Unsupervised, Auto-Weighted Audio Clustering using Automatic Feature Selection

Jacob Elliot Reske*
Department of Music
Yale University
New Haven, Connecticut 06511 USA

May 2, 2014

Abstract **– Fully unsupervised audio clustering by musical features (e.g. timbre, pitch, and rhythm) presents a number of problems. Selecting essential criteria for musical feature extraction, preprocessing the data, and finding similarities in the results introduces typical machine learning problems of overfitting and high dimensionality. The benefits of creating a fully unsupervised clustering algorithm, however, are clear. Fully unsupervised clustering could find unlikely matches in disparate musical styles, as well as a method to process large (or very new) datasets from streaming sources. Music researchers could use this tool to observe similarities in new, uncategorized, or recently digitized music (i.e. without ID3 tags). In this paper, we propose a toolkit that combines a robust musical feature selector and an auto-k, auto-weighted k-means clustering algorithm.**

Index **– unsupervised clustering, k-means, feature selection, Gaussian mixture models, musical instrument recognition, Minkowski Distance, genre clustering.**

1 Introduction

This is a test of the LaTex typesetter system. We use *this* to write words in italics. We use **this** to type words in bold. We use `this` to set in typewriter font, but this is not usually how you type computer code.

2 Feature Extraction

The core component of any audio analysis method, the feature extraction phase attempts to quantify audio features as meaningful streams of analytical data. A wealth of literature on audio feature extraction already exists, detailing methods to extract information about timbre, pitch, rhythm, and energy of a given audio file. The issue, becomes one of curation: the features extracted must be both general enough to apply across a number of different musical styles and specific enough to convey meaningful information. Our feature extraction tool is a modified version of Yaafe, a Python library that can process and output 20 distinct features. [3] Our implementation can make use of any of them, but we will focus on three features whose traits prove salient in the genre problem: Mel-frequency cepstrum coefficients (MFCC), octave band signal intensity (OBSI), and amplitude modulation (AM).

2.1 Mel-frequency cepstrum coefficients

Loosely defined, MFCCs attempt to approximate the timbral activity of a given sound or set of sounds. They attempt to represent an audio spectrum as a series of discrete frequency blocks a cepstral representation each with their own spectral properties. In an MFCC transformation, each cepstrum is mapped onto the mel scale, a pitch scale designed to space spectral information so that, perceptually, pitches are spaced equally apart. Our algorithm for converting hertz to mels uses the formula:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.[22]$$

The general technique for MFCC computation is as follows: first, the given audio signal is partitioned into windowed segments of width w (specified by the sample size). For each frame, we apply a Fast Fourier Transform. The signal is then converted to a Mel-warped spectrum using X, creating a modified time-

*Advisors: Ian Quinn, Brian Kane, Dept. of Music, Yale University

frequency window on the mel scale. A triangular filterbank is then applied directly on the resulting signal. Since they are uniformly distributed on the mel scale, each triangular windows is comparable from file to file. [10] We then apply a Disrete Cosine Transformation (DCT) to the resulting log of each windows output to get an MFCC vector for that window. We use other techniques, such as spectral mean subtraction, to normalize the vector at this stage. The result is a spectral vector, and its amplitude is the MFCC value for that time and mel-frequency window. A summary of this process is provided below. [PICTURE] At this point, we can also use each MFCC coefficient matrix to create a new matrix of first and second time derivatives, and the lower rows of the new matrix are trimmed to give this new matrix the same shape as the original.

Mel-frequency cepstrum coefficients are very popular when timbral extraction is needed, and much has been written about general MFCCs utility in speech recognition systems, instrument recognition, and genre classification. However, until recently, MFCCs were considered invariant to other extra-timbral features, such as tempo and key. In their paper, Genre classification and the invariance of MFCC features to Key and Tempo, Tom Li and Antoni Chan suggest that MFCC data does incidentally encode tempo and key information, and that songs performed in atypical keys/tempi were less accurately classified than those with typical traits. Whats more, their findings suggest that genre classifiers, like the genres that they model, are in fact influenced by the fundamental keys of the instruments involved, despite the common intuition that key does not influence genre. [14] We will explore this postulation in our own results section (X).

2.2 Octave band signal intensity

While MFCCs encode timbral information by splitting the spectrum into discrete time-frequency bands, there is merit in capturing timbral information in other ways. Octave band signal intensity (OBSI) attempts to describe the power distribution of an instrument or set of instruments by splitting it into octave sub-bands.

The process is similar to the MFCC computation. For each time step, We partition the subset of the spectrum that contains normal musical notes (A0 at 27.5Hz to A8 at 7040.00Hz) into octaves using a triangular filterbank. Each octave band is calculated by multiplying frequencies in the usual manner, using the An frequencies as edges. We use triangular filters to force different distributions for two instruments that inhabit the same octave band. We then calculate the log of the energy spectral density in this octave band, and define the energy spectral density by the formula:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.[21]$$

As with MFCCs, the frequency band is standardized by octaves, so the columns of the resulting matrix are comparable from file to file. We also can compute the log ratio of the first bands energy with each subsequent band to get a matrix of OBSI ratios (labeled OBSIR). The general algorithm for OBSI feature extraction is described in [10]. While not as useful for more timbrally complex pieces of music, this feature is especially useful for encoding the differences in power spectrum between instruments or a group of instruments. Below, we reproduce the graph of octave band signal intensity found in 6, comparing the spectrums of an alto sax and Bb clarinet.

2.3 Amplitude Modulation

Amplitude modulation attempts to encode two values: the average pitch displacement given constant amplitude (tremolo) and the average amplitude displacement given constant pitch (grain). [9] For each spectral fame, a modulation of 4 8 Hz in tremolo and 10 40 Hz for grain are recorded an analyzed. For each range, the maximum energy is found, and the difference between this and the mean energy in this range is computed. [3] The product of these two values represents the amplitude modulation in this time-frequency range.

3 Preprocessing: KL Divergence and normalization

3.1 Mean and variance/covariance pairs

As described in 2, each feature extraction method in the process produces a matrix in R2 with a high dimension. Since each feature is processed by time step (in our algorithm, the default step size is 256 samples) and thus is composed of n sparse vectors, the resulting matrices are unsuitable for direct comparison. The MFCC extractor, for example, returns a m x 13 matrix, where m is 860 for a 5:00 minute file.

To remedy this, a sample mean and variance-covariance pair is computed for each set of vectors. We use the standard formula for computing variance-covariance: the mean vector preserves the average value for each column (in the MFCC case, the log spectral partition). The variance-covariance matrix preserves the variances of variables i in position (xi, xi), and the covariance between values i, j in position (xi, xj). The formula for covariance is given as:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.$$

and the formula for the mean vector is intuitive. This computation achieves two goals: it standardizes the dimensionality of each files feature matrix, and it reduces dimensionality so computation is manageable. Computation time for later steps is reduced by limiting the number of compares required. The resulting matrices are stored in the current clusters FeatureSet object for later use.

3.2 KL-Divergence

Next, we need a reliable method to represent the distance between two mean/covariance pairs. Any number of norms could be used on the covariance pairs [LEN(X Y)], or any number of Li norms (I e 1 infinity) (SEE PAGE BELOW), but the mean vector would be disregarded. We therefore consider feature vectors x1, x2 as multivariate normal (Gaussian) distributions modeled by means u1, u2 and covariance matrices sigma1, sigma2, and compute the Kullback-Leibler divergence between distributions. This represents the difference between normal distributions X1 and X2 [6]; Information Theory terms this information gain (entropy) between a two probability distributions, one of which is true or accurate. In terms of our .mp3s MFCC vectors, this represents the difference in probability distribution between a given songs calculated timbre and one that is close in timbral space. A KL-divergence test between an .mp3s MFCC feature vector and the same .mp3, then, should return a distance of 0. The formula for KL-Divergence between distributions X1 and X2 is given as:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.$$

and, in terms of our normal distributions X1 and X2 and mean/covariance pairs u1/u2 and sigma1, sigma2:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.$$

Taken to an arbitrary power to preserve accuracy, this gives us an integer that represents a distance between a true song and another, as represented by probability distributions X1 and X2. However, unlike most distance calculations, KL-Divergence is asymmetric; Dkl(X1 —— X2) != Dkl(X2 —— X1). Multiple methods of overcoming this (e.g. calculating resistor-average distance) [12] we implement a simple solution by defining a (necessarily) Symmetric KL-Divergence:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.[17]$$

We compute DSk,l for all files in our FeatureSets manifest object (a list of all .mp3s to be processed). To speed our later clustering step, a general nxn Divergence matrix (DIV) is created for each feature for N .mp3 files, where DIV(xi, xj) = DSk,l(xi, xj). This matrix should always adhere to a few requirements: DIV(xi, xi) = 0 FORALL i e N (the distance in feature space between an .mp3 and its duplicate should be zero), and DIV(xi, xj) = DIV(xj, xi) (the Symmetric KL-Divergence should be symmetric). A simple check function is implemented at the end of the DIV matrix calculation to issue warnings and recompute if these requirements are not met. The resulting matrix is then stored in our FeatureSet object, and the process is done for every feature in FeatureList. If more features are introduced later in the processed and a DIV matrix was not computed, a new matrix will be added and recomputed.

3.3 Normalization

One additional (optional) step is added here before clustering can begin: normalization across dimensions. Because each audio feature is independent and operates on different scales, the relative mean values of these feature vectors can vary drastically from feature to feature. The MFCCs average distance values, for example, might be several powers greater than other features distances (such as AM or OBSI), skewing our clustering algorithm naturally in favor of the formers computed distances. To remedy this, we normalize each DIV matrix by dividing each entry by that matrixs infinity-norm, or the maximum of each of the row sums. [23] This normalization method is inherently rough, but it suits our purposes by scaling each features distances to be comparable.

3.4 Summary

To summarize: this process standardizes the feature vectors and prepares them for clustering by considering them as multivariate normal distributions. It computes the KL-Divergence (distance) values between .mp3 files and stores them for easy retrieval later. This preprocessing step drastically reduces the compute time of the clustering step, which references these distance values constantly.

4 Clustering

4.1 Basic K-means overview

Now that we have distance values between all songs in our dataset, we can begin to cluster these songs into different, dynamically-sized categories. All of the algorithms proposed are variants on the standard K-means algorithm, so it is worthwhile to review the basics of this approach. K-means clustering is widely used for both its simplicity and applicability to a large variety of use cases from small to large dataset sizes,

cluster count, and dimensionality. It is also a speedy algorithm and can be parallelized, making it optimal for our use case.

However, the K-means algorithm does not come without distinct disadvantages. The most obvious disadvantage is that a fixed number of clusters k must be specified before the algorithm runs, requiring a level of guesswork and data snooping to get the right amount. Outliers also pose a problem; in the basic implementation, no data point is left out, and a few outlier points can quickly reduce the number of usable clusters. [13] Thus, in its most basic form, K-means is not always suitable; the problem of song clustering should have different requirements and heuristics to overcome the algorithms pitfalls. We therefore propose a number of improvements to the basic K-means algorithm for our use case some of them implementations of existing research, others our original algorithms and heuristics. It is beyond the scope of this research to present an optimal variant of K-means for generalized music clustering. Our hope is to implement several variations and document their strengths and pitfalls in the context of the aforementioned genre problem.

The K-means algorithm is an evolutionary clustering algorithm that groups n objects into a specified k number of clusters. Every K-means implementation has four steps that represent the core of the algorithm. They are:

1. Given k, assign points in Rn as the centroids of k initial clusters.

2. For each point x e X and k e K, compute the distance from x to centroid k. The cluster that x belongs to is the one whose centroid is closest to k.

3. After every point x e X is assigned to a cluster, recompute the centroid of the cluster using the new elements.

4. Repeat until threshold condition is met, then return the resulting clusters. [13]

In 4), the threshold condition is usually related to the inertia of each clusters centroid how far it moves in feature space from round to round. Depending on the datasets size and sparseness, a cluster can take anywhere from 2-3 to hundreds of rounds to properly settle. Our input dataset never contains more than 600 songs, and the distance metric to centroids is implicit, so we have opted to hard-code a round maximum of 40, in the interest of simplicity. However, in our tests, we have found that even optimal k-sized clusters take significantly fewer than 40 rounds to stabilize.

4.2 Our implementation

To accommodate our data, a few basic changes had to be made for our K-means implementation. Our FeatureSet object has computed distances between every song in all dimensions, but our representation of features as Gaussians means that these songs values do not exist as points in any discernible feature space. We only have KL-Divergences that relate feature vectors to each other; this keeps us from computing an explicit n-dimensional centroid point. [17] proposes a solution: defining the centroids of each cluster implicitly, preserving the distance between centroid and point x while eschewing a concrete value for the centroid. Formally, the distance between centroid C and point y is defined as:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.$$

The distance function between xi and xj in Rn is also optimized to support a large dimension of disparate musical features. Since songs can be very metrically similar on one feature dimension but very different on another, picking a proper metric is very important. The most basic distance metric Euclidean distance FIX THIS FIX THIS FIX THIS FIX THIS and it is most effective with features that have comparable scales (such as Rn). [11] Other metrics weight dominant distances more or less prominently: the Manhattan Distance (1-noorm), for example, simply takes the sum of the respective dimensional distances. Formally, the general formula for the p-noorm is:

$$COV = 5x^{10} - 9x^9 + 77x^8 + 12x^7 + 4x^6.[16]$$

In other literature, this generalized form is called the Minkowski distance. As p increases, the metric will become naturally biased toward dimensions with dominant (greater) distances; the infinity-norm, for example, is simply the max value of all distances. Choosing a proper p-value, then, represents how much we would like clusters to form based on the dominance of a single feature versus the even distribution of all features. Our current approach allows for a p-value to be input manually for each cluster; future versions of this algorithm will include a method to calculate an optimal p-value, much like the implementation in Rs library. [24]

4.3 Heuristics

A few heuristics have been implemented to address some of K-means inherent pitfalls. The random selection of initial centroids, for example, can easily derail a clustering job if the centroids are suboptimal. If the optimal centroids are, by definition, the final centroids of the k clusters, the initial points should not be

too close to each other, relative to the size of feature space. This could unfairly exclude outlier clusters, all while dividing more central clusters along seemingly arbitrary lines. To remedy this, a distance heuristic is used to evenly distribute the initial k clusters. [26] Once k random points are picked, the distances D(ki, kj) are checked against the average distance of all points (xi, xj) e X on every dimension. If any pair of centroids is closer than the median distance on that dimension, the cluster resets and reinitializes the centroids. This initialization process can take hundreds of tries for larger datasets, but (since all these values are stored) the compute time is largely negligible. Planned for future versions is a minimum distance, so that outliers do not dominate the list of initial centroids. A more practical solution, of course, may be to prune outliers entirely (discussed below).

A second solution to this problem is implemented on the other end; to prune bad initial centroids, we implement an iterate function in our ClusterFactory object to run the same cluster job a proportion of times related to the number of n points. For each iteration, we store the final clustering in a dynamic stack to compare later. A simple sort is performed on the final clustering before the push() to make multiple results comparable, since K-means does not guarantee that clusters will be sorted the same way each time. After the cluster iterates, the stack is emptied, and the final clustering with the highest frequency over all iterations is recorded. This is considered the final, definitive clustering, removing outlier results from the solution. Instead of an automatic iteration count, a manual iteration value can be specified in the interface if the automatic value is insufficient or too computationally expensive.

4.4 Auto-weighted k-means

While the above tactics provide safeguards against outliers in both initialization and results essentially by brute force they do not address some of the limitations inherent in the K-means algorithm. As mentioned in I), a weighted implementation of K-means requires weights to be assigned beforehand. Picking optimum weights before testing based on those criteria invokes the hazard of assuming that these weights are optimum for the test set before testing has even begun. Testing a wide range of weight hypotheses and finding an making statistically inferences too quickly, on the other hand, can result in data snooping: making spurious or statistically insignificant correlations after the fact.

However, there is definite merit in using a heuristic on K-means to measure the optimum weight strategy, especially in datasets with many features. At high dimensions, n-dimensional space appears homogeneous [18]. Small correlations in one dimension (or subset of dimensions) can be outweighed by the relative homogeneity among the rest. To solve this, we consider the idea of cluster-specific weights. This stems from intuition about the nature of musical genres criteria: one genre of music may be more dependent on a specific subset of features as its defining elements. While some genres are practically defined by instrumentation the instruments in a Jazz combo, for example other genres may cluster more predictably based on a specific tempo.

To that end, we propose a set of heuristics built into the standard K-means algorithm, in an object labeled KMeansHeuristic. In addition to rounds, K-means is broken up into clusters. First, for n dimensions a list of W possible even weights are computed, where each weight $w \in W$ (the total number of even weights) is contains $0 \leq i \leq n$ nonzero weights of the same value v, and $v * i = 1$. For $n = 3$, for example, the list of subsets are $[\mathbf{0}, \mathbf{0}, \mathbf{1}], [\mathbf{0}, \mathbf{1}, \mathbf{0}], [\mathbf{1}, \mathbf{0}, \mathbf{0}], [.\mathbf{5}, .\mathbf{5}, \mathbf{0}], [\mathbf{0}, .\mathbf{5}, .\mathbf{5}], [.\mathbf{5}, \mathbf{0}, .\mathbf{5}]$, and $[.\mathbf{33}, .\mathbf{33}, .\mathbf{33}]$. Each cluster then simulates the outcome of the coming round as if every cluster played one of the following weights, and plays the best weight for that round. For each cluster, the best weight is defined by the weight that gives the least Squared Error Distortion. Let $V = v_1, v_2, ..., v_n$ be the set of n data points and X be the set of centroids. Then $d(vi, X) = \min(d(v_i, x_i)) \; \forall \; x_i \in X$. The Squared Error Distortion is given by the formula

$$d(V, X) = \sum_{i=1}^{n} \frac{d(v_i, X)^2}{n} \quad \forall v_i \in V.$$

After each cluster has picked its optimal weight, the round is run in full, with distances calculated using the chosen weights for each cluster. A pseudocode illustration of this algorithm is given below:

Compute a list W of the total number of even subsets of w, as defined in X For each cluster round r, r ¡ R (the total number of rounds) For each Cluster c, c ¡ C (the total number of clusters) For each weight value w in W, Run one round of K-Means with all clusters c in C playing w. Compute the Squared Error Distortion for weight w, and store it in a array A. Cluster c will play weight w whose Squared Error Distortion is smallest, as recorded in A. Run round r proper, where the distance from point x to cluster c is computed using the corresponding weight w associated with c. Return cluster configuration C.

KMeansHeuristic can be combined with any of the above variations, such as multiple iterations and initial-point pruning, as its method is contained to a single round of K-means.

4.5 Finding the best k

Another basic limitation of the basic K-means algorithm is that- naturally- the number of clusters k has to be specified beforehand. This can be necessary or

useful in some cases for example, sorting a list of n songs into k pre-specified genres but it is not ideal for situations where the nature of the music being analyzed is not yet known. Furthermore, our goal for this algorithm (as mentioned in 2) is to find musical similarities in songs that may or may not be expected ones that may require more or fewer clusters than the intuitive number.

To account for this, we implement an auto-k method to determine and implement a clustering using the optimal k-value. As with our auto-weights implementation, our auto-k uses Squared Error Distortion (SQE) as a heuristic for determining how uniformly the clusters points convene around a given centroid. This is run for $k : 1 \leq k \leq \log n/3$, a value determined through testing to give a good range of possible k values. The pseudocode for our auto-k implementation is below:

For each value k from 1 to int(log(n)/3) Run k means with value k For each cluster c in final configuration C, find the Squared Error Distortion. Store the average of all values with key k in array A. Find key k in array A with the smallest average Squared Error Distortion. Proceed with k-Means with value k.

As with other applications of Squared Error Distortion, our auto-k heuristic is greatly determined by the p-value chosen to measure Minkowski distance from point to cluster. A higher p-value (such as the infinity norm) will weigh distortion more heavily in favor of clusters that group tightly in any one dimension. Choosing a proper p-norm is a machine learning problem in itself; our algorithm uses a semi-supervised method for finding an optimum p as described in [7]. This method, however, requires a small portion of labeled data to find the p-value that ensures the best cluster, which defeats the purpose of making a fully unsupervised, auto-k algorithm. However, tests with supervised data and variable p on various datasets suggest that $p = 3$ is near optimal for most distortion cases. Unsupervised methods for finding an optimum p have been proposed, [7] but they are beyond the scope of this research to implement.

Like our first optimization, auto-k can be run with other heuristics, such as multiple iterations and initial-point pruning.

5 Testing and Results

In this section, we will consider performance of both feature selection and variants on weighted k-Means clustering on a number of different datasets. Our implementation, `MUS490`, is designed to work on any song dataset of any style or genre with little preprocessing. However, to test the algorithms effectiveness ourselves, we prepared six distinct datasets of various styles and utility for testing. They are:

1. `5Albums`, a dataset of five popular music albums released in 2013
2. `600Songs`, a dataset of 600 .mp3 files selected at random from the authors music collection
3. `Beatles`, the complete discography of the Beatles
4. `ClassicalPiano`, a collection of solo piano works by four Classical and Romantic composers (Bach, Mozart, Chopin, and Rachmaninoff)
5. `Jazz1959`, five albums by Jazz musicians released in 1959 (Bill Evans, John Coltrane, Miles Davis, Ornette Coleman, and Dave Brubeck)
6. `Instruments`, a custom sample library by Open Path Music, made available for the One Laptop Per Child project [20]

In the interest of replicating this data, manifests of each of these datasets files (with metadata) will be made available upon request. In this section, we will consider the performance of `MUS490` in three separate context: instrument/ensemble identification, genre categorization, and wildcard/unlabeled clustering. The most representative results for each of the tests are given in `<name>.log`, in the `logs` folder of the program.

5.1 Instrument and ensemble identification

Test1: Our first test is a 2-cluster split of the Piano dataset, with the hopes that `MUS490` splits by instrument type. (The Bach examples are recorded with a harpsichord, while the other three composers works are played on piano). We run a 2-cluster test using equally weighted MFCCs, along with their first and second derivatives, and iterate until a result is seen at least three times. The results are consistent: `MUS490` successfully splits into one cluster of harpsichord and one of piano. Closer inspection of the saved divergence matrices confirms that the MFCC divergence is comparatively high for piano/harpsichord pairs. A similar run with the auto-k optimization enabled successfully selects $k = 2$ before splitting into the same cluster as above.

6 Future Work

To address these results, and improve the accuracy of future clustering, a number of features are planned for future iterations of `MUS490`. Proposed solutions to these problems are given below.

Outlier problem– In the default algorithm, K-means must consider all points and cluster each one into a separate category. Our initial-point pruning

heuristic favors points that are maximally distant on all dimensions, and it runs recursively until points distant enough are chosen. The consequence of this approach is that it unfairly favors outliers for initial centroids, forming clusters that have their own set of problems. Here we define outliers strictly to better illustrate this point. We say a point o is an outlier if it is the centroid of cluster c and $\min(d(v_i, x_i)) \neq o \ \forall x_i \in X$. If this is true, c will never accrue any more points under either the normal or auto-weight version of K-means, and o will always be the centroid of a one-point cluster. This is especially relevant for datasets of music with individual files that are extremely distinct on one or more feature scales from the other files. The Beatles Revolution 9, for example, is a clear timbral outlier from the rest of their catalog; repeated tests usually pair Revolution 9 with one or two songs at most. When picked randomly as an initial centroid, this song tends to accrue no points. Several researchers have proposed methods for eliminating the outlier problem in a standard K-means setting. Tomi Kunnunen and Pasi FraUMLAUTntis research proposes outlier removal clustering (ORC), consisting of the normal K-means clustering stage and measuring an outlyingness factor for every vector, depending on the distance from each centroid. [25] [15] Outlier removal seems preferable to other methods that increase cluster count with outlier detection.

Minkowski weighted K-means– In our implementation, we adopted the Minkowski metric to compute both literal distance and Squared Error Distortion when finding the best k for clustering. In his PhD thesis, Learning feature weights for K-Means clustering using the Minkowski metric, Dr. Renato Cordiero de Amorim calculates dispersion within a cluster and uses this value to determine the weights per feature. The intuitive idea is that features with a small relative dispersion within a cluster should have a higher weight than a feature with a high relative dispersion within a cluster. [8] His weight calculation is specific to each cluster, but the best weight is found not by subset generation and point accruement (as with ours) but explicitly:

[INSERT FORMULA HEREEEEEEEEEE]

Where the dispersion Dkv is calculated by FORMULAAAAAAAA. This method has the benefit of drastically reducing the weight computation time, especially compared to our approach. We have not yet tested an implementation of this approach, but its integration with the Minkowski metric would make it easy to add to our current implementation.

Parallelization– Unsupervised clustering on large datastets can be time-consuming; our desire to make `MUS490` extensible means that feature vectors have to be calculated from scratch on new data, a processor-intensive task. To account for this, `MUS490` was originally written with parallelization in mind. Feature extraction is designed to be modular, making it easy to parallelize the process. In addition, the venerability of K-means has spurred a number of improvements to the algorithm, both by parallelizing the initialization phase [1] and the clustering phase. [4] While `MUS490` was not designed to prioritize efficiency, improvements from parallelization could increase the number of iterations, more efficiently compute good initial points, or deal with datasets orders of magnitude larger than our test sets.

Other clustering algorithms– Since standard K-means was introduced and canonized by ML researchers for its versatility and effectiveness, other unsupervised clustering algorithms have come to prominence and received much attention. Mean shift algorithms can be specialized to do K-means like clustering, with the added benefit of choosing the number of clusters dynamically. [5] DBSCAN, or Density-based spatial clustering of applications with noise, is another commonly-used clustering algorithm that is most useful with high-volume, high-density clusters separated by low-density spaces. Like mean shift clustering, it does not require a predefined cluster count, and it can both find oddly-shaped, separable clusters and remove outliers effectively. For `MUS490`, K-means was chose for its utility on many types of datasets, regardless of size. In addition, we focused on datasets whose divisions were not always clearly defined, a situation where DBSCAN generally thrives. For more conventional genre classification and small clusters in a large database of songs (e.g. finding rock, hip-hop, and jazz in the Million Song Dataset [2]), an implementation of DBSCAN might yield better results. However, recent genre clustering research has questioned the utility of DBSCAN, owing to the drastic shifts in density from cluster to cluster on all but the most pristine of datasets. [19]

References

[1] Andrea Vattani Ravi Kumar Bahman Bahmani, Benjamin Moseley and Sergei Vassilvitskii. Scalable k-means++. 2012.

[2] Thierry Bertin-Mahieux, Daniel P.W. Ellis, Brian Whitman, and Paul Lamere. The million song dataset. In *Proceedings of the 12th International Conference on Music Information*, 2011.

[3] T.Fillon J.Prado G.Richard B.Mathieu, S.Essid. Yaafe, an easy to use and efficient audio feature extraction software. `http://yaafe.sourceforge.net`, 2010.

[4] Ajay Padoor Chandramohan. Parallel k-means clustering. 2012.

[5] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Anal-*

ysis and Machine Intelligence, 17(8):790–799, Aug 1995.

[6] Peter Chi and C. T. Russell. Statistical methods for data analysis in space physics. http://www-ssc.igpp.ucla.edu/personnel/russell/ESS265/Ch9/autoreg/node14.html, 1999.

[7] Renato Cordeiro de Amorim. Finding the minkowski exponent in mwk-means (semi-sup). http://renatocamorim.com/2012/11/29/, November 2012.

[8] Renato Cordeiro de Amorim. Minkowski weighted k-means. http://renatocamorim.com/2012/11/21/minkowski-weighted-k-means/, November 2012.

[9] Antti Eronen. Automatic musical instrument recognition. Master's thesis, Tampere University of Technology, Tampere, 2001.

[10] Slim Essid and Gael Richard. Musical instrument recognition by pairwise classification strategies. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1403, 2006.

[11] Earl F. Glynn. Correlation "distances" and hierarchical clustering. http://research.stowers-institute.org/mcm/efg/R/Visualization/cor-cluster/index.htm, 2005.

[12] Don H. Johnson and Sinan Sinanović. Symmetrizing the kullback-leibler distance. *IEEE Transactions on Audio, Speech, and Language Processing*, 2000.

[13] Dimple Malik Kehar Singh and Naveen Sharma. Evolving limitations in k-means algorithm in data mining and their removal. *IJCEM International Journal of Computational Engineering and Management*, 12:105–109, April 2011.

[14] Tom LH. Li and Antoni B. Chan. Genre classification and the invariance of mfcc features to key and tempo. In *International Conference on MultiMedia Modeling*, pages 32–34, 2011.

[15] M. H. Marghny and Ahmed I. Taloba. Outlier detection using improved genetic k-means. *International Journal of Computer Applications*, 28(11):33– 36, August 2011.

[16] Carl D. Meyer. *Matrix Analysis and Applied Linear Algebra*. Society for Industrial and Applied Mathematics, first edition, 2001.

[17] Yang Hong Michael Haggblade and Kenny Kao. Music genre classification. Undergraduate honors thesis, Stanford University.

[18] Levent Ertoz Michael Steinbach and Vipin Kumar. The challenges of clustering high dimensional data. 2004.

[19] Walter Nordström and Jacob Hakansson. Finding clusters of similar artists - analysis of dbscan and k-means clustering. April 2012.

[20] OpenPathMusic. The openpath music custom sample library. https://archive.org/details/OpenPathMusic44V1.

[21] Alan V. Oppenheim and George C. Verghese. Signals, systems, and inference. page 150. Massachusetts Institute of Technology, 2010.

[22] Douglas O'Shaughnessy. *Speech communication: human and machine*. Addison-Wesley Pub. Co., 1987.

[23] Walter Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, third edition, 1976.

[24] Ryota Suzuki and Hidetoshi Shimodaira. Pvclust: an r package for assessing the uncertainty in hierarchical clustering. svitsrv25.epfl.ch/R-doc/library/pvclust/html/pvclust.html, 2006.

[25] Ismo Ka rkka inen Tomi Kinnunen Ville Hautama ki, Svetlana Cherednichenko and Pasi Fränti. Improving k-means by outlier removal. 2005.

[26] Matthias Zerbst and Lars Tschiersch. The concentration centroid minimum distance clustering criterion. 2002.