GENERATION OF MUSIC USING LSTM AND FEATURE ANALYSIS BY CLUSTERING

Ashwin Ravishankar, Satishchandra Subbarao Mula
aravisha@gmu.edu, smula2@gmu.edu
Computer Science Department,
George Mason University,

Introduction

People for a long time now, have expressed their emotions via music. Understanding the significance of music, for our project, we originally decided to work on developing a model that can learn the underlying patterns in songs composed by an artist or band and generate new music sequences for previously unseen lyrics of the same artist. But, given the complexity of the problem and our inability to articulate correct assumptions, the idea is still left to be perfected.

The thought of analyzing music stuck strong, as there are numerous real-world applications of music information retrieval[1] recommender systems, instrument recognition, genre classification, music transcription and music generation. Exploring the smaller parts that make up a song, the 2 main components are vocal and instruments. Ignoring the vocal component of the song, we modified our goal to experiment an approach to generate new musical sequences of notes and chords that make up the music. In a developing world that nurtures several musicians who start at zero, there is a need for self-help kick starter systems that can help these budding artists learn and compose.

Analyzing the generated music to understand the underlying components that make it up gives us an insight into the music itself. These insights can help us understand the usefulness of music data, development and application of computational approaches, tools to compose better music[2]. Here, we aim to identify different classes of musical instruments that contribute to the music generated.

Satish worked on building the model that learned from a dataset to generate music. Ashwin developed the unsupervised learning algorithm to cluster the musical instruments contained in the generated music.

Background

Our base paper[3] discussed about using neural networks for prediction and generation of polyphonic music following the musical rules.They used probabilistic model using sequences of notes as an input and learnt the likelihood of the appearance of subsequent notes, which were later used to generate new music. To test the hypothesis proposed in the paper we decided to work on music generation using probabilistic models and LSTM networks.

Related work in the domain of music information retrieval has gone into employing machine learning techniques to develop a classifier for automatic instrument recognition given a mono-track audio recording[4]. The classifier is trained on monophonic audio files containing a single note of the instrument. The other classification approaches[5] extracts features of individual instruments to recognise them in monophonic test files. Though monophonic music analysis stands to be a solved problem, there havent been major breakthroughs in polyphonic music analysis.

Approach

In order to ease the task at hand, we broke it up into two components. First step was generation of music and the next was to analyse the generated music. The problem with music generation is to learn the patterns or combinations of notes and chords that constitute to the music. This is important because random sequences can break the laws of music, making it very unpleasant to hear. Since traditional Neural Networks are bad at remembering the information over a period of execution, we decided to use Recurrent Neural Networks specially LSTM to avoid long term dependency.

The objective here is to take the input audio files and learn about notes and chords information for each file. Run it over LSTM to train a model and generate new notes and chords using the generated model and some probabilistic methods. Although we are not proposing a new technique, we are implementing the approach as discussed in background work, in order to learn more about the implementation of LSTM and its effects over music generation.

As a baby step into polyphonic music analysis we propose an approach to cluster different classes of musical instruments contributing to the generated music.. Here, we analyse the experimental output of three clustering algorithms - KMeans, Affinity Propagation and DBSCAN (Density Based Spatial Clustering of Applications with Noise) run on the same input audio file and plot the results. As the generated audio files sound rather unpleasant, given that there are no real evaluation heuristics to test the quality of generated music, we aim to gain some insights into the instruments present in the file.

An audio signal has several means of representations. In this project, audio is considered as vibrations, to be a function of time. The basic representation of audio in time domain is represented as a waveform[6]. The envelope of a signal wave is the curve that approximates the amplitude extremes of a waveform over time. Envelops are modelled as ADSR - attack(sound builds up with noise component), decay(sound stabilizes to reach steady pattern), sustain(sound is constant), release(fading away of sound). The approach we used is to first extract features, two at a time, from the audio signal. On the extracted features, the clustering algorithms are run and the results are observed.

Experimental Design

The programming language we used is Python. Libraries include the use Keras[7] that runs on the top of tensorflow in order to train and build our models. For data preprocessing and generation we used Music21[8][9] an open source tool kit, to extract notes and chords information from the input midi files and when generating, binds together the predicted sequences to output midi audio files. Librosa[10] is a python package that was used for audio analysis. It also provides the building blocks for music information retrieval. The other python libraries used are: Matplotlib for visualizations, SciKit-Learn for the clustering algorithms and Numpy.

In music generation for data preprocessing we took 40 input midi audio files[11] consisting of files in classical genre. Music21 is used to get information of notes and chords from the midi file. Any music is made up of notes and chords. Notes stores the information about the frequency of the sound. Notes ranges from A(highest note) to G(lowest note). Chords contains information about the combination of notes that are being played together. All the obtained note sequences are mapped to orchestrated integer values, which would make it easier to pass them to the training phase.

For Training the model, we use 3-Layered LSTM with 64 units each which provides the next layer with note sequences. We used 3 dropout layers with fraction of 0.5 where dropout layer sets a fraction of inputs to 0 at each update, preventing overfitting[12], 2 Dense layers with first one connecting to 256 output units and the second one connecting to output units same as that of total length of notes. The final layer we used is the activation layer using softmax activation function over all the output notes. We trained our model using model.compile function[13] for 50 epochs and calculated the loss using categorical cross entropy.

The final part was generation of notes from the trained model. In order to do that we took a random set of initial note sequence and passed it to the trained model, and using the trained weights we predicted the most probable sequence that would occur after the current note, using model.predict function[13]. We have limited the notes generated to 300 with the gap of 0.5 between each and later re mapped these integer notes to corresponding string values, to generate back the midi audio file using Music21[8]. The generated audio is roughly over a minute.

The generated audio file is transformed to time domain as a temporal array of floating point values using librosa. This time series is sampled at default sampling rate of 22050 Hz per 60 seconds. Onset[14] is that event when the musical note hits a peak (attack phase). In the time series, onset is computed for the entire length of audio to obtain the onset sequence of frames. These sequences as expected, aligns with the highest amplitude of vibration of that note (Appendix, Fig 4). Some of the features extracted for every onset from the audio are:

Zero Crossing Rate
Indicates the number of times that a signal crosses the horizontal axis[15] (Appendix, Fig 5).

Mel Frequency Cepstral Coefficients (MFCC)
The small set of features that concisely describe the overall shape of a spectral envelope (Appendix, Fig 6). It is used to describe timbre.

Energy and Root Mean Square Energy (Appendix, Fig 7)
Energy of a signal corresponds to how loud the signal is, or the magnitude of the signal. Total energy is defined as

$$\sum_n |x(n)|^2 \tag{1}$$

Root mean square energy is defined as

$$\sqrt{\frac{1}{N}\sum_n |x(n)|^2} \tag{2}$$

Timbre It is that quality of music that distinguishes the tone of different musical instruments despite constant pitch and loudness.

Every onset window is a segment of size - ten percent of sampling rate, starting at the onset frame. For a pair of features, the feature values are normalized to a constant scale (-1 to 1) using scikit learn (Appendix, Fig 8). Different clustering algorithms are modelled on these feature pairs to obtain cluster plots.

KMeans
In this algorithm, K centroids are assigned randomly in the space of the data, centroids being the center of mass for each of the cluster. In our case, the K averaged at 2. Once the centroids are assigned, the following algorithm is iterated until the centroids stop changing: Assign each sample to its nearest centroid Update centroids as the center of mass of the samples assigned to that cluster. The centroids update stops when the algorithm converges (maybe to local minima).

Affinity Propagation
Unlike KMeans, this algorithm does not require an input of the number of clusters. Algorithm proceeds by alternating two iterative steps: Responsibility matrix R has values r(i, k) that quantify how suited sample k is to serve as exemplar for sample i, relative to other candidate exemplars for sample i Availability matrix A contains values a(i, k) that represent how appropriate it would be for sample i to pick sample k as its exemplar taking into account other points preference for sample k as an exemplar[16]

DBSCAN
Two input parameters (eps) and the minimum number of points required to form dense region (minPts) are considered. Find the points in eps neighbourhood of every point and identify the core points with more than minPts neighbors. Find the connected components to core points on the neighbor graph, ignoring all non core points. Assign each non core point to nearby cluster if cluster is an eps neighbor, otherwise it is an outlier. [17]

Results

The figure (Appendix, Fig 1) is the waveform of sample input from the training data and (Appendix, Fig 2) is the waveform of the generated audio file. The input audio clearly follows the song structure and it is more spread and has bigger intervals, differentiating the chorus and verse. But analyzing the output audio we found that it had no structure leaving no gaps for verse and chorus.

Working on the generated music, the feature pairs are plotted (Appendix, Fig 8) before running the clustering algorithm to have a sense of the spread of sample data.

KMeans
Implementing KMeans algorithm returns two separable cluster classes A and B, each representing a class of musical instrument (Appendix, Fig 9).

Affinity Propogation
This algorithm ran on the same audio file, clusters the feature space into 9 groups with individual center of mass (Appendix, Fig 10). We are unable to draw any conclusive result form this algorithm as to why there are multiple clusters identified.

DBSCAN
Running dbscan algorithm on the audio file gave results similar to KMeans. Two clusters were identified, which are representing a musical instrument in feature space. (Appendix, Fig 11)

As we have modelled an unsupervised learning algorithm to cluster unlabelled data, we are unable to identify class labels as to which class represents which instrument. But, with more input data, and features that can critically label the instrument being played, we can evaluate that as well.

Future Work

One way of improving current work would be finding some evaluation function that would evaluate the quality of the generated music. For classification find a real time classifier model that classifies the instruments being played in the generated music.

References

[1] Music Information Retrival (https://en.wikipedia.org/wiki/Music_information_retrieval)

[2] NYU Music Information Retrieval (http://www.nyu.edu/classes/bello/MIR_files/1-Introduction.pdf)

[3] Generating Music Using LSTM Network (https://arxiv.org/ftp/arxiv/papers/1804/1804.07300.pdf)

[4] Lewis Guignard and Greg Kehoe. Learning Instrument Identification (http://cs229.stanford.edu/proj2015/010_report.pdf)

[5] KLMS: Music instrument classification (https://github.com/hash2430/music_instrument_classification)

[6] Music information retrieval (https://musicinformationretrieval.com)

[7] Keras (https://keras.io/)

[8] Music21 (http://web.mit.edu/music21/)

[9] MIDI Processing(https://github.com/Skuldur/Classical-Piano-Composer)

[10] Librosa (http://librosa.github.io/librosa/)

[11] MIDI World (http://www.midiworld.com/search/?q=classic)

[12] Keras Core Layers(https://keras.io/layers/core/)

[13] Keras Models(https://keras.io/models/model/)

[14] Onset (https://en.wikipedia.org/wiki/Onset_(audio))

[15] Zero Crossing Rate (https://librosa.github.io/librosa/generated/librosa.core.zero_crossings.html)

[16] Affinity Propogation (https://en.wikipedia.org/wiki/Affinity_propagation)

[17] DBSCAN (https://en.wikipedia.org/wiki/DBSCAN)

Appendix

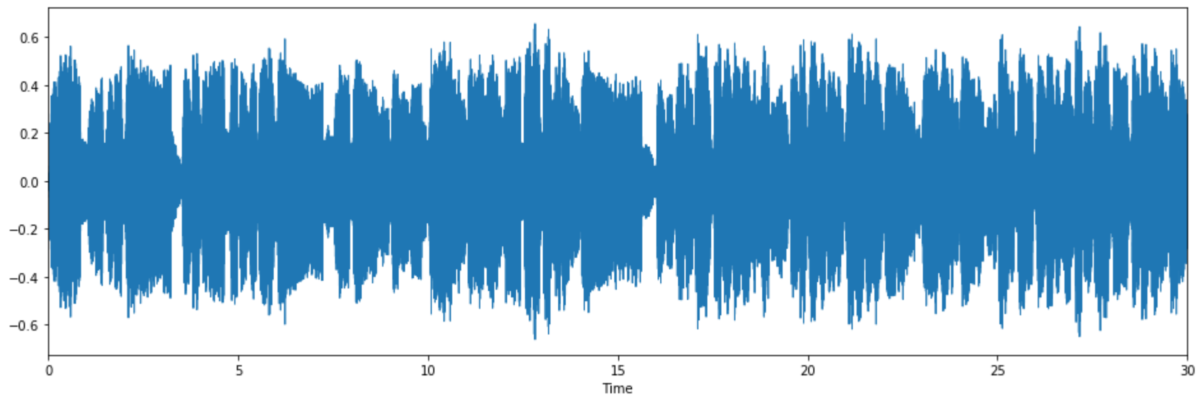

Figure 1: Input Wave Form

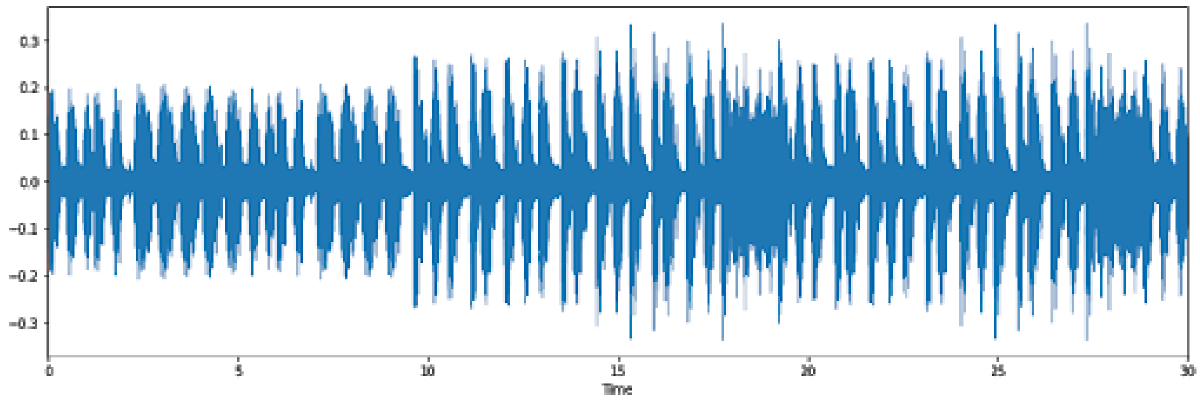

Figure 2: Generated Audio Wave Form

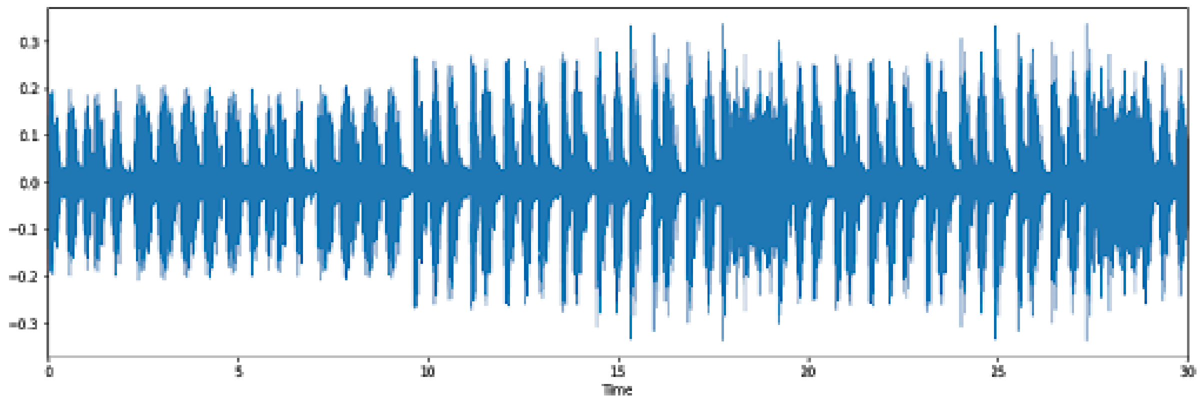

Figure 3: Music Analysis Input Wave Form

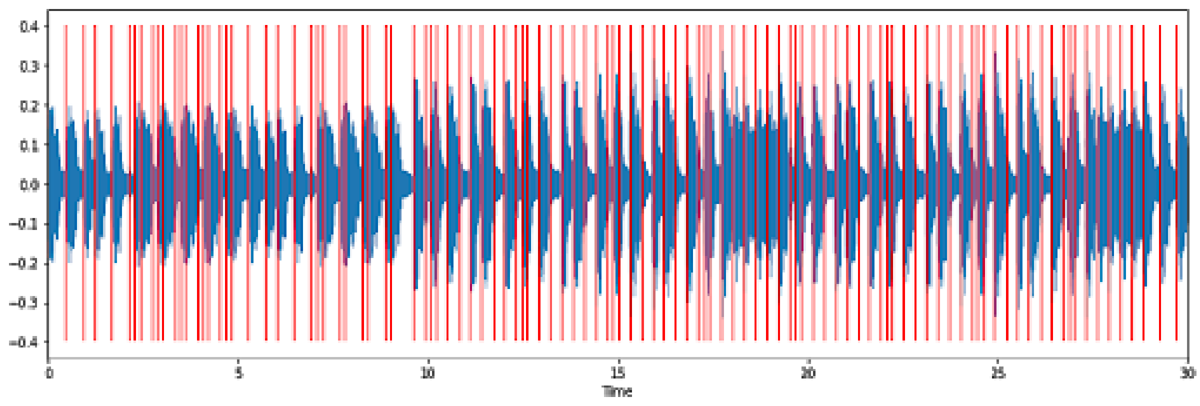

Figure 4: Onset Sequence

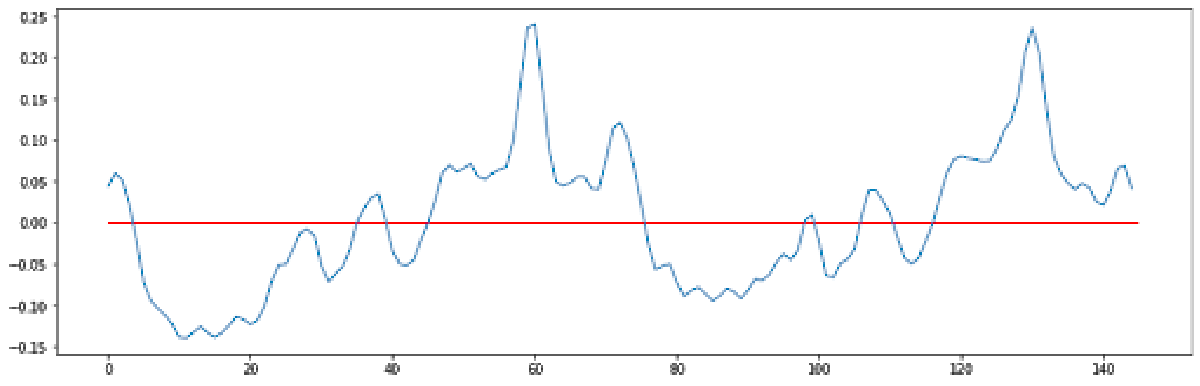

Figure 5: Zero Crossing Rate

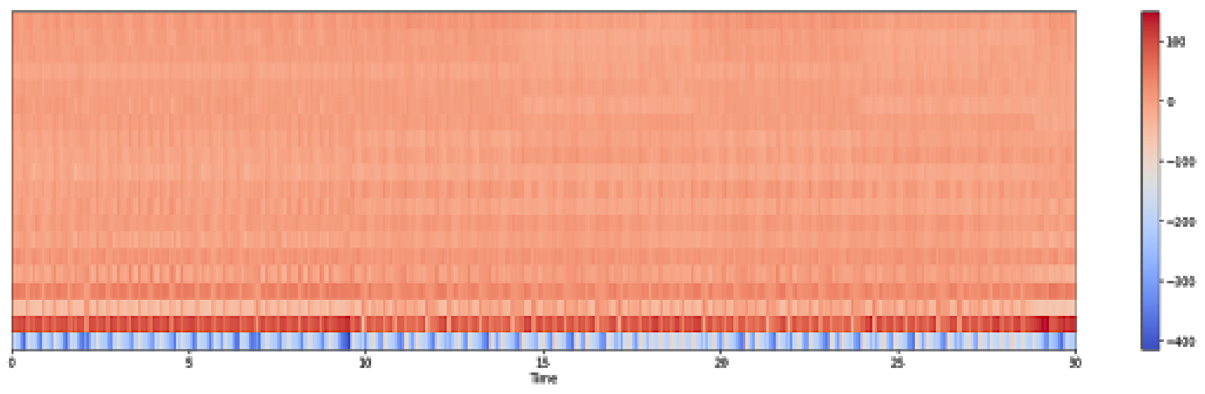

Figure 6: Spectral Wave - MFCC

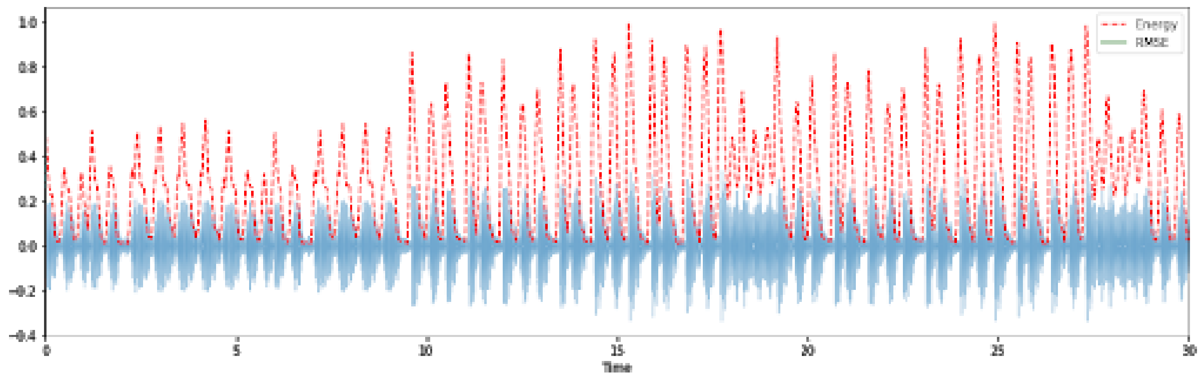

Figure 7: Energy

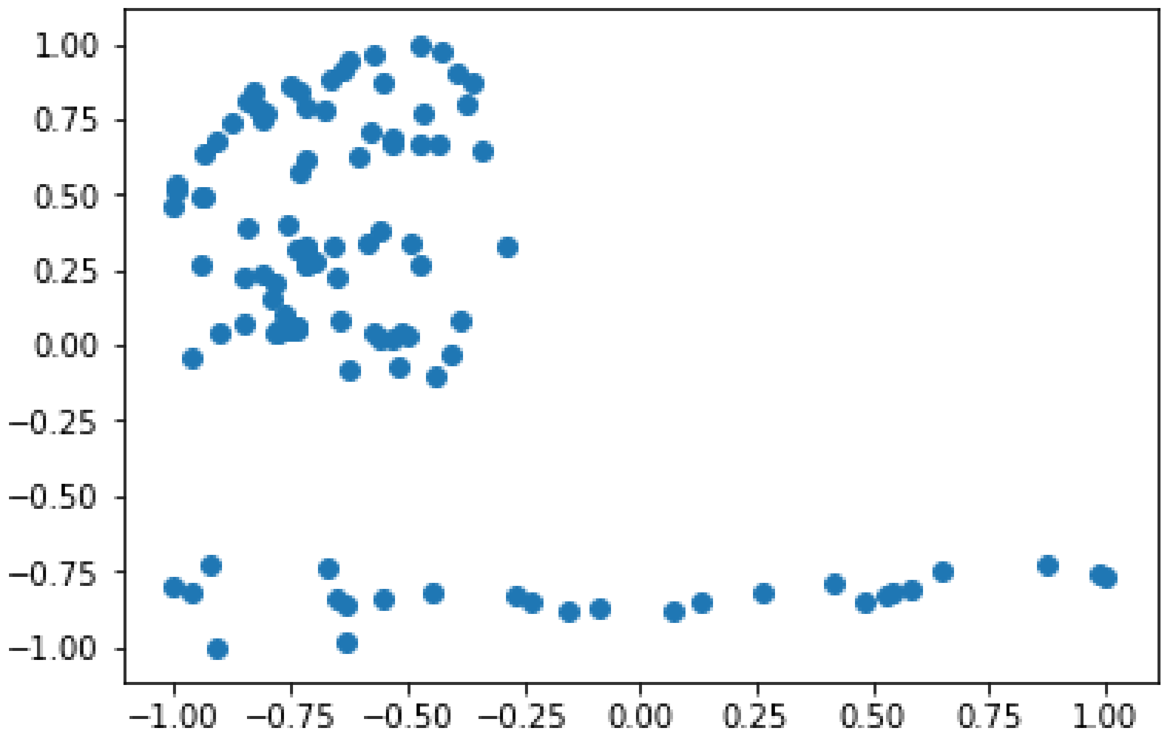

Figure 8: Plot Before Clustering

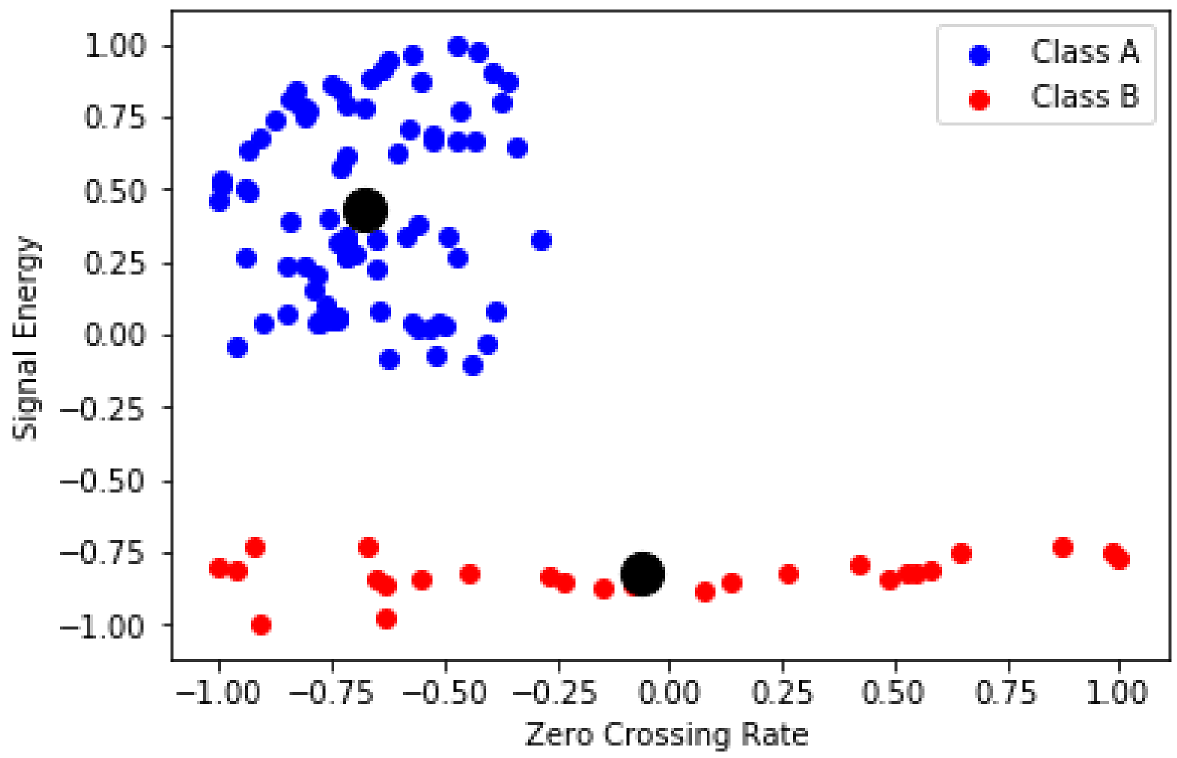

Figure 9: K-Means Clustering (K=2)

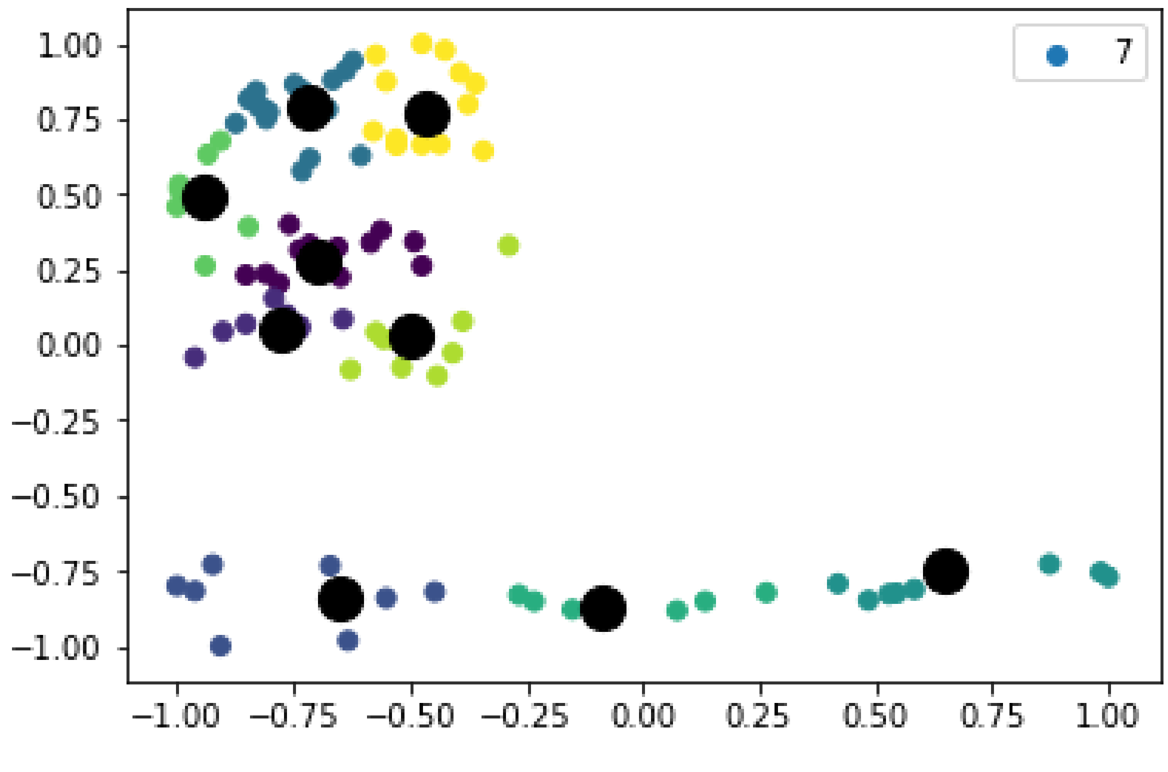

Figure 10: Affinity Propagation Clustering

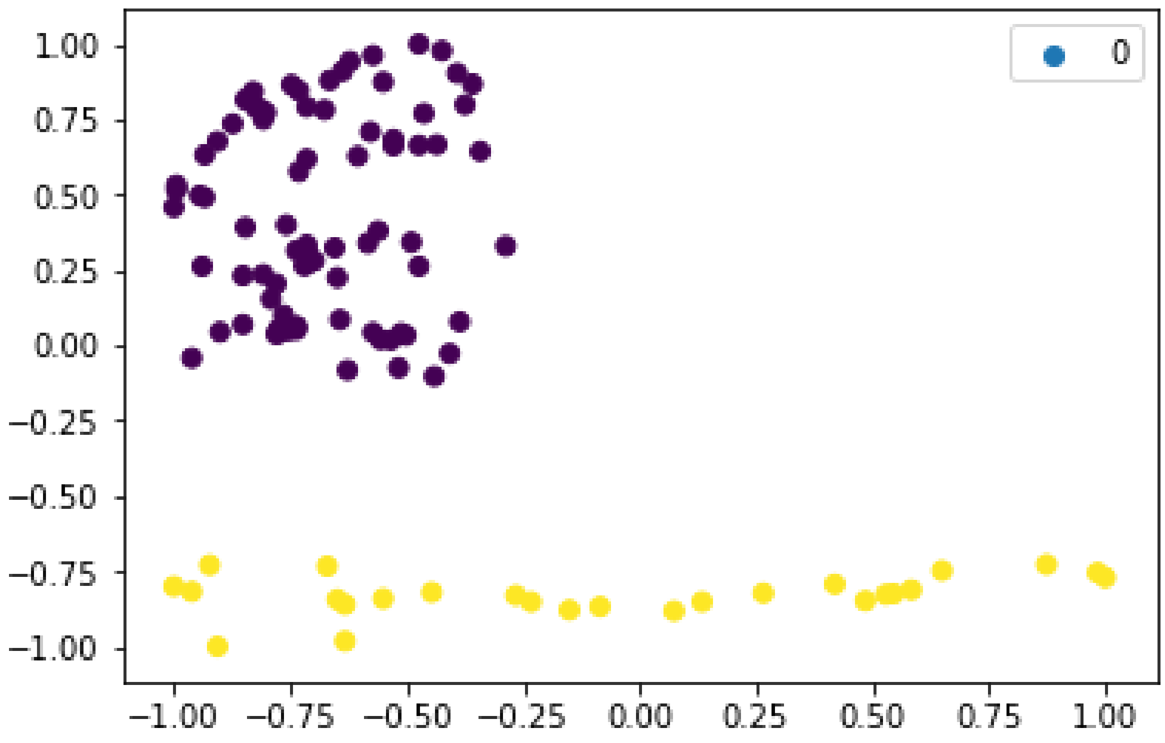

Figure 11: DBSCAN Clustering