MusicMapper: Interactive 2D representations of music samples for in-browser remixing and exploration

Ethan Benjamin
Columbia University
Department of Computer Science
eb2947@columbia.edu

Jaan Altosaar
Princeton University
Department of Physics
altosaar@princeton.edu

ABSTRACT

Much of the challenge and appeal in remixing music comes from manipulating samples. Typically, identifying distinct samples of a song requires expertise in music production software. Additionally, it is difficult to visualize similarities and differences between all samples of a song simultaneously and use this to select samples.

MusicMapper is a web application that allows nonexpert users to find and visualize distinctive samples from a song without any manual intervention, and enables remixing by having users play back clusterings of such samples. This is accomplished by splitting audio from the Soundcloud API into appropriately-sized spectrograms, and applying the t-SNE algorithm to visualize these spectrograms in two dimensions. Next, we apply k-means to guide the user's eye toward related clusters and set $k = 26$ to enable playback of the clusters by pressing keys on an ordinary keyboard. We present the source code[1] and a demo video[2] of the MusicMapper web application that can be run in most modern browsers.

Author Keywords

Machine Learning, t-SNE, JavaScript, Remix, Visualization

1. INTRODUCTION

Currently, one needs to be a domain expert in music production to select appropriate samples and piece them together creatively in aesthetically pleasing ways. A typical production pipeline involves selecting a song, choosing interesting parts from it (the samples), and combining these to form a remix or new musical composition. By combining the detection of similar parts of songs and letting users visualize and play back these similar clusters, and doing all of this in-browser, we hope to reduce the barrier of entry to experiencing the fun of remixing and visualizing music. Furthermore, every song analyzed and visualized using this method effectively becomes a new 'instrument,' whose similarly-clustered parts can be deconstructed and remixed at will.

There exist automated approaches to sample identification in music [5] but there is little work on the inverse question: what parts of a given song should be sampled? Our intuition for this is that nearby mel-spectrograms of chunks of songs should sound similar enough to combine them as samples in a remix.

There is also previous work related to visualizing samples for music production purposes using supervised visualization techniques [2] and the visualization of loop libraries [1]. Here, we use an unsupervised technique from machine learning, t-distributed Stochastic Neighbor Embedding (t-SNE) [3] for visualizing samples. The t-SNE algorithm is ideal for taking high dimensional features (such as spectogram data) and reducing them to a low dimensional mapping which faithfully portrays intra-data differences. Using t-SNE, we are able to show all samples of a song as a two dimensional scatterplot layout.

By employing such techniques and exploiting modern JavaScript optimization methods in web browsers, we hope to reduce the activation energy required on a user's part to start remixing and visualizing songs. Rather than open audio production software, load audio, select samples manually, map them to MIDI notes, then play them back, a user need only go to the web address of the MusicMapper web application, select a song, wait for convergence of the algorithm, and begin exploring and remixing.

2. ALGORITHM

Our algorithm has three primary phases. First, we divide the input song into evenly sized chunks. We then calculate features for each song chunk (Subsection 2.1). We input these features into the t-SNE algorithm and use a heuristic for adequate subjective convergence (Subsection 2.2). Finally, we perform k-means clustering on the final t-SNE output to visualize each cluster by its convex hull (Subsection 2.3).

2.1 Feature Selection

When given a song from an MP3 file or the Soundcloud API, we extract an array of its raw audio samples and divide this array into evenly sized "chunks" of of 16394 samples. Exploring different chunk sizes automatically is noted as a direction for future work.

We then perform a discrete Fourier transform on each chunk to get a frequency spectogram. Because typical implementations of the discrete fourier transform perform optimally with arrays whose length is a power of 2, we opted for chunks whose length is the power of 2 closest to 0.3 seconds (resulting in 16384 samples per chunk for the most common sampling rate of 44,100 Hz).

Our algorithm converts Fourier magnitudes to histograms of a perceptually based mel scale. We use O'Shaughnessy's formula [4] to convert frequency f to mel m, $m = 1127 \log_e(1 + f/700)$. Buckets in this histogram correspond to equal ranges of mel scale frequencies. For each bucket, we aggregate the magnitudes of all raw frequencies that belong to that

[1]Code: `https://github.com/fatsmcgee/MusicMappr`
[2]Demo video: `http://youtu.be/mvD6e1uiO8k`

NIME'15, May 31-June 3, 2015, Louisiana State Univ., Baton Rouge, LA.

range. For instance, consider the bucket that contains mel scale frequencies from 320 to 352. Mel 320 is equivalent to 229.85 Hz and mel 352 is equivalent to 256.63 Hz. Therefore this bucket will equal the summation of magnitudes in the Fourier spectrums from 229 to 256 Hz.

We found that using 100 mel histogram buckets worked better than raw magnitudes from the discrete Fourier transform in subjectively distinguishing song chunks. As an added benefit, the reduction from thousands of Fourier spectrum values to 100 buckets allowed t-SNE to converge considerably faster (as t-SNE computes intra-feature euclidean distances at each iteration).

In experiments, we also found that these features conveyed richer information than mel-frequency cepstrum coefficients, which tended to conform closely to a one dimensional structure in their low-dimensional embedding. When t-SNE processed MFCCs in two dimensional space, the resulting embedding tended to be a single curve in which distance along the curve was proportional to pitch.

2.2 t-SNE and Convergence

After computing mel histogram data for each song chunk, we feed these features into t-SNE.

t-SNE finds an optimal embedding through gradient descent and updates its solution on each iteration. During this process, we visualize the current t-SNE solution in two dimensions, stretching it to fill the browser window.

Determining good convergence criteria for t-SNE proved difficult. Ultimately, we developed heuristics that mirrored a user's subjective perception of convergence in the animated t-SNE solutions. Specifically, we track the 2-D points and quantify how much each point has moved from iteration to iteration, scaling by the extent of all 2-D points in the solution. Because t-SNE's solution can change rapidly from iteration to iteration but remains stable if it has moved slowly at a high enough number of iterations, we ensure that solutions are stable for a certain number of steps. We found that ensuring stability for 50 step increments worked well in practice.

2.3 Clustering

After t-SNE has converged, we mark clusters in the data and allow the user to interact with each cluster. To reveal additional structure in the data and allow interaction with related groupings of song chunks, we run k-means clustering on the final t-SNE solution, with $k = 26$ (each cluster corresponding to one alphabet letter on a standard keyboard). We denote each cluster in our visualization by drawing curves which match their convex hulls. The graham scan algorithm is used to calculate each convex hull.

We mark the center of each cluster with its corresponding alphabet letter. Pressing that letter on a user's keyboard will play through chunks corresponding to that cluster in order of their appearance in the original song. Users can simultaneously play different clusters at the same time, as they would on a sampler or drum machine.

3. IMPLEMENTATION

We chose to implement MusicMapper as a static web application using HTML, CSS, and JavaScript with no server backend, as shown in Figure 1. This allowed us to avoid server costs and easily deploy the application, and take advantage of the rich layout and visualization features available in modern web development.

Though JavaScript is not a typical target for machine learning and numerical applications, we were able to find several libraries needed by our algorithm. In particular, we found that existing libraries for the discrete Fourier transform, t-SNE, k-means clustering, and graham scan were suitable. We used D3.js to visualize t-SNE solutions, including clusters and their convex hulls. To extract raw audio data from song files, we used the WebAudio API.

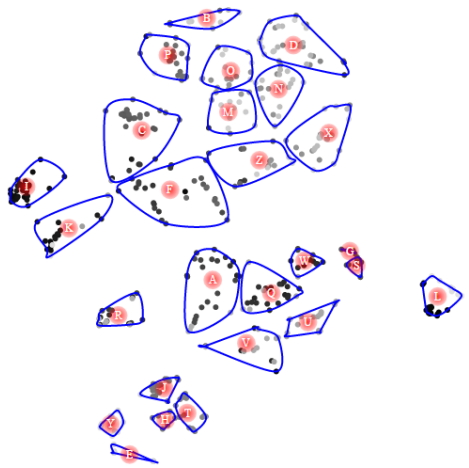

Figure 1: Final solution after convergence and clustering of "One (Your Name) Feat. Pharrell" by Swedish House Mafia following spectrogram generation, t-SNE, and k-means. Pressing keys on the user's keyboard will play through the samples in the cluster corresponding to the label, enabling in-browser remixing.

3.1 JavaScript Optimization

Feature computation, requiring Fourier transforms for every chunk of a song, proved to be the most time-consuming portion of our application. To improve runtime, we used WebWorkers to exploit multiple cores and fully utilize the CPU. Because features can be calculated independently for each chunk, we simply divided Fourier calculation among multiple workers.

4. CONCLUSION

We presented MusicMapper, an interactive online tool for allowing nonexperts to visualize and play back samples of songs. This was accomplished through a pipeline of breaking songs up into chunks, computing their mel-spectrograms, using the t-SNE algorithm for visualization, and k-means clustering for visualizing similar chunks. The code is open sourced at `https://github.com/fatsmcgee/MusicMappr`, and a demo video is at `http://youtu.be/mvD6e1uiO8k`.

5. REFERENCES

[1] S. Dupont, T. Dubuisson, C. Frisson, R. Sebbe, and J. Urbain. Audiocycle: Browsing musical loop libraries. *Content-Based Multimedia Indexing*, 2009.

[2] O. Fried, Z. Jin, R. Oda, and A. Finkelstein. AudioQuilt: 2D Arrangements of Audio Samples using Metric Learning and Kernelized Sorting. *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 281–286, 2014.

[3] L. V. D. Maaten and G. Hinton. Visualizing Data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[4] D. O'Shaughnessy. *Speech Communications: Human and Machine (Addison-Wesley Series in Electrical Engineering)*. Addison-Wesley, 1987.

[5] J. Van Balen, J. Serrà, and M. Haro. Sample identification in hip-hop music. *Music and Emotions*, pages 301–312, 2013.