CARL VON OSSIETZKY universität OLDENBURG

Informationsverarbeitung und Kommunikation: Project Description - Sample Classification for Electronic Dance Music

Bastian Bechtold
Mat.Nr: 2195907

Manuel Kaletta
Mat.Nr: 2076097

Carl von Ossietzky Universität Oldenburg
SS 2013

Oldenburg, October 18, 2013

Contents

1 Introduction

Electronic Dance Music (EDM) is usually built from electronic sound sources such as synthesizers or drum machines. In the last decade, more and more software products such as Digital Audio Workstations (DAW) and software instruments have been used as well. One of the most important sound generating procedures is sampling. In this technique small audio snippets that are played back to generate a sound.
These samples could be base elements such as single snare hits, or whole rhythmic loops, or even complex drum arrangements. In this project, we assume that electronic dance music is comprised only of samples. Thus, complex musical productions should be representable as superpositions of many short samples. We liken these samples to words in speech in that a song is made of samples much like a sentence is made of words.
During the production of electronic dance music, it is often interesting to find similar samples to a known one. The process of choosing these samples can be very complex. We created an algorithm to aid in this selection process. The ideal case would be to find similar samples to all samples used in a track.
A conceptual framework for this would be a three step system: First, an existing collection of samples is divided into different classes. Second, short parts of a musical track are classified according to these classes. For typical tracks, there will be no clear classification, but more likely a superposition of different probabilities for each short part. Lastly, these classifications can be used to match the musical track back to the samples, thus re-synthesizing the music. In order to do this, the classes do not need to correspond to musically relevant classes such as bass drums or string instruments.
In this project, we are doing some basic research on this topic. We create new classes from a sample database using a clustering algorithm, and then classify test samples from these classes. Also, we classified our sample database by hand, and then classify test samples to these classes.

2 Preprocessing of the Samples

In order to make samples more comparable, all samples are preprocessed. First, the multi-channel samples are panned to mono. Since there are no spatial features used in the project, no stereo information is needed. To remove the stereo information the arithmetic mean in between the left and right channel is calculated. This is the mid signal of the stereo signal

$$x_{\text{mid}}[k] = \frac{1}{2}\left(x_{\text{left}}[k] + x_{\text{right}}[k]\right).$$

Second, silence at the beginning and end of each sample is cut off. All signal below a threshold level at -75 dBFS is detected as silence

$$x_{\text{sil}}[k] = \begin{cases} 0 & \text{if} \quad 20 \cdot \lg(x_{\text{mid}}[a]) < -75 \;\forall\; a \leq k \\ 0 & \text{if} \quad 20 \cdot \lg(x_{\text{mid}}[a]) < -75 \;\forall\; a \geq k \\ x_{\text{mid}}[k] & \text{else} \end{cases}$$

Lastly, all samples were amplified to have an root mean square (RMS) of 1 in order to normalize the energy of the samples

$$x_{\text{pre}}[k] = \frac{x_{\text{sil}}[k]}{\sqrt{\overline{x^2}}},$$

$\overline{x^2}$ is the mean of the squared signal.

3 Features

Each sample is split into windowed and overlapping blocks. Different features are calculated on each block. The blocks have a length of 20 ms, 50 % overlap and are Hann windowed.

Features used are the RMS, peak, crest factor, spectral centroid, logarithmic spectral centroid, spectral variance, spectral skewness, spectral flatness, spectral brightness and mean spectral absolute slope. They are described in detail as follows.
For every feature, the block is given as x, $x[k]$ is a frame within the block, K is the number of frames within one block. The FFT of the block is X, $X[n]$ is an FFT bin, $2N$ is the FFT-Length. The features are calculated as follows:

RMS: The RMS is a measure for the energy of the block.

$$\mathrm{RMS} = \sqrt{\frac{1}{K}\sum_{k=0}^{K-1} x[k]^2}$$

Peak: The peak is simply the maximum of all absolute values within the block.

$$\mathrm{PK} = \max|x[k]|$$

Crest Factor: The ratio of the peak to the RMS value is the crest factor.

$$\mathrm{CF} = \frac{\mathrm{PK}}{\mathrm{RMS}}$$

Spectral Centroid: The spectral centroid is the normalized frequency weighted mean of the absolute spectrum. The frequency is normalized to the sampling frequency.

$$\mathrm{SC} = \sum_{n=0}^{N} |X[n]| \cdot \frac{n}{2N}$$

Logarithmic Spectral Centroid: To account for human frequency perception, the spectral centroid is calculated relative to a logarithmic frequency axis as well.

$$\mathrm{LSC} = \sum_{n=0}^{N} |X[n]| \cdot \log\frac{n}{2N} + 1$$

Spectral Variance: The spectral variance gives a value of the change of the absolute spectrum over all frequencies.

$$\mathrm{SV} = \sum_{n=0}^{\frac{K}{2}} (|X[n]| - \overline{|X[n]|})^2$$

with the spectral mean

$$\overline{|X[n]|} = \frac{1}{N+1}\sum_{n=0}^{N} |X[n]|.$$

Spectral Skewness: The skewness of the absolute spectrum shows how much its distribution leans towards high or low frequencies.

$$\mathrm{SS} = \sum_{n=0}^{\frac{K}{2}} (|X[n]| - \overline{|X[n]|})^3.$$

Spectral Flatness: Given the spectral mean $\overline{|X[n]|}$ and the geometric spectral mean

$$\overline{|X[n]|}_{\mathrm{geom}} = \sqrt[N+1]{\prod_{n=0}^{N} |X[n]|},$$

the spectral flatness is calculated as the ratio of them.

$$\frac{\overline{|X[n]|}_{\text{geom}}}{\overline{|X[n]|}}$$

Spectral Brightness: The spectral brightness is the ratio of high to the low frequency energy. To get the energy of low and high frequency, the weighted sum of the absolute spectrum is taken. The weighting of the high frequency bins is given as

$$\begin{aligned} w_{\text{high}} &= 0.5 - \frac{\cos(\omega)}{2} \\ w_{\text{low}} &= \frac{\cos(\omega)}{2} + 0.5 \end{aligned}$$

with ω as N logarithmic ordered values in the range $[0;\ \pi]$. The point of intersection between the low and high frequency weighting was set to approximately 2 kHz.

$$\text{SB} = \frac{\sum_{n=0}^{\frac{K}{2}} |X[n]| \cdot w_{\text{high}}}{\sum_{n=0}^{\frac{K}{2}} |X[n]| \cdot w_{\text{low}}}$$

Mean Absolute Spectral Slope: The mean of the absolute spectral slope gives the amount of change in the spectrum over the frequency. It is calculated by

$$\text{MASS} = \frac{1}{N} \sum_{n=1}^{N} ||X[n]| - |X[n-1]||$$

The features of all blocks of all samples are saved within a Pandas DataFrame[1]. This makes them easily searchable and saveable.

4 Principal Component Analysis

To reduce the dimensionality and increase the variance, a Principal Component Analysis (PCA) is used. The PCA transforms the data into a new set of dimensions with a minimum amount of correlation. This new feature space contains the dimensions, which are responsible for most of the variance within the data. Hence the covariance is maximized. The covariance matrix of a vector x of length n is given as

$$\mathbf{S} = \frac{1}{N} \sum_{i=1}^{n} (\mathbf{x}_i - \overline{\mathbf{x}})(\mathbf{x}_i - \overline{\mathbf{x}})^{\mathsf{T}}$$

with $\overline{\mathbf{x}}$ as mean value of the vector. A PCA resulting in one dimension would always result in the eigenvector $\mathbf{u}_1$ with the largest eigenvalue λ_1 of the covariance matrix

$$\mathbf{S}\mathbf{u_1} = \lambda_1 \mathbf{u}_1.$$

The eigenvector gives then the transformation of the PCA. More dimensions could be calculated by defining Iteratively the eigenvectors. For higher dimensions also more efficient algorithms are known. Details on the theory can be found in [Bis06, p. 561]. Using the PCA[2], we reduced our features to $d = 5$ dimensions.

To get a data point for one sample, the PCA of all the features in the n blocks can be seen

[1] http://pandas.pydata.org/

[2] Using sklearn.decomposition.PCA from scikit-learn (http://scikit-learn.org/)

as one long feature vector of length $d \cdot n$. Consequently, the features of each sample can be considered as a point in a $d \cdot n$ dimensional space.
All feature data and PCA-reduced feature data are saved as Pandas DataFrames. This reduces the whole data set from about 560 Mb of wave files to a 60 Mb HD5 database. Calculating this takes about 5 minutes on a modern computer[3].

5 Distance

In order to measure the similarity of different samples, the Dynamic Time Warping (DTW) algorithm was used. DTW compares sequences of feature vectors to one another, even if the sequences are of different lengths. This is important, since the samples in our sample base have different lengths. DTW stretches and compresses sample feature vector sequences for maximum similarity. It can thus find similarities in samples that are time-stretched versions of one another or that contain each other.
DTW first compares every feature vector of one sample to every feature vector of another sample by calculating the Euclidean distance between each feature vector. This creates a cost matrix C of distances between every block of the first sample to every block of the second sample. This matrix is of dimension $N \times M$, where N and M are the lengths of the feature sequences of both samples, respectively.
Then, DTW searches for the cheapest path through C. In order to not evaluate every possible path through this matrix, DTW only calculates the cheapest path for every positive time step from $C_{n-1,m}, C_{n,m-1}, C_{n-1,m-1}$ to $C_{n,m}$. The final distance between the samples is then calculated by adding all the steps $C_{n.m}$ on the cheapest paths from $C_{0,0}$ to $C_{N,M}$. Additionally, every step is multiplied by $\frac{1}{N}$ or $\frac{1}{M}$ or $||(\frac{1}{N}, \frac{1}{M})||$ to correct for the stepping distance through the cost matrix.
This algorithm was implemented in Python, but it was too slow for practical comparisons of big sample sets. Thus, we further implemented it in C and called that version from Python, which provided two orders of magnitude of speedup. Still, calculating all the distances between all the samples in the sample database takes about two hours on a modern computer[4]. This distance matrix was saved as Pandas DataFrame to a HD5 database of 60 Mb.

6 Modified Mini Batch k-Means Algorithm

The k-Means algorithm takes k random points in space representing the center of k classes. These class centers are then refined in two steps: First, each data point in a set of training data is classified to belong to one of the classes by selecting the closest class center. Second, the class centers are moved towards the geometric center of all the data points classified as belonging to the class. These steps are repeated until the class centers stop moving.
However, this approach is impractical for large data sets, since every iteration needs to calculate distances from every data point to every class center. A modification of this algorithm called Mini-batch k-Means [Elk03] uses only a random subset of all data points for each iteration. This results in slightly less accurate class centers, but much greater performance.
In our case though, there is no Euclidean feature space in which we could position class centers. The only measure available is the DTW-distance between the different samples. Thus, we modified the second step of the Mini-batch k-Means algorithm to choose the center-most sample instead of the arithmetic center. The center-most sample is defined as the one sample in the class that has the least distance from all other samples in the class.
Running this algorithm on our data set results in k class centers. New samples can then be classified to belong to one of these centers by selecting the closest one.
Calculating class centers requires the calculation of many distances between many samples. In order to speed up this process, we pre-calculated all the sample distances as described in chapter 5.

[3] The multiprocessing module in Python seemed to have a problem with Windows, thus forcing the Windows version to use only one core, which would quadruple the runtime.
[4] Four times longer on Windows

7 k-Nearest-Neighbors Algorithm

The k-Nearest-Neighbors (k-NN) algorithm performs a distribution free classification task. It uses a lazy learning approach which means that all the training data is saved. The Algorithm searches that data base for the k closest data points to the point to classify. The distance measurement depends on the application. The class which is most often found in those k nearest neighbors is the one the sample is classified as [AD07, p. 338 f.].
The space of the data points is the PCA-reduced feature space described in section 3. The distance used in our project is the DTW-distance described in section 5. The feature space version of a test sample is calculated and compared to all other feature vectors in the database from section 3. This comparison table is then sorted by distance using the sorting algorithm built into Python. As output of the k-NN classification, the ratio of found neighbors to the total number of neighbors is given for the k nearest neighbors. This can be interpreted as the probability of a point belonging to a class.

8 Measurements and Results

A test sample base was created with ten tracks for each class. All of the test samples were new samples that were not included in the training sample base.

8.1 Modified Mini Batch k-Means

The k-Means algorithm was used to calculate $k = 10$ class centers in our sample base using a batch size of 500 points and 50 iterations. These classes were used to classify our test sample base. The tags of the test samples within each of the 10 class clusters created by the k-Means algorithm were counted. The results are shown in figure 1.
It can be seen that the algorithmically calculated classes do not correspond well to the tags defined by our sample base. In eight different measurements the percentage of right classified samples was 23 % in the mean, minimum 8.3 % and maximum 33.3 %. Such modest results were to be expected. Nevertheless, it seems that there are some consistent mappings in the structuring of the tags. This needs to be verified with further research.

8.2 k-Nearest-Neighbors

All the samples in the test sample bank were classified using the implemented k-NN algorithm with $k = 10$. The results are shown in figure 2. The tags of the test bank are plotted against the tags of the classification.
The results show that many of the samples are classified correctly. The percentage of correct classified samples is 48.3 %. This even exceeds our expectations. The variations in the classification correlate with the variances between samples of the same tag in the sample base. This means that the classes that are badly classified are also hard to classify by human perception.

9 Conclusion

A classification of audio samples was performed using two approaches. The results of the k-Means clustering show that algorithmically calculated classes differ strongly from human classification. The k-NN classification of samples based directly on human perception showed very good results. Consequently, the task of assigning samples to a given musical track is possible and further research on this has the potential to strongly improve the classification.
One big area of open research are the features. More or better features should give a much better differentiation of the samples. In particular, the given features represent pitch only very crudely. Informal listening tests confirm that the algorithm tends to ignore pitch differences.
Also, a listening test for measuring the perceived distance between samples could be done. This

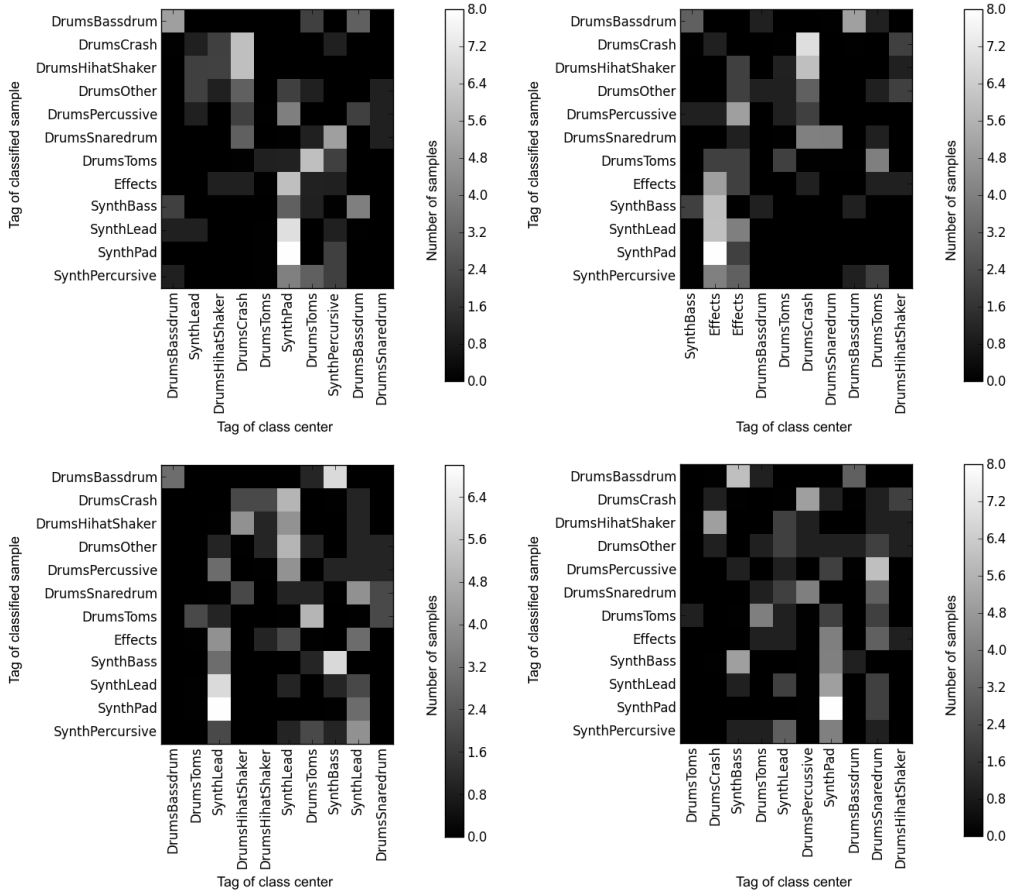

Figure 1: Number of samples, depending on their tag and the tag of the class center they were assigned to. The 10 classes were generated by a modified mini batch k-Means algorithm with a batch size of 500 using 50 iterations. The four plots represent four different runs of the modified mini batch k-Means algorithm.

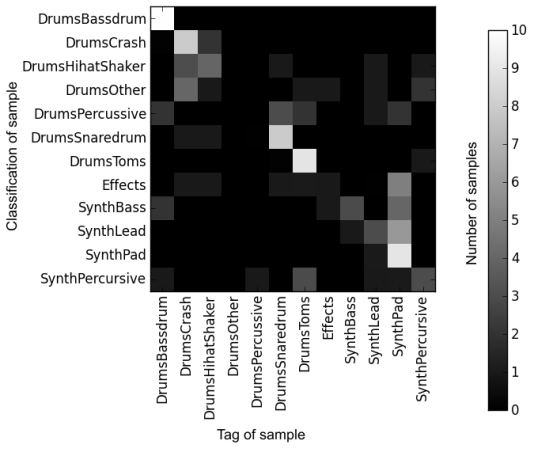

Figure 2: Number of Samples, depending on their tag and the assigned tag by the k-NN algorithm with $k = 10$.

could then be used to check how the perceived differences correlate with the DTW-distances the algorithm calculates.

References

[AD07] Mauricio Orozco Alzate and German Castellanos Dominguez. Nearest feature rules and dissimilarity representations for face recognition problems. In Kresimir Delac and Mislav Grgic, editors, *Face Recognition*, pages 337–357. InTech, 2007.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[Elk03] Charles Elkan. Using the triangle inequality to accelerate k-means, 2003.