Opening up Pandora('s box)

Brian Magee and Prasanth Prahladan

December 15, 2015

1 Introduction

Audio/Music hosting applications like Pandora, Spotify, SoundCloud, Saavn etc, have the immense responsibility of being expected to serve as search-engines within the audio/sound-space. As a user, one expects that one's "taste" in music, is "understood"(learned) by the application, and that it shall then help the user explore and listen to artistes, tracks and albums, that would otherwise be difficult for the user, to individually search out for herself. The greater the search and exploratory experience, the better is the chance of the application being the favoured source of music-entertainment for its human-user-base.

Given this context, the course project helps to give us a hands-on experience with building up a "music-genre classification engine". In the "music-genre classification" problem, we are required to determine the genre of a given query song, based on an algorithm that builds an internal representation of different music-genres, using a fixed database of classified(tagged) songs. The purpose of the project, is to enable the student to understand how to build a rudimentary (but functional) prototype of a music-classification system. As a design challenge, the students are required to understand the high-level process of mining for patterns in a given music database, to split the process-flow into different sub-systems(algorithms) and determine how these functional-blocks may be combined to determine a process-flow for improving the efficiency of the classification problem. The specifications of the course-project as described in the project-guide, indicates the genres we need to focus on and the number of songs assigned to each genre.

The music-genre classification process can be summarized by the following stages:

1. Embedding of Dataset into a Metric Space:
   It is important to note that the representation of data-files in memory, needs to be chosen based on the application for which that data shall be utilized. A given song when represented/stored as a .mp3 or .wav file, may be processed and played using an Audio player. However, for the purpose of classification, we need to design/develop a particular vector-representation that incorporates certain "descriptive features" of the audio file. These "features" may be based on analytical notions of time-frequency domain representation or based on "perceptive/cognitive" notions. The representation of these features, as mathematical vectors embedded in a high-dimensional space, then permits us to use algorithms for processing vectorial data, for the purpose of automated classification/clustering. We intend to use the "mel-frequency cepstral coefficients", its derivatives and possibly other "high-level-feature" information for each of the song files.

2. Visual Pattern Detection of Music Database:
   The preliminary step involves the use of a Data-visualization library, to determine whether there are any patterns like clusters in the data, which can then be exploited. It would be helpful to understand the logic/ideas behind the algorithms used to guide the visualization process, which could then be incorporated into the classification engine, that we need to develop.

3. Determine Intrinsic Dimension of the Data:
   If any patterns(clusters) have been detected in the data-visualization, it helps to assume that the different genres of high-dimensional data in fact is distributed along low-dimensional manifolds embedded within the high-dimensional space. We can then use algorithms for determining the intrinsic dimension of these manifolds (e.g "Correlation Dimension"). By determining the intrinsic-dimension of each of the genres, we can determine the minimum dimension of the space into which all the data can be embedded without losing much information.

4. Metric Learning:
   Ideally, when data are categorized into categories, we would like to believe that the songs/data-points that

are ”similar” to each other are ”closer” to each other, and those that are ”dissimilar” are ”farther” apart. The closeness of two data-points is determined by the ”distance metric” used to compute the distance between the two points. The reason such a ”metric” would exist, can be deduced from the fact that the data points are distributed upon a Manifold. The measure of distance between any two points, thus becomes the distance travelled along the manifold surface between the points, rather than the shortest euclidean straight-line distance between them. The ”metric-learning” problem deals with the problem of ascertaining the metric for each of the manifolds separately, or the notion of a ”global-metric” for all the data-points.

5. Dimension Reduction:
   Once the minimal intrinsic dimension of the datasets has been determined, and we have a notion of ”distance-metric” that helps compute the distances between the points, we then deal with the problem of reducing the dimension of the data. We note that this procedure, helps us determine how much of the information from the initial choice of vector representation of the dataset is redundant. The solution to this problem, is the design of a Map/function that projects every data-point from the high-dimensional space to the low-dimensional space. To ensure that the representation for each data-point is unique in the reduced dimension, we impose the constraint of an injective mapping, on this function.

6. Clustering:
   If we have identified patterns in the data-visualization stage, we would assume that the songs belonging to the same music-genre are distributed as clusters in the high-dimensional space. We expect that the dimension-reduction process does not hamper/destroy the clustering of the dataset, but rather reinforce/amplifies the extent of clustering - ”similar” songs are brought closer together and ”dissimilar” songs are pushed farther apart. It is obvious that the ”distance-metric” used to measure the distances between songs in the reduced-dimensional space is very-different from that in the high-dimensional space. However, we can demand that the distance metric in the reduced-space be closer to the Euclidean space. The ”metric” in the reduced-space shall then be used to identify clusters in the dataset.

7. Query Processing (Cluster Assignment/Statistical Learning):
   This forms the final stage of the Genre-Detection problem. Once the abstract model - a spatial distribution/representation of the songs in the database has been determined, we have now developed the capacity to determine the genre/family that a particular song-query might belong to. We can develop a method to determine the genre-of song, by determining the cluster that it belongs to, the proximity to its neighbours and the classification of its neighbors into the different genre.

2 Feature vector representation

To derive content-based features representative of songs, we would have to implement digitial signal-processing algorithms upon the .wav song files, to compute quantitative information about the song. Some of the features that can be extracted from a song are mfcc, fluctuation patters, etc. In this project, we focus on using the Spectral Information of the songs - represented by MFCC Coefficients, to serve as the primary signature of a song-frame. Songs are then classified into genre based on the similarity between these signatures.

2.1 Single Gaussian Model of each song

By adopting a Single-Gaussian Model representation of a song, one assumes that d-dimensional mfcc-coefficient vectors representative of each of the innumerable frames extracted from a song, can be obtained by sampling a multi-variate Gaussian in the d-dimensional song-mfcc-space. Thus, each song is represented by a Gaussian, with a mean-mfcc-coefficient vector and a covariance-mfcc-coefficient matrix that is obtained from the mfcc-coefficient vector representations of the frames extracted from the song.

Given, this representation of the song, the feature-vector used for computational purposes, could either assume the songs to be objects between which the distances are measured, by specialized mathematical schemes like KL-divergence, Earth-Movers-Distance, etc which are methods for computing distances between any two distributions. Or, as we have implemented, a custom-feature vector can be derived by concatenating the mean-mfcc vector($[1xd]$ dimensions) and the upper-triangular part of the mfcc-covariance-matrix, to form an extended feature-vector ($[1x(d + d(d + 1)/2)]$ dimensions).

2.2 Code-word Histogram

The code-word histogram [6] approach to feature-vector representation of a song can be described intuitively as follows. All the music files in the database can be assumed to be 'sentences' that are made up of a fixed number of 'code-words' in the vocabulary of music. Each song is represented by the number of times specific 'code-words' occur in it. The word histogram is used to describe this method of counting the number of times a given code-word occurs in the song. Thus, by assuming such a finite word vocabulary to describe all possible songs in the database, we enable a mechanism of drawing parallels between comparison of textual-documents via Natural-Language processing algorithms, and the analysis of music.

Mathematically, the code-word histogram generation process can be described as follows. To obtain a compact summary of audio signals each song is represented as a histogram over a dictionary of timbral codewords. As a first step, a codebook is constructed by clustering a large collection of feature-descriptors. Once the codebook has been constructed, each song is summarized by aggregating vector-quantization representations across all frames in the song, resulting in codeword histograms. Finally, histograms are represented in a nonlinear kernel space to facilitate better learning of the distance metric between the songs.

2.2.1 Codebook training

The primary audio feature descriptor used for training of the codebook, is the mfcc coefficients of frames of a song. The audio-feature descriptors of all the songs in the database is aggregated into a single bag-of-features, which is then clustered to produce the codebook. In our implementation, the number of words in the codeword dictionary is arbitrarily chosen to be about 512.

For each song x in the codebook training set X_C, we compute the 13 MelFrequency cepstral coefficients(mfcc) from each overlapping 25-ms frame. From the time-series of the MFCC vectors we compute the first and second instantaneous derivatives, which are concatenated to form a sequence of 13×2 dimensional *first-order dynamic-MFCC*(FD-DYN-MFCC) and 13×3 dimensional *dynamic-MFCC*(DYN-MFCC). These descriptors are then aggregated across all $x \in X_C$ to form an unordered bag of features Z, where each $z \in Z \subset \mathcal{R}^D$ is either an MFCC, FD-DYN-MFCC or DYN-MFCC feature-vector representation of the song-frames.

To correct for changes in scale across different dimensions of $z \in Z$, each vector is normalized according to the sample mean $\mu \in \mathcal{R}^D$ and standard deviation $\sigma \in \mathcal{R}^D$ estimated from Z. The i'th coordinte is mapped by

$$z[i] \mapsto \frac{z[i] - \mu[i]}{\sigma[i]}. \tag{1}$$

The normalized feature vectors are then clustered into a set $\mathcal{V}$ of $|V| = 512$ codewords by k-means algorithm.

2.2.2 (Top τ) Vector Quantization

Once the codebook $\mathcal{V}$ has been constructed, a song $x \in X_C$, is represented as a histogram h_x over the codewords in the codebook. Each song $x \in X_C$ is understood to be a time-sequence of feature-vectors, $z = z_i \in Z \subset \mathcal{R}^D$ where z_i is the feature-vector representation of a frame in the song. Each $z_i \in z$ is normalized according to (1). The codeword histogram of song $x \in X_C$ is constructed by counting the frequency with which each codeword $v \in \mathcal{V}$ quantizes the elements of z i.e

$$h_x[v] = \frac{1}{|z|} \sum_{z_i \in z} \frac{1}{\tau} 1 \left\{ v = arg \overset{\tau}{\min_{u \in \mathcal{V}}} ||z_i - u|| \right\}. \tag{2}$$

where we have chosen to adopt multiple codeword quantizers of each vector z_i by defining the quantization set

$$arg \overset{\tau}{\min_{u \in \mathcal{V}}} = \left\{ u \text{ is a } \tau\text{-nearest neighbor of } z_i \right\}. \tag{3}$$

where $\tau \in 1, 2, \cdots, |V|$. Note that the codeword histograms are normalized by the number of frames $|z|$ in the song in order to ensure comparability between songs of different lengths. Further, the normalization by $1/\tau$ ensures that $\sum_v h_x^\tau[v] = 1$, so that for $\tau > 1$, h_x^τ retains its interpretation as a multinomial distribution over $\mathcal{V}$.

2.3 Elias-Pampalk's Choice of Features

The music-analysis toolbox that was provided as a part of the initial project-dataset contained a host of functions that were used by Elias Pamapalk's submission for the MIREX'2004 competition [1]. He had chosen to represent each song by certain high level features called 'fluctuation patters' and also, a single-gaussian model for the frames of mfcc coefficients that were extracted from each song. We chose to use the same set of feature-vector representation and distance/similarity computation measures, as we believed that it would serve as an effective base-line for comparing the utility of alternative music-genre classification that pipelines we develop as a part of the course.

The distance measure was computed using the formula:

$$\begin{aligned}D_{ij} &= 0.7 * (-exp(-1/450 * d_{mfcc}) + 0.7950)/0.1493 + 0.1 * (d_{fp} - 1688.4)/878.23 \\ &\quad + 0.1 * (d_{fpg} - 1.2745)/1.1245 + 0.1 * (d_{fpb} - 1064.8)/932.79 + 10;\end{aligned}$$

We note that the distance measures from different feature-vector representations of the data-set, were combined to obtain a single distance measure by means of numerous parameters that were obtained from other experimental techniques, that the author mentions in his thesis.

3 Distance and Similarity in song-space

The choice of the Distance-Metric depends on the chosen Feature-Vector-representation for each song. The naive method to proceed would be to assume that all the songs are embedded in an Euclidean Space, and hence the distances can be computed using the L-2 Norm.

However, for the specialized representations of the songs that we have adopted, we believe that the songs are distributed over a manifold, and hence the distance between the points need to be computed by using a metric intrinsic to that manifold. To explore this idea, we have decided to implement the Information Theoretic Metric Learning toolbox [3], for determining the appropriate "metric" that helps classify and categorize the music genres.

3.1 Information Theoretic Metric Learning

The information theoretic metric learning [3] approach to learning the distance function, is a particular technique for computing the Mahalanobis distance between two points in a metric-space. The Mahalanobis distance generalizes the standard Euclidean distance by admitting arbitrary linear scalings and rotations of the feature space. The problem of learning an optimial distance-metric is translated to learning the optimal Gaussian with respect to an entropic objective function. The model adopted leverages the relationship between the multivariate Gaussian distribution and the set of Mahalanobis distances.

Given a set of points $x_i \subset \mathcal{R}^n$, and a distance metric A, the distance measure between the points is computed as

$$d_A(x_i, x_j) = (x_i - x_j)^T A (x_i - x_j)$$

Consider (dis)similarity relationships between points provided by information regarding the points belonging to the same(or different) genres. This may also be interpreted as, similar points have their Mahalanobis distance less than a given upper-bound ($d_A(x_i, x_j) \leq u$), and dissimilar points have their distances greater than a lower-bound ($d_A(x_i, x_j) \geq l$), for sufficiently large l.

Assume that we have two different metrics, A and A_0. We can quantify the closeness between A and A_0 via a natural information-theoretic approach. There exists a simple bijection between the set of Mahalanobis distances and the set of Gaussian Multivariate distributions. Given a Mahalanobis distance characterized by A, we express its corresponding multivariate Gaussian as $p(x; A) = \frac{1}{Z} exp(-\frac{1}{2} d_A(x, \mu))$, where Z is a normalizing constant and $A^(- 1)$ is the covariance of the gaussian distribution about the mean μ. Using this bijection, we measure the distance between two Mahalanobis distance functions parametrized by A_0 and A by the differential relative entropy between their corresponding multivariate Gaussians:

$$KL(A_0 || A) = \int p(x; A_0) log(\frac{p(x; A_0)}{p(x; A)}) dx \tag{4}$$

The distance provides a well-founded measure of closeness between two mahalanobis distance functions and forms the basis of the optimization problem given below:

Given the pair of similar points S and set of pairs of dissimilar points D, the distance-metric learning problem is

$$\begin{array}{lll} min_A & KL(p(x;A_0)||p(x;A)) & \\ \text{subject to} & d_A(x_i,x_j) \leq u & (i,j) \in S \\ & d_A(x_i,x_j) \geq l & (i,j) \in D. \end{array} \tag{5}$$

The above optimization problem is solved as a log-determinant optimization problem. The details of the algorithm used and the implementation of the algorithm can be found in the original research paper referenced below [3].

3.2 Similarity between songs in song-space

Given a measure of distances between songs, as computed by the computed/assumed distance metrics, we also have information regarding the genre-id's of each song. We expect that songs of a same genres are more similar than songs of dissimilar genres. The similarities between songs can be quantitatively represented as a matrix containing values which are a function of the distances between the points.

When using only the distance information to describe similarity, we can choose the function to be any number of non-linear functions like:

1. Gaussian Similarity $s_{ij} = exp(-d_{ij}^2/2\sigma)$,
2. Inverse distance $s_{ij} = \frac{1}{1+d_{ij}^p}$, where $p \geq 1$,etc.

We use the Similarity Matrices computed above, as inputs to some of the nonlinear dimension reduction techniques like the Refined-Graph-Embedding.

4 Data Visualization

The t-SNE algorithm [4] is an award-winning algorithm, for visualizing high-dimensional datasets. We shall first implement this algorithm, to determine a possible visual representation of the database, that shall indicate visual patterns in the data. This visualization shall help us develop an intuition of what the data distributions might look like.

This algorithm provides us the opportunity to check the efficiency of the vector-space embedding of the selected features from the songs in our database. If the visualization does not separate our vector-space data set into a sufficient number of discrete clusters, then it is likely that our choice of features is inadequate to truly separate the different genres when implementing our own dimension reduction techniques. In this case we will augment the vector-space embedding by adding additional features from the songs. This visualization algorithm can also be used after each stage of the pipeline to check that the information in our dataset has not been adversely deformed. Further, this algorithm can also be used a dimension-reduction technique, and shall be described in the appropriate section below.

5 Dimension reduction

We shall create and compare multiple pipelines for the dimensional reduction process. Both linear and non-linear graph based methods shall be explored. An interesting application we intend to explore, is the extension of the work on learning multimodal similarity [5] to determining similarity between genres. The process would thus adopt a soft-categorization procedure, rather than assuming the clusters to be shaped as hard-bounds.

Linear method using Fast Johnson-Lindenstrauss transform (FJLT) algorithm. This method uses random projections to map our data from the initial high d-dimensional space to a much lower k-dimension space by allowing low distortions of the data which is controlled with a tolerance parameter (epsilon). The choice of ϵ impacts the choice of $k-$dimensional space into which the data can be suitably embedded.

Non-linear, Graph-based Refined Embedding. This method builds a similarity graph, between all the data-points, where the edge-weights are computed using a particular kernel function and distance-metric. From the similarity graph, we derive the Graph-Laplacian and follow the procedure of spectral embedding of this graph. The eigenvalues and the eigenvectors of the Graph Laplacian shall indicate the existence of the clusters.

5.1 Fast JL Transform

In some implementations of our test algorithms, we use the Fast Johnson-Lindestrauss Transform (FJLT)[2] as a method to reduce the dimensionality of our feature vector data. This allows a fast, robust method for which to reduce the dimension while incurring a slight deformation of the pair-wise distances between data points in Euclidean space.

$$||x-y||_2(1-\epsilon) \leq C||\Phi x - \Phi y||_2 \leq ||x-y||_2(1+\epsilon)$$

We show that with our 256-dimension feature vector for each of 729 songs, we can reduce the feature representation to 20 dimensions while incurring a reasonable deformation of the pair-wise distances. The non-uniform deformation across all pair-wise distances due to the randomized nature of the FJLT is countered by performing an iterative procedure on the FJLT. We found via experimental results that 20 iterations was suitable to obtain near-exact results to those in the original high-dimension, and this was used for all implementations of FJLT in our cross-validation experiments.

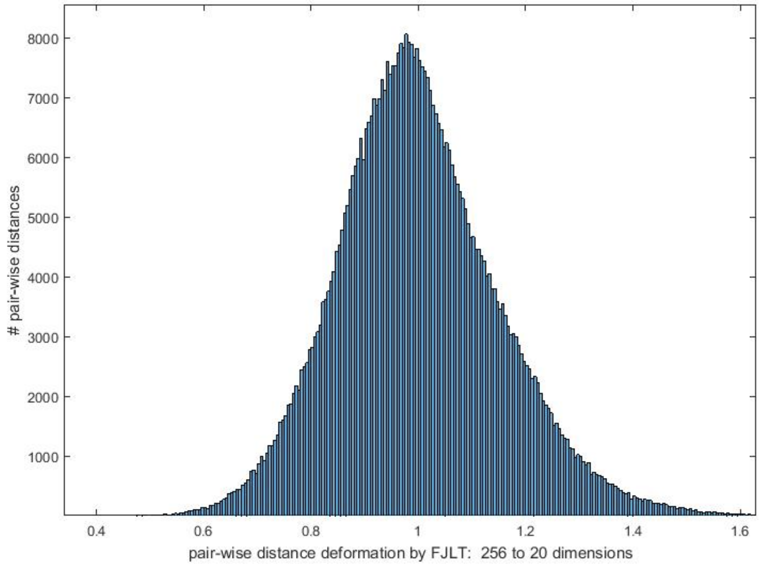


Figure 1: FJLT Deformation

The deformation of the pair-wise distances due to application of the FJLT from the original 256 dimension feature vectors to 20 dimension feature vectors. The deformation is calculated by dividing the transformed pair-wise distances by the original pair-wise distances, and then dividing each by the mean of this distribution (the constant C, which is approximately 1/27). In this single instance of the FJLT, we find 82% of all pairwise distances within a deformation of 1 ± 0.2; 95% within 1 ± 0.3; and 99% within 1 ± 0.4.

5.2 Refined Graph Embedding

The Refined Graph Embedding relies on theoretical concepts covered in Spectral Graph theory. Let $G(V, E)$, with $|V| = n$ and $|E| = m$, be an undirected graph without self-loops. Let A be the adjacency matrix defined as $A_{ij} = \imath[(i,j) \in E]$ and D be the degree-matrix of the graph $D = diag(d_1, d_2, \cdots d_n)$ with $d_i = \sum_{i=1}^{n} A_{ij}$. We further define the following matrices

$$\begin{array}{cc} L = D - A & \text{combinatorial unnormalized graph Laplacian} \\ P = D^{-1}A & \text{markov probability transition matrix} \\ \mathcal{L} = I - D^{-1/2} * A * D^{1/2} & \text{normlaized graph laplacian} \end{array} \tag{6}$$

The largest eigenvalues and the corresponding eigenvectors of $\mathcal{L}$ are computed. Next, a stationary probability distribution is computed for the Markov Chain transition matrix P. The coordinates of the point in the reduced dimensional-space are obtained from the eigenvectors and the stationary-probability distribution computed. The details of the implementation of the graph-embedding is covered in the lecture notes [8] and the home-work assignment for the course, and hence shall not be explained in detail here.

5.3 t-Distributed Stochastic Neighbour Embedding

The t-SNE algorithm [4], was introduced earlier as a technique for visualization of high-dimensional datasets, to built a visual intuition of the data-distribution. From the fact that local neighbourhood information of the data is preserved by the technique, it would serve as a useful dimensional reduction technique, and hence used this method as an alternative to the Graph Embedding Technique.

tSNE starts by converting the high-dimensional Euclidean distances between data points into conditional probabilities that represent similarities. The similarity of data-point x_j to x_i is the conditional probability $p_{j|i}$ that x_i would pick x_j as a neighbour, if the neighbours were picked in proportion to their probability density under a Gaussian centred at x_i. Mathematically, the conditional probability is computed as

$$p_{j|i} = \frac{exp(-d_{ij}^2/2\sigma_i^2)}{\sum_{k\neq i} exp(-d_{ik}^2/2\sigma_i^2)} \tag{7}$$

where σ_i is the variance of the Gaussian centred about x_i. Because, we are only interested in modelling pairwise similarities, we set the value of $p_{i|i} = 0$.

Now, we assume that there's a particular projection operation that projects each of the songs x_i to lower dimensional point y_i. For the low dimensional counterparts y_i and y_j of the high-dimensional points x_i and x_j, we can compute the conditional probability, $q_{j|i}$ as

$$q_{j|i} = \frac{(1+d_{ij}^2)^(-1)}{\sum_{k\neq i}(1+d_{ik}^2)^(-1)} \tag{8}$$

where, we use a t-Student distribution with a single degree of freedom, instead of a Gaussian distribution about point y_i. This is because, it has the particularly nice property that $(1+d_{ij}^2)^(-1)$ approaches an inverse-square law for large pairwise distances in the low-dimensional map. This makes the maps representation of the joint probabilities almost invariant to changes in scale of the map for points that are far apart. Which also implies that large clusters of points that are far apart interact in just the same way as individual points, so the optimization operates in the same way at all but the finest scales.

If the projections correctly model the similarity of the points in the high dimensional space, then the conditional probabilities $p_{j|i} = q_{j|i}$. Motivated by this intuition, the tSNE algorithm aims to find a low-dimensional data representation that tries to minimize the mismatch between joint conditional probabilities P and Q. A natural measure of the distance between two distributions is the Kulback Liebler-divergence. tSNE minimizes the KL-divergence between P and Q, via a gradient descent algorithm. The cost function is given by

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} log(\frac{p_{j|i}}{q_{j|i}}) \tag{9}$$

where it is assumed that $\sigma_i = \sigma$ for all data-points.

The gradient of the Kullback-Liebler divergence between P and the t-Student based joint probability distribution Q is given by

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1+d_{ij}^2)^(-1) \tag{10}$$

where $d_{ij} = ||y_i - y_j||$

6 Clustering and Statistical Learning

A k-nearest neighbor (k-NN) approach is adopted for the task of genre-classification once our data has been transformed into its final representation (after feature vector compilation and any dimension reduction). Euclidean distances are computed between the query and all training data, and the k-nearest neighbors are returned with their distance and genre. In the raw voting case, each of these k neighbors is granted an equally weighted vote of value 1. Thus the genre with the largest total vote value is simply the genre which comprised the highest number of neighbors in the set of k neighbors. We also institute the desired option of vote weighting, by which we apply a weighting function to each individual vote. Each vote can be weighted on the distance of the neighbor, typically weighted by 1 / (1 + d) such that closer neighbors are given more voting power. Additionally we allow the application of genre weighting which applies a constant weight to the votes depending on the genre, in order to help to account for the disparity between sample sizes of the different genres. It is inherently unfair, for example, that jazz has 26 total data points compared to 320 for classical. We first experimented with each vote being weighted by (total data points / data points per genre) but this ultimately led to each jazz data point having a tenfold voting power over an equally close classical song. Such a scheme ended with an extreme amount of false positives granted to the genres with fewest samples. We therefore implemented some functions such that we capped the genre-to-genre weighting ratio to a maximum of 3-to-1, which gave better overall results.

7 Putting them together

7.1 Experiment 1: Baseline

For our baseline algorithm, we represent each song as a 256 dimension feature vector representing the average spectral response of the entire song. We first read in each .wav file and extract the first 20 MFCC coefficients for all frames (23 ms) of the entire song. We then compute the mean MFCC coefficients over all extracted frames, as well as the covariance matrix of these coefficients. We then vectorize the covariance matrix and append all values together in a standardized vector structure (removing redundant entries of the symmetric covariance matrix). The dimension is set to 256 so that we can utilize an implementation of FJLT to run the experiment in a lower dimensional space. Here we use the standard Euclidean distance as our distance metric. As a baseline, this experiment is able to obtain decent overall results (65% for all songs, 70% for 3 genres), and presents a good comparison for our further algorithms.

7.2 Experiment 2: MFCC code-word histograms

We expect the different genres are comprehensive, in the sense that all songs that do not get classified into either of the specialized genres get assigned to the category ”World”. Thus, it is possible that even white-noise could be classified as ”World” music. Therefore, it is important to build a binary classifier that distinguishes between World and all the other families. We hope the method of Support Vector Machines can be used to discriminate between the ”world” and ”non-world” categories.

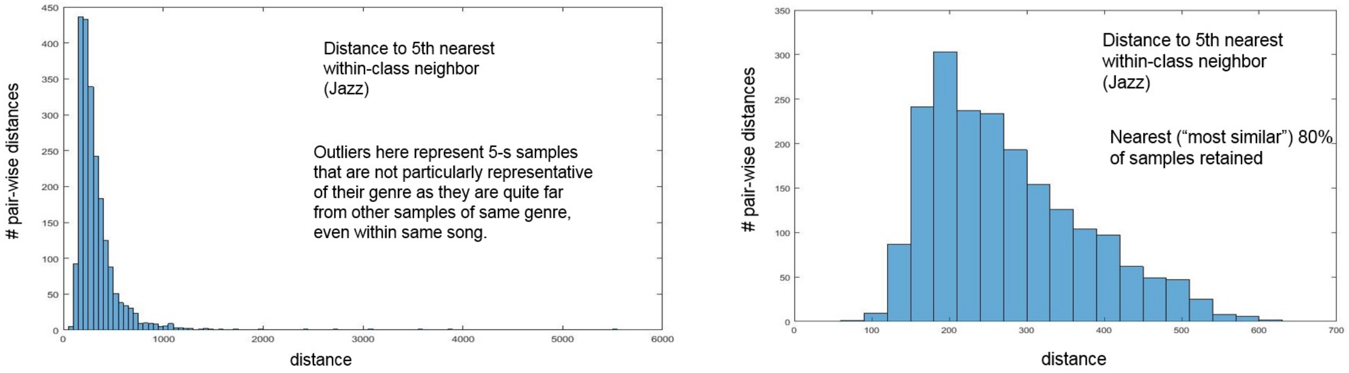
This is followed by a pipeline of algorithms that implement global dimension reduction, clustering algorithm, and statistical learning, to determine the probability of cluster membership of the query data.

7.3 Experiment 3: 5-second song samples

We also decided to explore the idea of representing songs as collections of shorter audio samples that occur in sequence. Much like the codeword histogram technique, this method aims to exploit more information about each song (i.e. its dynamic nature) as opposed to simply summarizing it with a single feature vector. The intuition behind this approach is that a human listener can tell a lot about a song (specifically the genre) fairly accurately when provided only a few seconds of the songs music. Generally, one need not listen to an entire song to assign that song to a genre. Separating each song into shorter song samples changes the dynamic of the genre classification problem, ultimately leading to several advantages as well as a couple of disadvantages. (advantages) Vastly increases the number of total data points representing our song database from 729 to 55,000. More connected sub-communities in high dimensional space Ability to determine and remove outliers Potentially better identify hybrid genre songs Potentially identify a songs genre (and song-wise similarity) accurately from only a portion of the original song. (disadvantages) Greatly increased sample size can hinder computation time via scalability problems Each song sample may describe less unique information (i.e. redundancy in the data)

Processing of the data is as follows. Each song was first normalized such that the maximum volume (waveform amplitude) was normalized to a value of 1. Then the MFCC coefficients were obtained using 23ms frames. From the total frames we determine the number of song samples available in the song, using 500 consecutive frames (5 seconds of audio) for each sample with a 250 frame overlap between consecutive song samples. The song samples were prepared such that any unused additional frames were evenly divided between the very beginning and very end of the song (i.e. generally less than 5 seconds per song is then unused). A feature vector is produced for each song sample as summarized by the mean of their 20 MFCC coefficients along with a vectorized representation of their covariance matrix (just like our baseline experiment). The values are placed into a vector of length (dimension) 256, with zeros in the unused dimensions, so that they are in a format ready to processed using a FJLT implementation if desired. This methodology allows us to vastly increase the sample size of the entire audio space from our given data set from 729 individual songs, to approximately 55,000 total five-second song samples. This explosion of total sample size does inherit the relative disparity in sample size between the genres, but allows some interesting possibilities in the way that we proceed with the analysis.

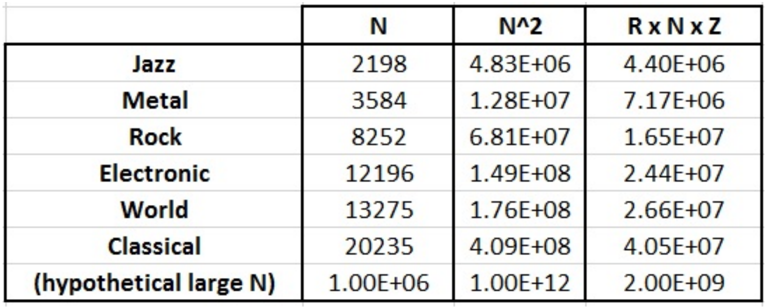
Now that our sample size is so large, each individual data sample is not as precious and we can diagnose and remove outliers that could decrease the efficiency of our genre classification algorithm. Our definition of outlier in this context is an individual song sample which does not accurately describe its own genre. To quantify this, we compare the pair-wise distances of each song sample with all other song samples of the same genre. We then look at the distance to the fifth-nearest within-genre neighbor and use this as a metric from which to compare the connectedness of the song samples. This metric was chosen in order to discriminate between song samples near to only a few others of the same genre, and those near to a more significant number of samples from the same genre. The $X\%$ of samples with the highest distance are removed as being outliers as they are assumed to be the least representative of their genre. However, with the large increase in sample size, this pre-processing step can be inefficient with respect to computation time. With a sample size of N, it requires N^2 computations to determine all pair-wise distances. This may be alright for Jazz (2,198 total song samples), but not for Classical (20,235 total song samples). Therefore we used an iterative technique, determining outliers for only 500 random song samples at a time for all groups of 500 samples per genre, then repeat the process four times. Thus we reduce the number of operations from N^2 to $R \times N \times Z$ where R is the number of repetitions (we chose 4) and Z is the number of song samples for which analyze for outliers at a time (we chose 500). Below is a table that shows the computational savings. In order to obtain an end result of retaining $X\%$ of the total samples, we must actually retain a larger proportion at each iteration $X = \left(X^{\frac{1}{R}}\right)^R$. With four iterations we must keep 95% of the samples at each iteration to ultimately retain 80% of all samples.



(a)

(b)

Figure 2: (a)Original distribution for 5-th nearest neighbour distance for all jazz song samples (b) Retained distribution with outliers removed.



	N	N^2	R x N x Z
Jazz	2198	4.83E+06	4.40E+06
Metal	3584	1.28E+07	7.17E+06
Rock	8252	6.81E+07	1.65E+07
Electronic	12196	1.49E+08	2.44E+07
World	13275	1.76E+08	2.66E+07
Classical	20235	4.09E+08	4.05E+07
(hypothetical large N)	1.00E+06	1.00E+12	2.00E+09

Figure 3: Computational savings from iterative outlier removal.

As this outlier removal step is considered pre-processing, we must only consider removing outliers from the training data and not the test data (query songs). Therefore an outlier index variable is stored in order to account for this, and known outliers are used as normal data points for query songs as it is assumed that this outlier detection could not be done in a real-time application.

Because each song is now represented as multiple data points, we have some flexibility in how to ultimately determine the genre for a query (test) song. We could run our k-NN algorithm and assign each query song sample to a genre and then choose the genre for which the highest number of samples were assigned. However, we chose to instead retain the k-NN votes for all samples and only compare their sum. This approach retains the vote-weighting and effectively applies that to each individual sample. In this way, a song sample which is far from its nearest neighbors (i.e. is an outlier) is not given the same voting power as a song sample which is very near to its nearest neighbors. Also, this can account for uneven voting for different parts of a song, and potential disparity among available votes per genre. A song whose samples had significant Jazz votes but never a majority of Jazz votes in any individual sample would retain the influence of those Jazz neighbors, whereas assigning each sample to a genre would acquire no Jazz votes in the final song genre assignment phase.

8 Testing and Verifying the pipelines

The different pipelines are evaluated in the following manner.

The feature vectors of concern is extracted for every song in the database. The distance metric used to computed the distance between the feature vectors is either assumed to be Euclidean, or is learned by using the 'Information Theoretic Metric Learning' [3] toolbox. Using the distance metric and the 'true' genre assignments (from the truth file) we use different dimensional reduction techniques to embed the data points into a smaller m-dimensional space. As a default value, for a preliminary run of the experiments, we have chosen $m = 3$. However, further experiments are required to determine the clustering accuracy on increasing the dimensions. One of the interesting experiments we identified, was to determine the intrinsic-dimensionality of the different genres, under different choice of feature-vector representation of the songs. This would give us some information on how to upper-bound the dimension(m) of the feature-space for the embedding procedure.

After computing the embedding of the songs in the reduced dimensional space, we adopt an iterative procedure of cross-validation of our classification scheme. At every iteration, the entire music database is randomly split into two segments - the training(80%) and testing dataset(20%), by generating proportional partitions within each genre. To be precise, for every genre, the set of all songs belong to it is partitioned into two subsets - training-set(80%) and testing-set(20%). The global-training-set(testing) is obtained by coalescing all the training-sets(testing) of each genre into one set. For each song, within the global-training-set, we determine its cluster-membership, by seeking out its neighbours from the global-training-set, and implementing a voting scheme. The results of the cluster-membership-assignment for the testing-set is compiled by creating a confusion-matrix.

The confusion-matrix is obtained by determining the percentage of songs of a given genre, that have been classified as belonging to each of the unique genre-types. This is possible, as the information regarding the true-categorization of the testing-data is known to us. The mean and standard-deviation of the confusion-matrices obtained over all iterations, for a given genre-classification-pipeline, are computed and archived. The effectiveness of each pipeline is determined by comparing these statistics of the confusion matrices. The results of the experiments are illustrated in the Appendix.

9 Discussion

In this project we have implemented two novel pipelines geared towards handling the specific challenges related to the genre-classification problem over the given dataset:

1. The limited number of song samples from specific genres
2. Identifying the metric for computing distances between the songs, in feature space.

The detailed statistical results of the experiments are provided in the appendix. However, we wish to summarize the outcome of the experiments below. The baseline experiment was implemented on the single gaussian model representation of each song(mfcc). The statistical results for this experiment was used to compare the performance of the novel pipelines that we designed. We also verified that using an iterative FJLT implementation to vastly reduce the dimension of the feature-space was effective in obtaining near-exact results to the performance in the original high-dimension.

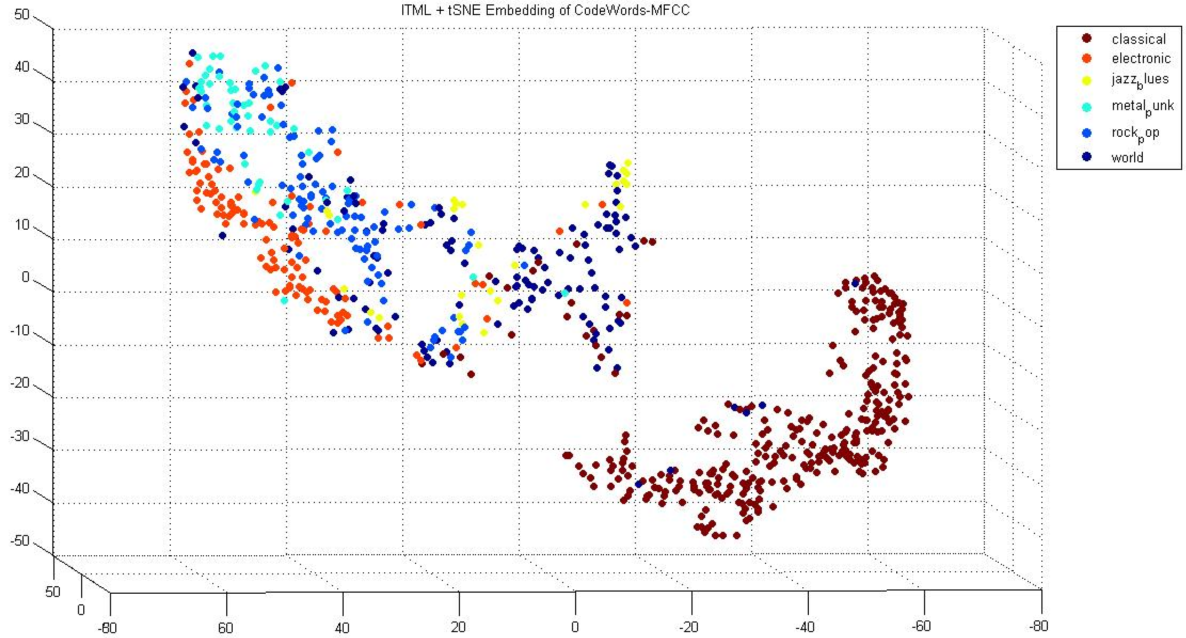


Figure 4: Plot of the structure of dataset obtained by using a pipeline comprising of mfcc-code-word histogram features, itml and tsne.

In the experiment based on the mfcc-codeword-histogram, we identified that the implementation of a metric-algorithm to compute the distances between the songs in feature-space, is of paramount importance. This is seen by the poor performance of the algorithm, that relies on an Euclidean metric in measuring song-similarity. Using a combination of codeword histograms, itml and tSNE , we obtained an average success rate of 68% for all songs, and more than 50% accuracy for all genres except 'world'.

Concerning the experiment based on the '5s song samples', we observed significant improvement over the baseline, particularly in the 'Electronic' genre. This is achieved by utilizing the same feature vector representation of the data, with only splitting up the song into an array of fixed 5s intervals. The overall results are slightly better than the code-word histogram pipeline. However, it does not achieve the same genre-genre success rate. While this is true, the confusion matrix indicates that the confusion between the genre's is generally compartmentalized to one or two other genres, rather than being spread over the entire genre space.

Thus, we wish to present these two novel approaches as suitable schemes for solving the genre-classification problem.

References

[1] *Elias Pampalk*Computational models of music similarity and their application in Music Information Retrieval; PhD Dessertation, TU-Wien, Faculty for Informatics, 2006.

[2] *Ailon, Nir and Chazelle, Bernard.* Faster Dimension Reduction, Communication of the ACM, Vol. 53, No. 2, February 2010; doi:10.1145/1646353.1646379

[3] *Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra and Inderjit S. Dhillon.* Information Theoretic Metric Learning, ICML June 2007, 209-216, http://www.cs.utexas.edu/ pjain/itml/

[4] *L.J.P. van der Maaten and G.E. Hinton.* Visualizing High-Dimensional Data Using t-SNE. Journal of Machine Learning Research 9(Nov):2579-2605, 2008, http://lvdmaaten.github.io/tsne/

[5] *McFee, B. and Lanckriet, G.R.G.*Learning multi-modal similarity, Journal of Machine Learning Research (JMLR) 2011.

[6] *McFee, B., Barrington, L., and Lanckriet, G.R.G.*Learning content similarity for music recommendation, IEEE Transactions on Audio, Speech and Language Processing, 2012.

[7] *M. Hein, J.-Y. Audibert.* Intrinsic dimensionality estimation of submanifolds in Euclidean space, In L. de Raedt and S. Wrobel, editors, Proceedings of the 22nd International Conference on Machine Learning (ICML 2005), 289 - 296, ACM press, 2005, http://www.ml.uni-saarland.de/publications.htm

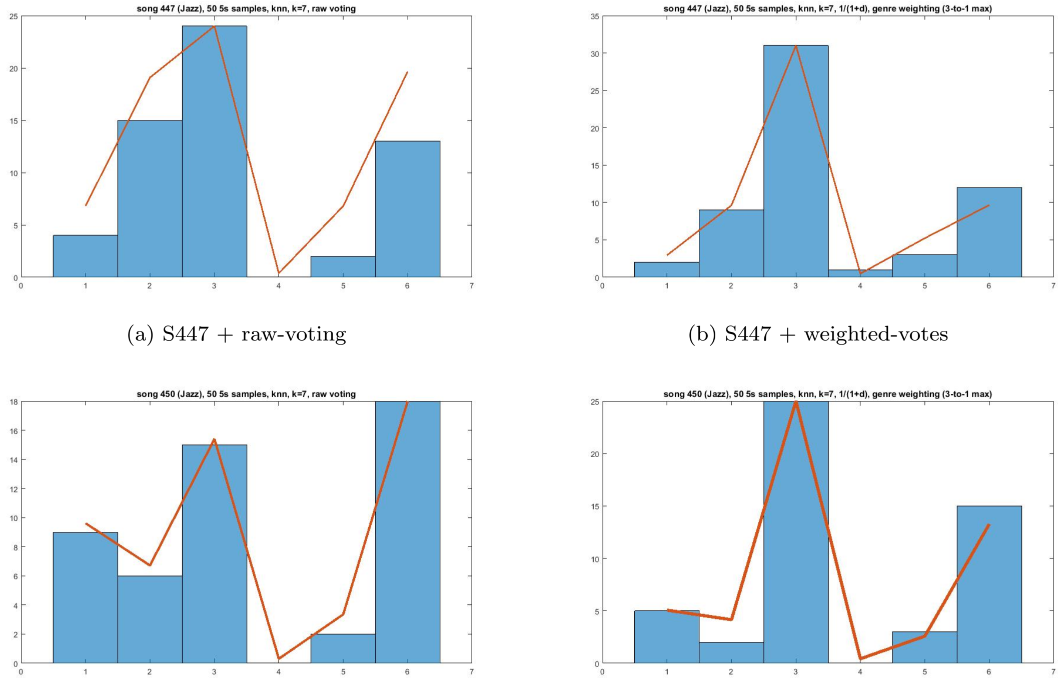
[8] *Non-normalized Laplacian on Weighted Graphs* http://ecee.colorado.edu/ fmeyer/class/ecen5322/laplacian.pdf

10 Appendix: Results of Experiments

10.1 5-second song samples algorithm

An initial validation of this approach was run on two songs in the Jazz category, as it is one of the most problematic genres to accurately identify with our other algorithms. Song 450 was a song that had three of its seven neighbors as other Jazz songs in the baseline experiment (Mean MFCCs plus covariance) and was correctly identified as Jazz in that experiment. Song 447 on the other hand, had zero of its seven neighbors as other Jazz songs and thus was not correctly identified as Jazz in the baseline experiment. Genre classification is shown in a pair of histograms for each song, one for raw voting of the k-NN (k = 7) and one for weighted voting of the k-NN. Distance weighting of $1/(1+d)$ is used, and genre weighting of $1/\sqrt{(nGenreSamples)}$ [this gives a maximum of 3-to-1 voting power between genres] are used in the weighted cases. We also show the votes per song sample for the 50 samples for both test songs, in both raw and weighted voting cases. The results show that genre-weighting is still an important consideration despite the increase in total sample size of the audio space - song 450 is classified as world using raw voting, whereas both songs are clearly assigned to jazz when weighting is used. We also see that there is a not a significant negative effect on genre-classification when assigning each sample to a genre. Most interestingly, it can be seen from the per-sample voting of both songs that, in the weighted voting case at least, a random selection of say 3-5 consecutive song samples would more often than not correctly characterize these songs as jazz. This means that this methodology has significant potential to use in an on-the-fly song comparison/classification application such as the song-identification mobile app Shazam.

When running this 5-second song sample through the cross-validation, we encountered long computation times, though the algorithm was fairly quick (10 seconds) on a per-song basis. Therefore we only present our results with one cross validation step. Results are given for both distance weighting only, and distance plus genre weighting (our focus of discussion below). Overall the algorithm achieves good results in the 72% to 75% range for all tested songs. However this is distributed unevenly between genres. However, an interesting result to note is that for the genres with poor results (Jazz and Rock/Pop), the genres with which they are confused is highly compartmentalized compared to our other approaches. Rock/Pop songs are correctly categorized only 39% of the time, however it is confused with Metal/Punk 33% of the time, together accounting for over 70% of the total Rock/Pop songs (with Electronic this number jumps to over 85%). A similar result holds for Jazz/Blues (49% correct), which is only confused with Electronic (35%) and Classical (15%). This suggests that with further analysis and work we could identify some binary classifiers between these confused genres to improve our total accuracy.



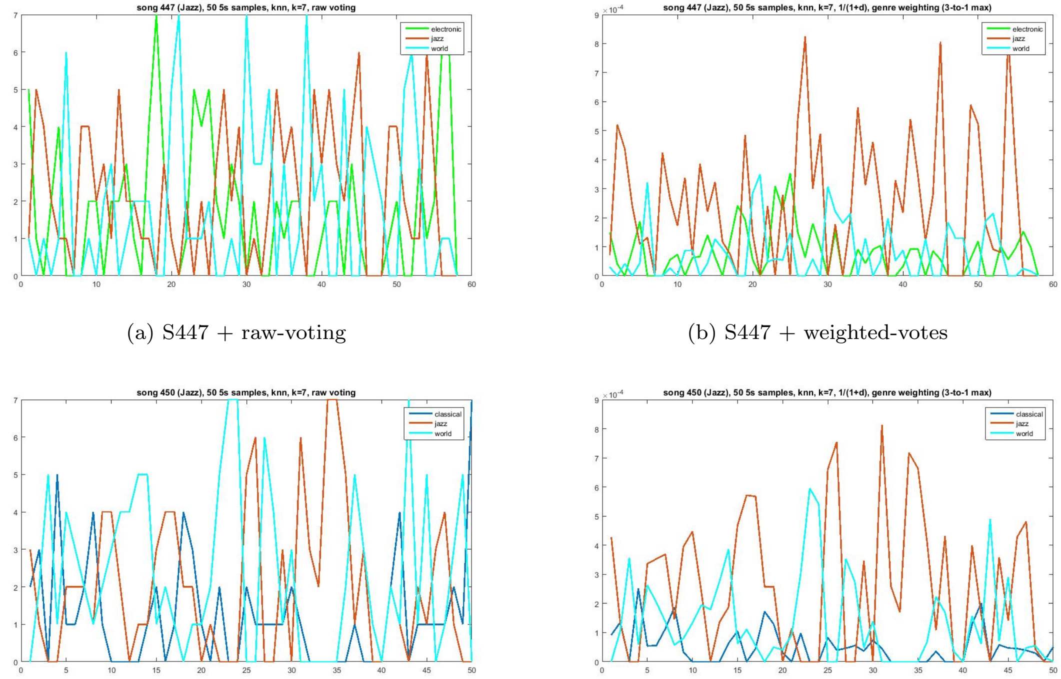
(a) S447 + raw-voting

(b) S447 + weighted-votes

(c) S450 + raw-voting

(d) S450 + weighted-votes

Figure 5: Histogram bars show the sum of votes over all song samples per genre. Lines show the votes when assigning each sample to a genre.



(a) S447 + raw-voting

(b) S447 + weighted-votes

(c) S450 + raw-voting

(d) S450 + weighted-votes

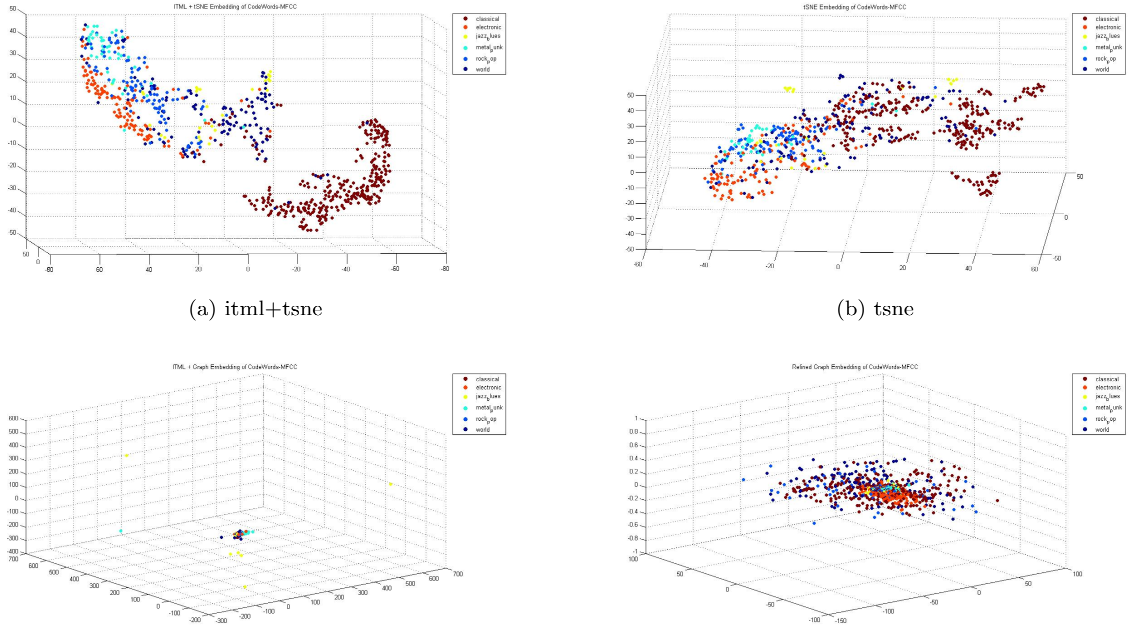
Figure 6: Shows the votes per sample for each song, showing only the votes for the known genre and the two other genres getting the most votes for this song.

10.2 MFCC-Codeword-Histogram of Songs

The plots of the experiments using the mfcc-codeword-histogram give a visual representation of the distribution of the songs dataset in the 3 dimensional space. The clustering algorithm used a combination of k-NN and weighting of votes based on the total number of songs of the same genre in the database. Visually, we identify the pipeline comprising of the following:

1. Code-word Histograms of the MFCCs
2. Information Theoretic metric learning in the high-dimensional space
3. t-Distributed Stochastic Neighborhood Embedding
4. k-NN Clustering with weighted votes

provided a reduced dimensional representation of the dataset that helped obtain a qualitative idea about the nature of the query songs, in addition to providing favourable classification accuracy. The confusion matrix and the probability of successful classification for the different genres is provided in the dataset attached. The plots also highlight the importance of choosing a good metric-learning mechanism at the high-dimensional regime, to ensure that the dimensional reduction techniques used later, can benefit from the better approximation of the distance between the songs, in comparison to an arbitrary choice of euclidean distances.



(a) itml+tsne

(b) tsne

(c) itml+graph

(d) graph

Figure 7: Plot of the structure of dataset obtained by using the following pipeline: mfcc-code-word histogram features and (a)itml and tsne; (b) tsne alone; (c) imtl and graph embedding; (d) graph embedding alone.