/lamtharnhantrakul/klustr`, 2017.

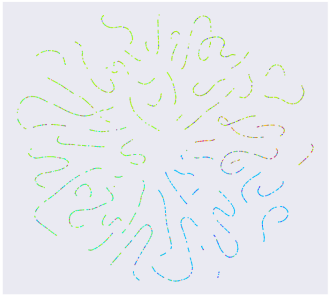
[17] Julius Smith. *Spectral Audio Signal Processing.* 01 2008.

[18] L.J.P. van der Maaten and G.E. Hinton. Visualizing high-dimensional data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

Appendices

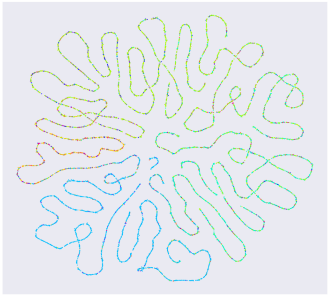
Feature: pca 3 mfcc
DimRed: umap(n_neighbors=30, min_dist=0.01, metric=correlation)

Normalized Convex_hull_overlap : 0.26966
Normalized Ripley : 2.00000
Normalized Roundness : 0.98454
Normalized Silhouette : 1.43345
Normalized Total : 4.68765



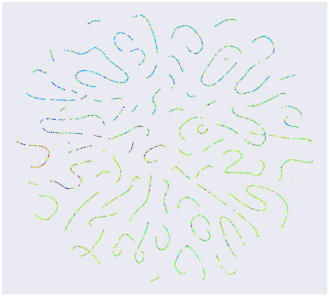
Feature: pca 3 mfcc
DimRed: umap(n_neighbors=200, min_dist=0.5, metric=correlation)

Normalized Convex_hull_overlap : 0.26608
Normalized Ripley : 1.93482
Normalized Roundness : 0.98419
Normalized Silhouette : 1.43585
Normalized Total : 4.62094



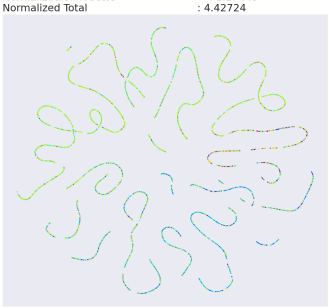
Feature: pca 3 stft
DimRed: umap(n_neighbors=30, min_dist=0.1, metric=correlation)

Normalized Convex_hull_overlap : 0.25260
Normalized Ripley : 1.92736
Normalized Roundness : 0.99240
Normalized Silhouette : 1.27336
Normalized Total : 4.44572



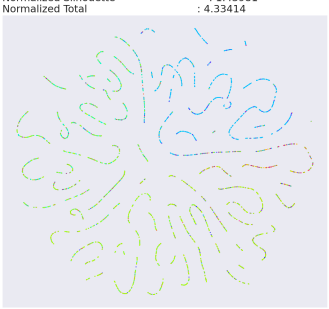
Feature: pca 3 stft
DimRed: umap(n_neighbors=200, min_dist=0.1, metric=correlation)

Normalized Convex_hull_overlap : 0.25110
Normalized Ripley : 1.91933
Normalized Roundness : 0.98237
Normalized Silhouette : 1.27443
Normalized Total : 4.42724



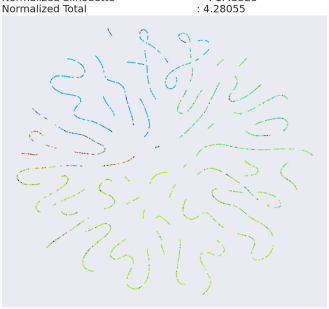
Feature: pca 3 mfcc
DimRed: umap(n_neighbors=30, min_dist=0.001, metric=correlation)

Normalized Convex_hull_overlap : 0.26757
Normalized Ripley : 1.64501
Normalized Roundness : 0.98175
Normalized Silhouette : 1.43981
Normalized Total : 4.33414



Feature: pca 3 mfcc
DimRed: umap(n_neighbors=50, min_dist=0.01, metric=correlation)

Normalized Convex_hull_overlap : 0.27472
Normalized Ripley : 1.58487
Normalized Roundness : 0.98571
Normalized Silhouette : 1.43525
Normalized Total : 4.28055

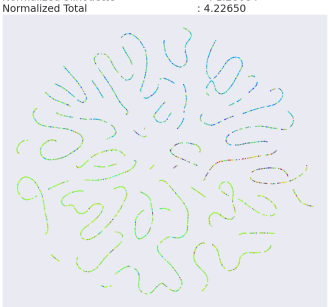


Feature: pca 3 stft
DimRed: umap(n_neighbors=100, min_dist=0.001, metric=correlation)

Normalized Convex_hull_overlap : 0.24961
Normalized Ripley : 1.72732
Normalized Roundness : 0.98935
Normalized Silhouette : 1.28258
Normalized Total : 4.24886

Feature: pca 3 stft
DimRed: umap(n_neighbors=50, min_dist=0.1, metric=correlation)

Normalized Convex_hull_overlap : 0.24804
Normalized Ripley : 1.70835
Normalized Roundness : 0.98929
Normalized Silhouette : 1.28084
Normalized Total : 4.22650



Feature: pca 3 stft
DimRed: umap(n_neighbors=30, min_dist=0.01, metric=correlation)

Normalized Convex_hull_overlap : 0.25200
Normalized Ripley : 1.70654
Normalized Roundness : 0.97589
Normalized Silhouette : 1.27753
Normalized Total : 4.21196

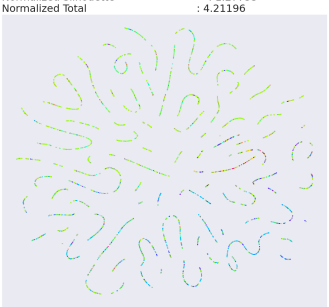


Figure 6: The top 9 scoring plots overall

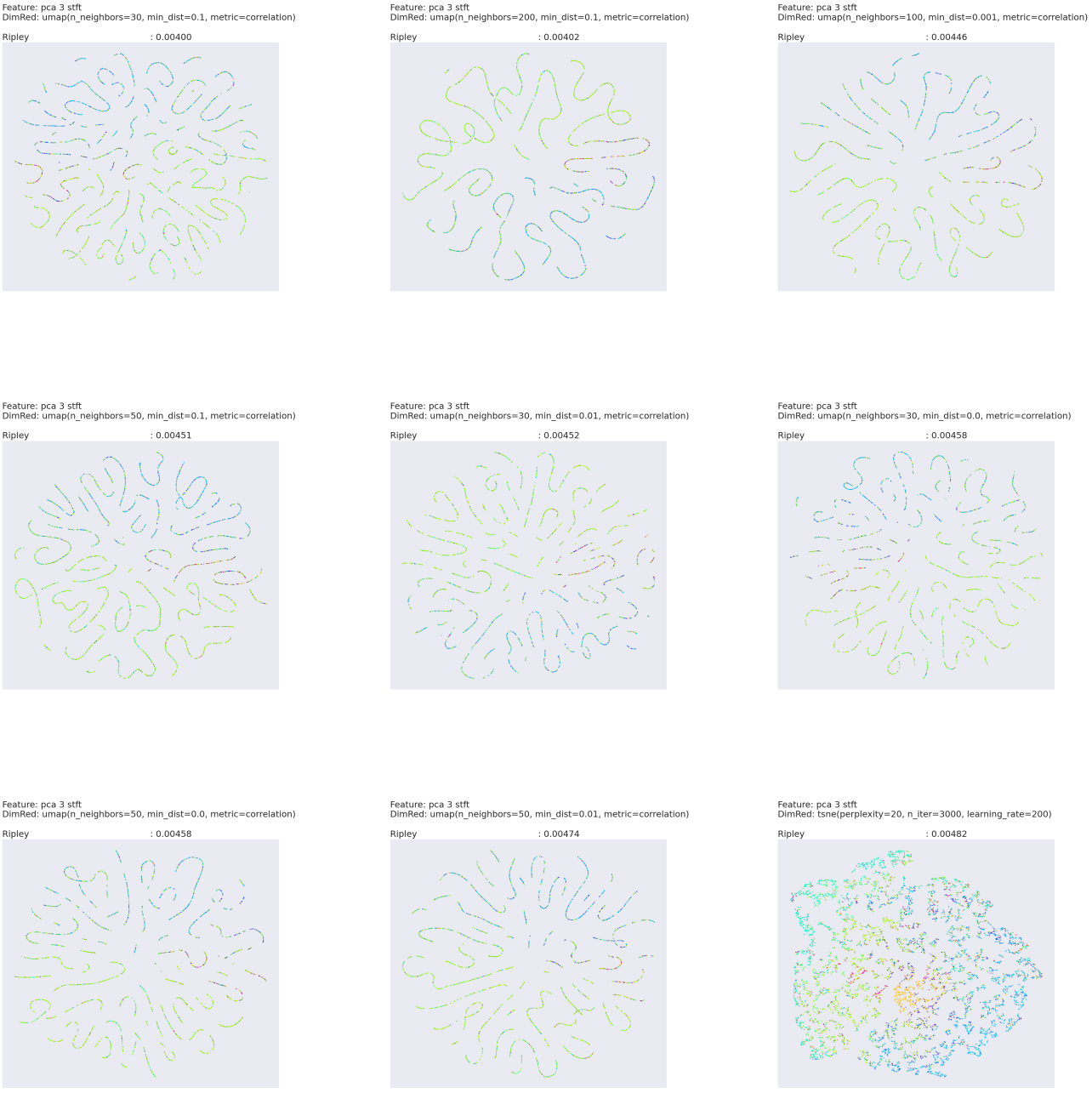


Figure 7: Top 9 scoring plots according to the Ripley measure

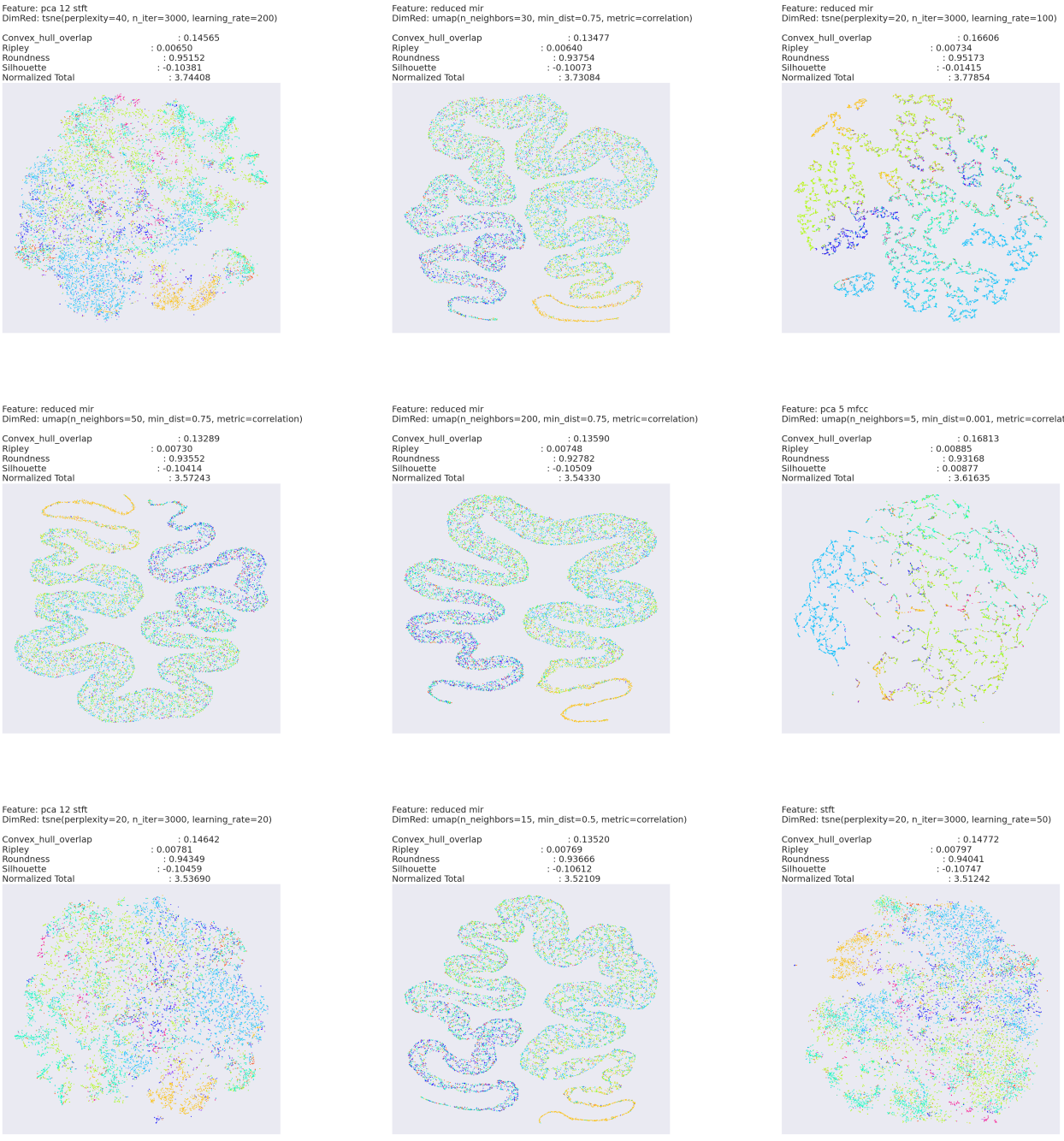


Figure 8: Top 9 scoring plots excluding the PCA 3 features

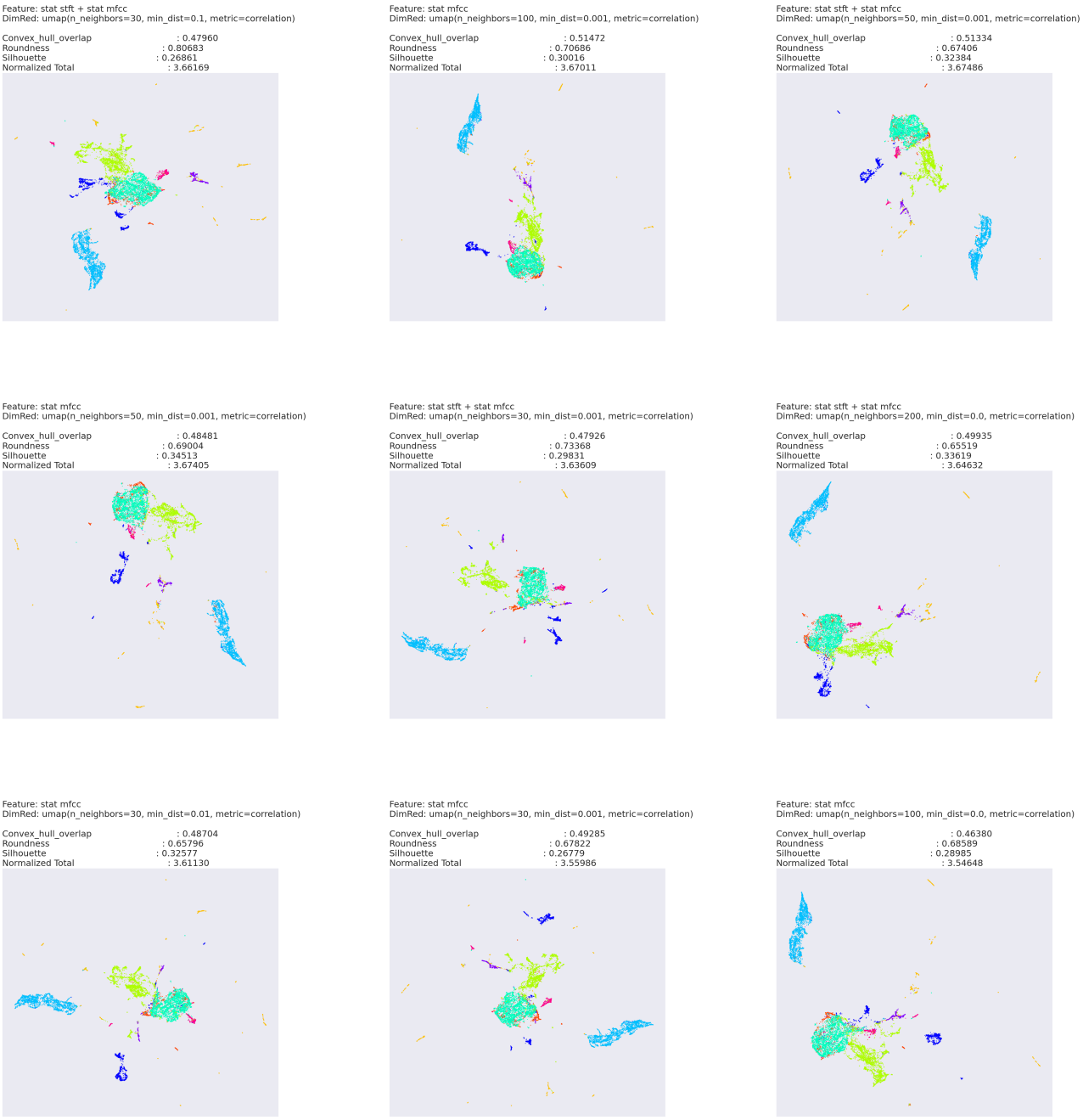


Figure 9: Top 9 scoring plots for metrics without the Ripley measure

Source Code

https://github.com/luk-pio/audioviz