Music Information Retrieval: Genre Classification

Sama Shrestha, Victoria Li

December 9, 2016

1 Introduction

Music Information Retrieval (MIR) system has been an active area of research due to the demand of curating digital music efficiently using algorithms. The fast shift in the music industry to a digital platform has created a need to develop tools to search a large digital music collection by content and to extract information for fast organizing of such music. The purpose of this project is to explore and develop an MIR system that is capable of classifying music according to its genre. At a time when musical genre classification was mostly performed manually, researchers started proposing algorithms for automated classification of audio signals based on certain feature sets in the songs [1, 2]. Specially since the International Conference on Music Information Retrieval took place in 2000, various approaches to music genre classification have been proposed. The biggest challenge that many of these approaches face is striking a balance between high accuracy and computational complexity of the method. With the soaring popularity of on the go online music platforms such as Spotify and Apple Music, the necessity for faster algorithms to correctly classify genres, suggest new music based on playlist history or put together playlists based on similarity is bigger then ever.

In this project, we outline our approach to the music genre classification problem. We explore several dimension reduction techniques and a combination of representations for the notion of distances between songs. We used the results of this pre-processing in several macine learning algorithms such as k-nearest neighbors, Naive Bayes and Graph Laplacian. The goal was to well train our classifiers to classify a new set of songs whose genres are unknown to us. Our algorithm was developed using the training data set that was also used in the conference containing 729 songs categorized into 6 broad genres: classical (320), electronic (114), jazz/blues (26), metal/punk (45), rock/pop (102) and world (122)[3].

2 Distance in song-space

We used various distance metrics to quantify the notion of distance between songs. After extracting key features from each song as described in detail in Section 3, we used a frame-based clustering approach by fitting each song with both a mixture of 30 Gaussian models (GMM) and a single Gaussian model (G1) over the time frame for each MFCC coefficent. The clustering was performed using the MA toolbox developed by Elias Pampalk [5]. We chose these two clustering methods based on Pampal's thesis which talks about their performance in capturing key features in our song space [4]. We chose a combination of the following distances depending on the results of our dimension reduction and clustering approach.

2.1 Euclidean norm

When appropriate, we used the Euclidean (Frobenius) norm to calculate the absolute distance between two vectors or matrices. The Euclidean norm is defined as,

- $\sqrt{\sum_{k=1}^{n} |x_k|^2}$ for a vector $x = [x_1, x_2, ..., x_n]$.
- $\sqrt{\left(\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2\right)}$ for a matrix $A_{m \times n}$.

2.2 Kullback-Leibler divergence

Euclidean norms may not be useful in the case when we have information about probability distribution of songs instead of a vector or matrix representation. Thus, we used the Kullback-Leibler (KL) divergence to quantify differences between our multivariate Gaussian mixture model and single Gaussian model for songs. For the single Gaussian model (G1), we define the KL divergence between two gaussians, f and g by,

$$D(f||g) = \frac{1}{2}\left[\log\frac{|\Sigma_g|}{|\Sigma_f|} + Tr[\Sigma_g^{-1}\Sigma_f] - d + (\mu_f - \mu_g)^T\Sigma_g^{-1}(\mu_f - \mu_g)\right] \tag{1}$$

Here, μ represents the vector of means, d is the length of the μ vector and Σ represents the co-variance matrix of each of the random variables f and g. However, for Gaussian mixture models, no such closed form representation of the KL divergence exists. Hence, we used the variational approximation of the KL divergence using likelihoods outlined in [14] for our GMM models.

3 Dimension reduction

3.1 MFCCs

The songs are given as mono channel WAV files with sampling rate f_s = 11025 Hz. We converted each song to a matrix of the first 20 MFCCs versus time. MFCC stands for Mel Frequency Cepstrum Coefficient, and gives the intensity (in dB) of each frequency band (in Mels), using transformations that model the way the human ear perceives sound [4]. Since this is the closest we can get to converting a song to its score, which contains the information used by the human brain to determine genre, we choose to study the MFCC feature. Each song is converted as follows:

1. The 2 minutes from the center of the song are used for analysis. If the song is less than 2 minutes long, the entire song is used.

2. The signal is divided into frames of size 256 samples (~23 ms) each, with 50% overlap.

3. A window function (Hann window) is applied to each frame to reduce spectral leakage.
4. FFT is applied to convert the WAV file to a power spectrum, which plots 129 frequency bins vs. the time frames and indicates intensity (dB) by color.
5. A filter bank is applied to transform the power spectrum to a Mel frequency spectrum, which plots 40 Mel frequency bands (minus the ones exceeding $f_s/2$) vs. the time frames and indicates intensity (dB) by color.
6. DCT is applied to compress the Mel frequency spectrum and replace the 40 frequency bands with 20 MFCCs, plotted vs. the time frames, with coefficient value indicated by color.

The code is provided by the Matlab ma toolbox[5], with some bug fixes by us.

3.2 FJLT

Johnson-Lindenstrauss is a dimension reduction technique that unlike PCA and compressed sensing assumes no prior info on the data, and unlike local sensitivity hashing preserves distances/angles (with low distortion) [6]. It works by projecting n data vectors with dimension d onto a randomly chosen low dimensional subspace of dimension $k = O(\frac{\log n}{\varepsilon^2})$ [7]. This projection is simply the action of a matrix Q on the data vectors. Then the following results hold [7]:

Johnson-Lindenstrauss Lemma

$$\begin{aligned}
&\forall \varepsilon > 0, x \neq 0 \,\epsilon\, \mathbb{R}^n : \\
&\qquad \lambda\{L \,\epsilon\, \text{Grass}(k,n) : \sqrt{\frac{n}{k}}\frac{\|Qx\|}{\|x\|} \geq \frac{1}{1-\varepsilon}\} \leq e^{-n\varepsilon^2/4} + e^{-k\varepsilon^2/4} \\
&\qquad \text{and } \lambda\{L \,\epsilon\, \text{Grass}(k,n) : \sqrt{\frac{n}{k}}\frac{\|Qx\|}{\|x\|} \leq 1-\varepsilon\} \leq e^{-n\varepsilon^2/4} + e^{-k\varepsilon^2/4} \\
&\qquad\qquad \text{where } \lambda \text{ is the Haar measure on Grass}(k,n). \qquad (2)
\end{aligned}$$

Johnson-Lindenstrauss Theorem

$$\begin{aligned}
&\forall \varepsilon > 0, x_1, ..., x_N \,\epsilon\, \mathbb{R}^n : \\
&\qquad k \,\epsilon\, \mathbb{N} \text{ s.t. } N(N-1)(e^{-k\varepsilon^2/4} + e^{-n\varepsilon^2/4}) \leq \frac{1}{200} \text{ (e.g. } k \geq \frac{8\ln(20N)}{\varepsilon^2}) \\
&\qquad \Rightarrow \lambda\{L \,\epsilon\, \text{Grass}(k,n) : \forall i,j = 1,...,N, (1-\varepsilon)\|x_i - x_j\| \leq \\
&\qquad\qquad \sqrt{\frac{n}{k}}\|Q(x_i - x_j)\| \leq \frac{1}{1-\varepsilon}\|x_i - x_j\|\} \geq 0.995. \qquad (3)
\end{aligned}$$

See [7] for proofs. The running time of the algorithm is $O(dk)$ per data vector [6]. However, selecting a random subspace is computationally costly [7].

For increased computational efficiency, we use FJLT for dimension reduction. It projects onto a low dimensional subspace of dimension $k = c\varepsilon^{-2}\log n$ for some large enough c[6, 8]. It assumes that d is a power of two, $d > k$, and $n\Omega d = \Omega(\varepsilon^{-1/2})$ [6] where Ω is the Bachmann-Landau notation [9]. This time the projection matrix is $Q = PHD$ where:

- P is a $k \times d$ matrix with $P_{ij} \sim N(0, 1/q)$ with probability q and $P_{ij} = 0$ with probability $1 - q$, $q = \min\{\Theta(\frac{\varepsilon^{p-2}(\log n)^p}{d}), 1\}$.

- H is a $d \times d$ normalized Walsh-Hadamard matrix.

- D is a $d \times d$ diagonal matrix where D_{ij} takes the values -1 and 1 with probability 1/2. [6]

A recursion for H is given by [8]:

$$H_2 = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \tag{4}$$

$$H_{2^{k+1}} = \frac{1}{\sqrt{2}} \begin{bmatrix} H_{2^k} & H_{2^k} \\ H_{2^k} & -H_{2^k} \end{bmatrix} \tag{5}$$

Then the following result holds [6]:

FLJT Lemma

For $x_1, ..., x_n \,\epsilon\, \mathbb{R}^d, \varepsilon < 1, p \,\epsilon\, \{1, 2\}$, and FJLT projection matrix Q, the following two events occur with probability at least 2/3:

1. For any x_i, $(1 - \varepsilon)\alpha_p \|x\|_2 \leq \|Qx\|_p \leq (1 + \varepsilon)\alpha_p \|x\|_2$ where $\alpha_1 = k\sqrt{2\pi^{-1}}$ and $\alpha_2 = k$.
2. The mapping $Q : \mathbb{R}^d \to \mathbb{R}^k$ requires $O(d\text{log}d + \min\{d\varepsilon^{-2}\text{log}n, \varepsilon^{p-4}\text{log}^{p+1}n\})$ operations.

See [6] for proof. The running time for this is $O(d \log d + qd\varepsilon^{-2} \log n)$ per data vector in the worst-case [6]. This is at most $\max(q, \varepsilon^{-2})$ times the running time of Johnson-Lindenstrauss, and is a reduction from the running time of Johnson-Lindenstrauss. We will implement FJLT by Gabriel Krummenacher's Python code [10] in combination with Matlab's Python import functionality [11].

4 Statistical learning

After reducing the dimension of our data and defining several notions of distance metrics, we went forward with training several machine learning algorithms to correctly classify unknown genres (testing data) using a handful of known ones (training data). We describe the three machine learning algorithms that we implemented on our data. They range from simple but widely used classifiers such as the k-nearest neighbor to more mathematically sophisticated ones such as the Graph Laplacian. We outline the results of our experiments and the cross-validation techniques used to assess the accuracy of the classifiers in Section 5.

4.1 k-nearest neighbors (kNN)

We used the non-parametric k-nearest neighbor algorithm as a starting point since it is so commonly used for classification and regression due to its relative simplicity. It can also prove to be a useful exploratory tool since being non-parametric implies it does not make assumptions about the underlying data distribution. This algorithm uses some notion of distance defined in our space to decide what neighbors a node has. The k is specified by the user and should be an optimal number based on the data. The number k is what distinguishes how many neighbors influence the classification of the node or in our case song that we are looking at. A key property of the k-nearest neighbor classification is that it stores and uses all of the training data in order to attempt to classify the testing data instead of constructing a generalized model. An outline of the algorithm is as follows:

- Given a song x, find its k nearest neighbors using the notion of distance define above
- Determine the genre or class of x based on the genres of its neighbors. This determination could be based on majority, weighed according to the distance or even probabilistic voting.

4.2 Naive Bayes

We used the Naive Bayes classifier on the data which differs from the k-nearest neighbors classification in a way that it uses the notion of 'probability' rather than a notion of some distance metric to classify nodes. This algorithm is based on Bayes' theorem which states that,

$$P(C_k|x) = \frac{P(x|C_k)P(C_k)}{P(x)}$$

where C_k is the k-th class and x is our training data. Thus, this essentially translates to the known form of Bayesian framework: *posterior* = *prior* × *likelihood.* The name itself is derived based on the 'naive' assumption of independence between every pair of features. We implemented a Gaussian Naive Bayes algorithm whose posterior can be written as:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right) \tag{6}$$

Here, x_i are independent pairs of feature vectors as per the naive assumption and $y = 1, .., k$ is our class variable with k components. Thus, the parameters σ_y and μ_y can be estimated using a maximum likelihood framework. An outline of the algorithm is as follows:

- Use training data to compute the posterior probability $P(x_i|y)$ for each class or genre.
- Find the vector of parameters $\hat{\theta}$ that maximizes the posterior probability.
- For a new testing data, the output label $\hat{y}$ of the classifier is given by,

$$\hat{y} = \arg\ \max_{y\in 1,..,k} P(x_i, y|\hat{\theta})$$

4.3 Graph Laplacian

The graph Laplacian is used for spectral clustering: It is simple to implement, efficiently solved by standard linear algebra methods, and very often outperforms traditional clustering methods such as k-means and single linkage [13]. Given a graph $G = (V, E)$ with n nodes and adjacency matrix W, it can be shown that partitioning the nodes (songs) into k nearly disconnected components (genres) is equivalent to partitioning the indices (by their corresponding entry values) of the first k eigenvectors (corresponding to the k smallest eigenvalues) of the graph Laplacian matrix [13]. Here, a disconnected component is defined as a subset $A_i \subset V$ that is connected but has no connections between vertices in A_i and $\overline{A_i}$. Specifically, define the degree matrix D as the diagonal matrix where d_{ii} is the degree of the i^{th} node, $d_i = \sum_{j=1}^{n} w_{ij}$. Then the Laplacian is given by $L = D - W$. It can be shown that L is equivalent to the discretized Laplacian operator [13]. If G perfectly consisted of k disconnected components $A_1, ..., A_k$, then the 0 eigenvalue would have multiplicity k, and the corresponding eigenspace would be spanned by the indicator vectors $\mathbb{1}_{A_i}$ of the components. The indicator vectors are defined as $\mathbb{1}_{A_i,k} = 1$ if node $k \,\epsilon\, A_i$ and 0 else.

If G is any undirected graph with nonnegative weights, then the following algorithm partitions the graph into clusters $A_1, ..., A_k$ such that RatioCut$= \sum_{i=1}^{k} \frac{\text{cut}(A_i,\overline{A_i})}{|A_i|}$, cut$(A, B) = \sum_{i \,\epsilon\, A, j \,\epsilon\, B} w_{ij}$, $|A| = \#$ of vertices in A, is minimized:

1. Compute the first k eigenvectors $v_1, ..., v_k$ corresponding to the k smallest eigenvalues of L.

2. Form the matrix V with v_i as columns.

3. Let y_i be the vector corresponding to the i^{th} row of V.

4. Cluster $y_1, ..., y_n$ into clusters $C_1, ..., C_k$ using the k-means algorithm.

5. Return clusters $A_i = \{j | y_j \,\epsilon\, C_k\}$.

Since our Graph Laplacian program can only cluster classical vs. non-classical, we use $k = 2$. L is the unnormalized Laplacian, but one can also use the normalized Laplacian given by $L_{rw} = D^{-1}L$, or with slight modifications to the algorithm the symmetric normalized Laplacian given by $L_{sym} = I - D^{-1/2}WD^{-1/2}$. Using either of the normalized Laplacians partitions the graph into clusters $A_1, ..., A_k$ such that Ncut$= \sum_{i=1}^{k} \frac{\text{cut}(A_i,\overline{A_i})}{\text{vol}(A_i)}$, vol$(A) = \sum_{i \,\epsilon\, A} d_i$, is minimized. [13]

We use a normalized Laplacian because although both RatioCut and Ncut minimize between-cluster similarity, only Ncut maximizes within-cluster similarity. Since the only difference between L_{sym} and L_{rw} is that L_{sym} is symmetric (and has eigenvectors $D^{1/2}$ times those of L_{rw}), we use Singular Value Decomposition (SVD) on L_{sym} to calculate the eigenvectors of L_{rw}. Since L_{sym} is normal but L_{rw} isn't, we apply SVD to L_{sym}. Instead of normalizing the eigenvectors of L_{sym} as in [13], we simply multiply $D^{1/2}$ by the eigenvectors of L_{sym} to obtain the eigenvectors in order of L_{rw}. The eigenvalues of L_{rw} are the same as the eigenvalues of L_{sym}. [13]

5 Experiments

We evaluated the performance of our statistical learning algorithms using confusion matrices which is a standard visualization approach in machine learning. For this project, we have 6 classes or genres of music so our confusion matrix was a 6×6 matrix. For instances when we simply classified classical versus non-classical, we had a 2×2 confusion matrix. Each row of this matrix represents the true genre while the columns has the predicted genres. The c_{ij} in our confusion matrix C, is the number of songs of genre i that were classified by our algorithm to be genre j. Thus, if a classifier had perfect performance, the confusion matrix would be a diagonal matrix.

We outline the 5-fold 10 times cross-validation that we performed for each of our classification:

1. First, organize all 729 songs into their respective genres.
2. Randomly divide each genres into 5 subsets of approximately equal size.
3. Select the first subset from each genre and combine them to form a testing set.
4. Combine the remaining songs to form a training set and train your classifier.
5. Test the performance of the algorithm on the testing set, and record the confusion matrix.
6. Repeat steps 3-5 with each of the subset.
7. Repeat steps 2-6 10 times recording confusion matrices each time.
8. Average element-wise over all 50 confusion matrices to get $C_{i,j}$.
9. Record the associated standard deviation matrix as well.

5.1 *k*NN with Single Gaussian Model

The single gaussian model was the first clustering approach that we used with a combination of the k-nearest neighbor classifier. In this classification algorithm, the first 20 MFCC coefficients over 1653 time frames for each song was fitted with a Gaussian distribution. Thus, the resulting features of each song can be written as a 1×20 mean vector and a full 20×20 matrix of co-variances. This clustering of each song took a total of 10 seconds for all 729 songs, so it was a very quick process. We used the KL divergences as defined in Equation (1) to calculate the distance between songs. In order to achieve the best accuracy, we explored different values of k for this method. Figure 1 shows the results of the parameter exploration, where interestingly, we get the best accuracy with just one neighbor. Thus, based on the results of this figure, we coded the classifier to look at the closes neighbor, $k = 1$. To investigate, how the genre specific accuracy performed when we're looking at more than one neighbor, we also ran the classifier with $k = 6$. Tables 1 and 3 show the results of the two classifiers. Tables 2 and 4 show the standard deviations associated with each of the 50 iterations. The total computational time for the $k = 1$ and $k = 6$ cases were 526 and 537 seconds respectively.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.9531**	0.0094	0.0000	0.0000	0.0031	0.0344
Electronic	0.0534	**0.6213**	0.0174	0.0352	0.2462	0.0265
Jazz/Blue	0.1533	0.0733	**0.6133**	0.0000	0.1200	0.0400
Metal/Punk	0.0000	0.0889	0.0000	**0.4889**	0.4222	0.0000
Rock/Pop	0.0200	0.0891	0.0000	0.0573	**0.7945**	0.0391
World	0.2372	0.1231	0.0167	0.0083	0.2628	**0.3519**

Table 1: Mean of Confusion matrix for *k*NN-G1 with $k = 1$.

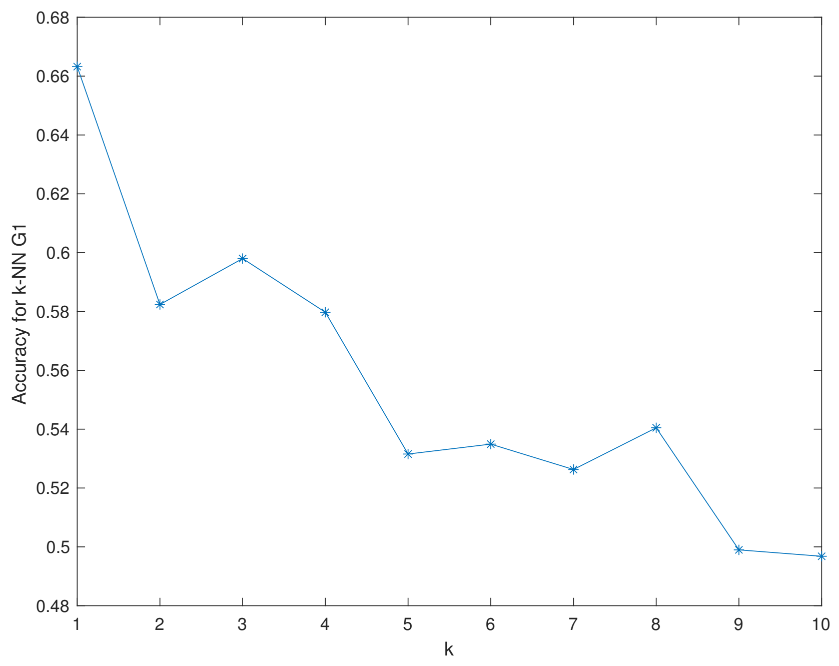

Figure 1: Exploration of different number of neighbors (k) to consider with respect to the accuracy among all the genres for k-nearest neighbor classifier on the single Gaussian clustering approach.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0264	0.0126	0.0000	0.0000	0.0063	0.0209
Electronic	0.0531	0.1377	0.0215	0.0330	0.0685	0.0219
Jazz/Blue	0.1500	0.0913	0.0269	0.0000	0.1616	0.0808
Metal/Punk	0.0000	0.1309	0.0000	0.2199	0.1100	0.0000
Rock/Pop	0.0404	0.0596	0.0000	0.0548	0.0564	0.0376
World	0.1125	0.0534	0.0206	0.0168	0.0698	0.0516

Table 2: Standard deviation of Confusion matrix for kNN-G1 with $k = 1$.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.9844**	0.0094	0.0000	0.0000	0.0031	0.0031
Electronic	0.0881	**0.5700**	0.0000	0.0000	0.3328	0.0091
Jazz/Blue	0.5067	0.1133	**0.1933**	0.0000	0.1867	0.0000
Metal/Punk	0.0000	0.1111	0.0000	**0.4444**	0.4444	0.0000
Rock/Pop	0.0791	0.0600	0.0000	0.0500	**0.8018**	0.0091
World	0.5237	0.1135	0.0000	0.0000	0.2391	**0.1237**

Table 3: Mean of Confusion matrix for kNN-G1 with $k = 6$.

5.2 kNN with Gaussian Mixture Model

Next, we looked at using the popular frame-clustering approach, Gaussian Mixture Model (GMM) with a cluster of 30 with a combination of the k-nearest neighbor classifier. In this classification algorithm, the first 20 MFCC coefficients over the same 1653 time frames for each

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0141	0.0077	0.0000	0.0000	0.0063	0.0063
Electronic	0.0297	0.0656	0.0000	0.0000	0.0569	0.0184
Jazz/Blue	0.1735	0.1588	0.0135	0.0000	0.1671	0.0000
Metal/Punk	0.0000	0.0710	0.0000	0.1004	0.1420	0.0000
Rock/Pop	0.0411	0.0495	0.0000	0.0782	0.0868	0.0184
World	0.0899	0.0945	0.0000	0.0000	0.1160	0.0392

Table 4: Standard deviation of Confusion matrix for kNN-G1 with $k = 6$.

song was fitted with a mixture of 30 Gaussians. Thus, the resulting features of each song can be written as a 30×20 mean vector and a diagonal 30×20 matrix of co-variances. It took 392 seconds of CPU time to fit the 30 GMMs to all 729 songs. While this time was manageable for our small dataset, this process could take a long time for bigger datasets and may not be feasible to use. Additionally, it took around 1800 seconds of CPU time to run the classifier with a majority of the time spent in estimating the KL divergence between two GMMs of songs. This method was our most computationally expensive classifier. We ran this algorithm with $k = 10$ which gave us the results shown in Tables 5 and 6.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.9688**	0.0031	0.0000	0.0000	0.0000	0.0281
Electronic	0.0443	**0.6850**	0.0000	0.0000	0.2360	0.0348
Jazz/Blue	0.3333	0.2000	**0.2667**	0.0000	0.1600	0.0400
Metal/Punk	0.0000	0.1778	0.0000	**0.2444**	0.5778	0.0000
Rock/Pop	0.0200	0.1545	0.0100	0.0500	**0.7555**	0.0100
World	0.3763	0.1968	0.0000	0.0000	0.2308	**0.1962**

Table 5: Mean of Confusion matrix for kNN-GMM with $k = 10$.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0141	0.0063	0.0000	0.0000	0.0000	0.0118
Electronic	0.0291	0.0731	0.0000	0.0000	0.0696	0.0329
Jazz/Blue	0.2254	0.2213	0.2371	0.0000	0.1979	0.0808
Metal/Punk	0.0000	0.1522	0.0000	0.0840	0.1796	0.0000
Rock/Pop	0.0404	0.0714	0.0202	0.0639	0.0604	0.0202
World	0.0761	0.0411	0.0000	0.0000	0.0541	0.0796

Table 6: Standard deviation of Confusion matrix for kNN-GMM with $k = 10$.

5.3 kNN with Fast Johnson-Lindenstrauss Theorem (FJLT)

We decided to use lower dimensional feature vectors of each song resulting from the application of the FJLT and perform a kNN algorithm on them with Euclidean distance metric. Since we experiment with several different dimensions of FJLT, we explored the accuracy of k-NN with $k = 10$ for all the different dimensions retained from applying the FJLT with different accuracy. Figure 2 shows the variations in the accuracy for different dimensions. Since, the accuracy didn't change significantly with the different dimensions, we simply chose the dimension (14870) that gave us the highest accuracy for our kNN classifier. The results of this classification are given in Tables 7 and 8. This classifier took 48.8 seconds of CPU time to run, so the fastest of all three of our kNN models.

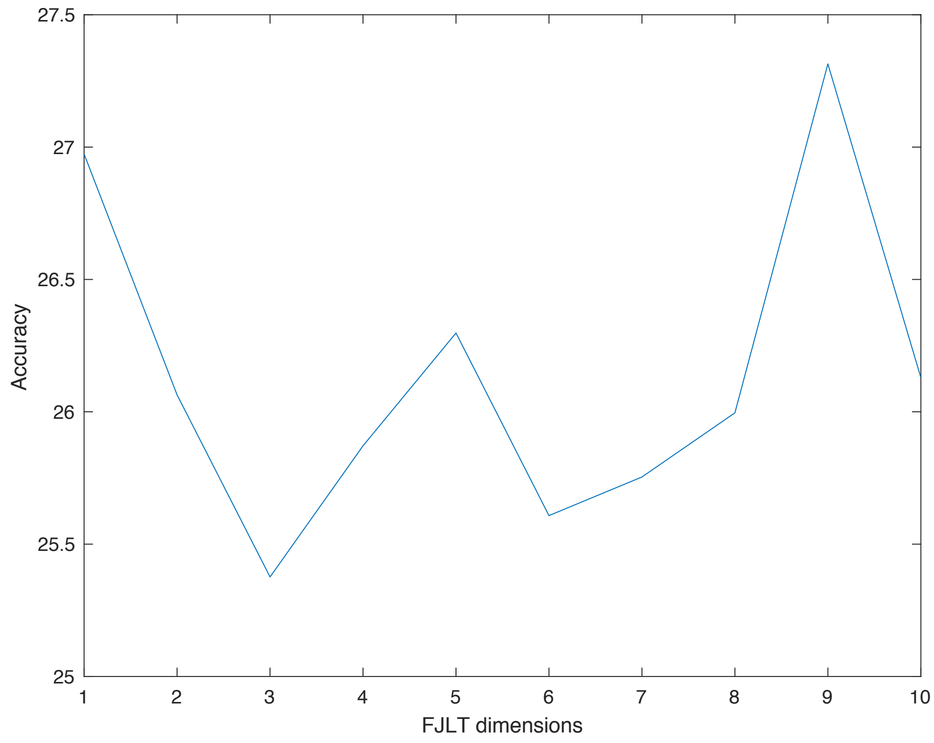

Figure 2: Exploration of average accuracy over all all genres for different dimensions retained after the FJLT dimension reduction with a kNN classifier with $k = 10$. The x-label indexes represent the following dimensions after FJLT: 7670, 8570, 9470, 10370, 11270, 12170, 13070, 13970, 14870 and 15770 respectively.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.8416**	0.0391	0.0016	0.0000	0.0187	0.0991
Electronic	0.3953	**0.1004**	0.0052	0.0983	0.1780	0.2227
Jazz/Blue	0.5973	0.1853	**0.0000**	0.0000	0.1373	0.0800
Metal/Punk	0.1489	0.2000	0.0000	**0.1489**	0.4067	0.0956
Rock/Pop	0.2686	0.1173	0.0000	0.1361	**0.3687**	0.1093
World	0.5708	0.1312	0.0000	0.0042	0.0979	**0.1960**

Table 7: Mean of Confusion matrix for kNN classifier with Euclidean distance for FJLT dimension reductions.

5.4 Naive Bayes with Single Gaussian Model

The next method we tried was the Naive Bayes with the single Gaussian model from Section 5.1. The 1×20 mean vectors for each song in a given genre were used as feature vectors

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0572	0.0203	0.0047	0.0000	0.0306	0.0425
Electronic	0.1251	0.0869	0.0143	0.0614	0.1004	0.0499
Jazz/Blue	0.1182	0.1565	0.0000	0.0000	0.1321	0.1616
Metal/Punk	0.1067	0.1634	0.0000	0.0942	0.2028	0.0810
Rock/Pop	0.0452	0.0681	0.0000	0.0765	0.0317	0.0838
World	0.0313	0.0497	0.0000	0.0126	0.0610	0.0473

Table 8: Standard deviation of Confusion matrix for kNN classifier with Euclidean distance for FJLT dimension reductions.

to calculate the posterior probabilities for that genre. After obtaining the mean vectors, the computational time for the Naive Bayes classifier was almost negligible (8 seconds). The results of this classifier are given in Tables 9 and 10.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.8063**	0.0219	0.0000	0.0000	0.0656	0.1062
Electronic	0.0613	**0.5000**	0.0087	0.0439	0.2810	0.1051
Jazz/Blue	0.1933	0.1867	**0.3533**	0.0000	0.1867	0.0800
Metal/Punk	0.0000	0.0222	0.0000	**0.8000**	0.1778	0.0000
Rock/Pop	0.0391	0.1445	0.0191	0.3055	**0.4818**	0.0100
World	0.3936	0.1731	0.0160	0.0000	0.1545	**0.2628**

Table 9: Mean of Confusion matrix for Naive Bayes with feature vectors from a single Gaussian model.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0354	0.0161	0.0000	0.0000	0.0392	0.0306
Electronic	0.0212	0.0439	0.0176	0.0393	0.0777	0.0813
Jazz/Blue	0.1812	0.1671	0.1588	0.0000	0.1077	0.1616
Metal/Punk	0.0000	0.0449	0.0000	0.1309	0.1522	0.0000
Rock/Pop	0.0376	0.0984	0.0237	0.0641	0.1166	0.0202
World	0.0194	0.0437	0.0199	0.0000	0.0563	0.0248

Table 10: Standard deviation of Confusion matrix for Naive Bayes with feature vectors from a single Gaussian model.

5.5 Naive Bayes with Gaussian Mixture Model

We also used the 30×20 mean vectors obtained from the mixture of 30 Gaussians from Section 5.2 as feature vectors for the Naive Bayes classifier. After obtaining the feature vectors, this classifier took 86.5 second of CPU time to run. Tables 11 and 12 show the results of this classifying algorithm.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	**0.7531**	0.0125	0.0156	0.0000	0.0344	0.1844
Electronic	0.0613	**0.5178**	0.0000	0.0522	0.2462	0.1225
Jazz/Blue	0.1533	0.1133	**0.3467**	0.0000	0.1133	0.2733
Metal/Punk	0.0000	0.0222	0.0000	**0.8444**	0.1333	0.0000
Rock/Pop	0.0491	0.1091	0.0100	0.2955	**0.4382**	0.0982
World	0.3929	0.1545	0.0083	0.0083	0.1885	**0.2474**

Table 11: Mean of Confusion matrix for Naive Bayes with feature vectors from a Gaussian mixture model.

	Classical	Electronic	Jazz/Blue	Metal/Punk	Rock/Pop	World
Classical	0.0642	0.0118	0.0141	0.0000	0.0184	0.0716
Electronic	0.0446	0.0767	0.0000	0.0329	0.0925	0.0641
Jazz/Blue	0.0785	0.0943	0.0785	0.0000	0.0943	0.1655
Metal/Punk	0.0000	0.0449	0.0000	0.2082	0.1650	0.0000
Rock/Pop	0.0320	0.0876	0.0202	0.0680	0.0949	0.0555
World	0.0577	0.0496	0.0168	0.0168	0.0565	0.0909

Table 12: Standard deviation of Confusion matrix for Naive Bayes with feature vectors from a Gaussian mixture model.

5.6 Graph Laplacian

The average accuracies over all 50 confusion matrices for each post-FJLT dimension k is plotted in Figure 3. The best accuracy was 70.22 with k = 13970, which gave the confusion matrix with average in Table 13 and standard deviation in Table 14. As a result, we choose k =

	Classical	Non-classical
Classical	.5444	.4556
Non-classical	.1400	.8600

Table 13: Average Confusion matrix for Graph Laplacian with k = 13970.

	Classical	Non-classical
Classical	.1769	.1769
Non-classical	.1066	.1066

Table 14: Standard deviation of Confusion matrix for Graph Laplacian with k = 13970.

13970.

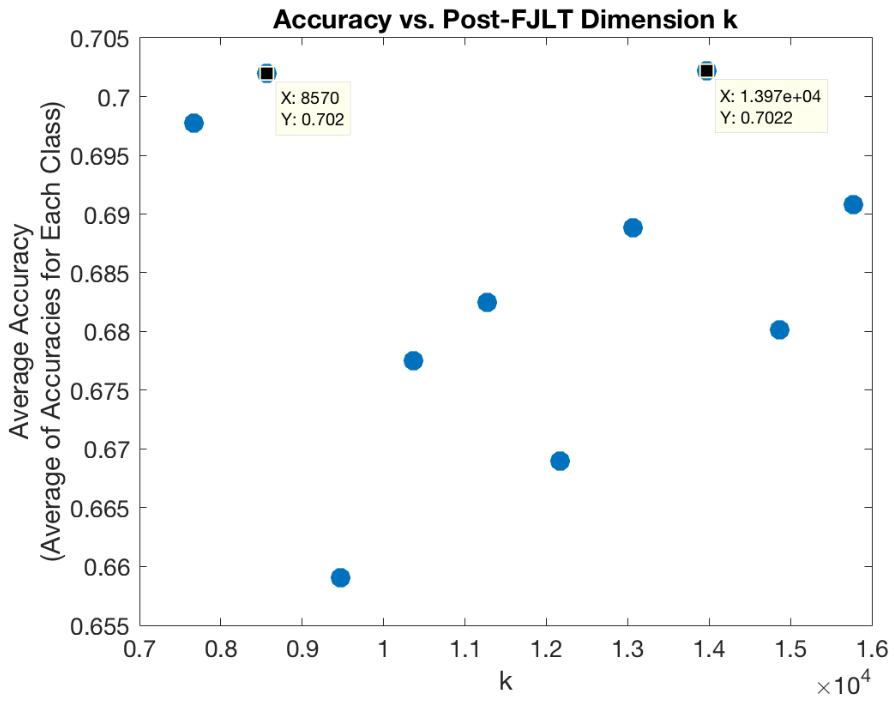

Figure 3: Exploration of different post-FJLT dimension (k) to consider with respect to the accuracy among all the genres for Graph Laplacian approach.

6 Discussion

We had some of our best average accuracy with the kNN models with the single Gaussian and Gaussian Mixture Models. This is not very surprising since it is similar to some of the standard MIR approaches in the MIR literature and is known to perform relatively well in classification. However, the choice of $k = 1$ with which it did really well is not very robust. The reason that it did so well with classical is biased due to the sheer number of training data that contained classical. Some other drawbacks of using the k-nearest neighbors was that it was computationally expensive and it gets worse with larger datasets. It also heavily relied on the notion of distance so the accuracy was definitely affected on how we defined distances in the song space. Additionally, we do not think that the improvement in accuracy from the single Gaussian to the mixture of 30 Gaussians was not significant enough to warrant the increase in computational expense in their use in kNN algorithms.

Our results show that reducing dimensions with the FJLT did not result in better accuracy when used in conjunction with kNN neighbors classifiers and Euclidean distance metric. This could be in part due to the low size of our dataset (n).

Our Naive Bayes classification results did not do as well as we expected. The strong assumption of independence did hurt the accuracy since some of the genres clearly wouldn't be independent in terms of their features like pitch and loudness. However, it did better than the standard kNN approach in classifying Jazz/Blue and Metal/Punk although they had small number of training songs. Thus, we still think that Naive Bayes could work really well with genre classification accompanied with better feature vectors. The trade off between accuracy and computational time was more balanced for this classifier compared to the kNN approach.

Although Graph Laplacian attained the greatest accuracy of 70%, it was only able to classify classical vs. non-classical genres. This was because when told to classify into 6 clusters, it classified all songs as classical except for one in each of the remaining 6 clusters. When told to classify 2 clusters, it classified all songs as classical except for one as non-classical. And so on. The probable reason for this is that there is a bug in our Graph Laplacian or Matlab's kmeans code. When we used the pre-packaged Python code sklearn.cluster.SpectralClustering, we were able to classify 6 clusters with different and sensible results. Thus in future we will use the prepackaged code for clustering by the Graph Laplacian method.

Comparing the run times for the various methods, we obtain:

- FJLT: 100 minutes per song = 10 minutes per fjlt per song

- Graph Laplacian: 5783 s per 729 songs

Thus Graph Laplacian attains speeds similar to the most accurate (but most slow) method to date, G30 (Figure 4). Meanwhile FJLT is much more time consuming.

	G30	G30S	G1
FC	25000	700	30.0
CMS	400	2	0.1

Figure 4: CPU times (ms) per song for Flame Clustering, Cluster Model Similarity [4]

Finally, we present our classification for the 6 genres on the new data set downloaded from the website in Table 15.

Track	**kNN G1**	**kNN G30**	**NB G1**	**NB G30**	**Mode**
1	6	5	1	1	1
2	1	1	1	1	1
3	1	1	1	6	1
4	4	5	4	4	4
5	5	5	5	5	5
6	1	2	5	5	5
7	2	2	5	2	2
8	2	2	1	2	2
9	1	6	1	1	1
10	2	2	2	2	2
11	1	1	6	1	1
12	5	2	2	6	2
13	1	1	1	1	1
14	1	1	1	1	1
15	3	3	5	3	3
16	6	6	1	1	1
17	1	1	1	1	1
18	2	5	5	2	2
19	1	1	1	1	1
20	1	1	1	1	1
21	1	1	1	6	1
22	1	1	1	1	1
23	3	2	1	1	1
24	1	1	6	6	1
25	1	5	1	1	1
26	1	1	1	6	1
27	2	5	2	4	2
28	4	5	4	4	4
29	5	5	4	4	4
30	1	1	6	6	1
31	1	6	1	1	1
32	1	1	6	1	1
33	5	5	4	5	5
34	2	2	4	4	2
35	1	1	1	1	1

Table 15: Contest data classification. The genres are (1)classical, (2)electronic, (3)jazz/blue, (4)metal/punk, (5)rock/pop, (6)world

References

[1] Foote, J.T. (1997) Content-based retrieval of music and audio. *Proc. SPIE 3229, Multimedia Storage and Archiving Systems II.* **3229**, 138-147.

[2] Tzanetakis, G. & Cook, P. (2002) Musical genre classification of audio signals. *IEEE Trans-*

actions on Speech and Audio Processing. **10**(5): 293-302.

[3] Cano, P., Gómez, E. et al. (2006) *ISMIR 2004 audio description contest* (MTG-TR-2006-02). Music Technology Group, Universitat Pompeu Fabra.

[4] Pampalk, E. (2006) Computational models of music similarity and their application in music information retrieval (Doctoral dissertation). Technischen Universität Wien, Vienna, Austria.

[5] Pampalk, E. (2004) A Matlab Toolbox to Compute Similarity from Audio Proceedings of the ISMIR International Conference on Music Information Retrieval (ISMIR'04), Barcelona, Spain, 2004.

[6] Ailon, N. & Chazelle, B. (2010) Faster dimension reduction. *Communications of the ACM.* **53**(2), 97-104.0) Mammalian cell transfection: the present and the future. *Analytical and Bioanalytical Chem.* **397**, 3173-3178.

[7] http://ecee.colorado.edu/~fmeyer/class/ecen5322/concentration-9.pdf

[8] http://ecee.colorado.edu/~fmeyer/class/ecen5322/nneighbors.pdf

[9] https://en.wikipedia.org/wiki/Big_O_notation#Big_Omega_notation

[10] https://github.com/gabobert/fast-jlt

[11] https://www.mathworks.com/help/matlab/call-python-libraries.html

[12] Kullback, S. & Leibler, R.A. (1951) On information and sufficiency. *The Annals of Mathematical Statistics.* **22**(1): 79-86.

[13] von Luxburg, U. (2006) *A tutorial on spectral clustering* (Technical Report No. TR-149). Max-Planck-Institut für biologische Kybernetik.

[14] Hershey, J and Olsen, P. (2007). Approximating the Kullback Leibler divergence between Gaussian mixture models. 2007 IEEE International Conference on Acoustics, Speech and Signal Processing-ICASSP'07. Vol. 4. IEEE, 2007.