Music Information Retrieval and Genre Classification

Tomas Rojo

Introduction

Machine Learning has been used extensively in the field of music. Both supervised and unsupervised learning techniques have been successfully applied for tasks such as genre classification, recommender systems, and clustering. However using music as input data is trickier than using other sets of data such as numeric vectors of expenses, for example. The first step in implementing Machine Learning systems for music is to treat our input data, music, in a way that the machine can understand.

This report offers an overview of the ideas and techniques that are commonly used in the field of Music Information Retrieval, as well as an exploration of musical genre classification using machine learning techniques. The code accompanying this report is intended for readers to follow along and explore this fascinating field by providing ways to import and play with the reader's own mp3 library.

Our end goal is then to take pure wave files and come up with a classifier that would correctly identify its genre. But before we do this we have to pick, understand, and extract relevant features from songs. In order to do that we begin by asking the most basic question: what is sound and how can we translate it into machine-understandable language.

The reader is referred to Apendix A for all necessary information on the python modules and programs required to follow along.

What is sound?

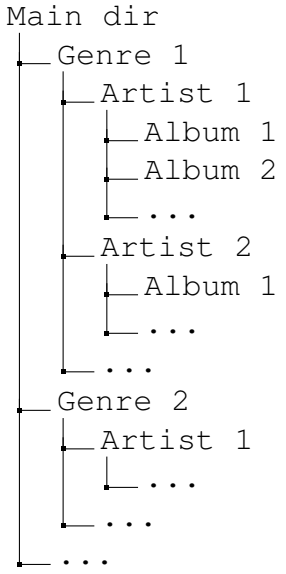
Before we can turn music into something the computer can understand, we need to study what sound is. Sound is just a pressure wave, which means it is a disturbance of the atmospheric pressure that travels to our ears. This pressure wave has a series of characteristics that are important when transforming into machine language. These are: amplitude and frequency. The amplitude tells us how much air is being moved by the pressure wave, and translates to volume. The frequency tells us how quickly the wave oscillates and translates to pitch. A pure tone playing an A would vibrate 440 times per second in a sinusoidal manner, and would consist of a single frequency. A violin playing the same note, however, contains many other frequency components, known as harmonics, which when added up together make the timbre of the instrument. Because these harmonics

interact and interfere with each other, a violin, or any other instrument playing a single note will not produce a purely sinusoidal wave.

A microphone picks up these pressure vibrations and translates them into an electrical current that replicates the shape, amplitudes and frequencies of sound. This current is then used, for example, to drive speakers which oscillate at those given parameters to reproduce sound. Let's take a look at one of this waveforms. But before we do that we might as well take our first aside, and turn to code to convert our music libraries into wav files.

Converting your mp3 and m4a library to wav

We begin by looking at the directory containing the music. It is advised that you store the music you want to convert in a structure similar to this since we use the"genre" folder name to extract the genre as a label for supervised learning.



Once the music has been organized, and after installing the necessary software as it appears on Appendix A, we can proceed to mass convert the selected media using the *mass_convert* function. Let's take a look at the first 20 seconds of three different tunes from three different genres: jazz, flamenco, and bluegrass, respectively:

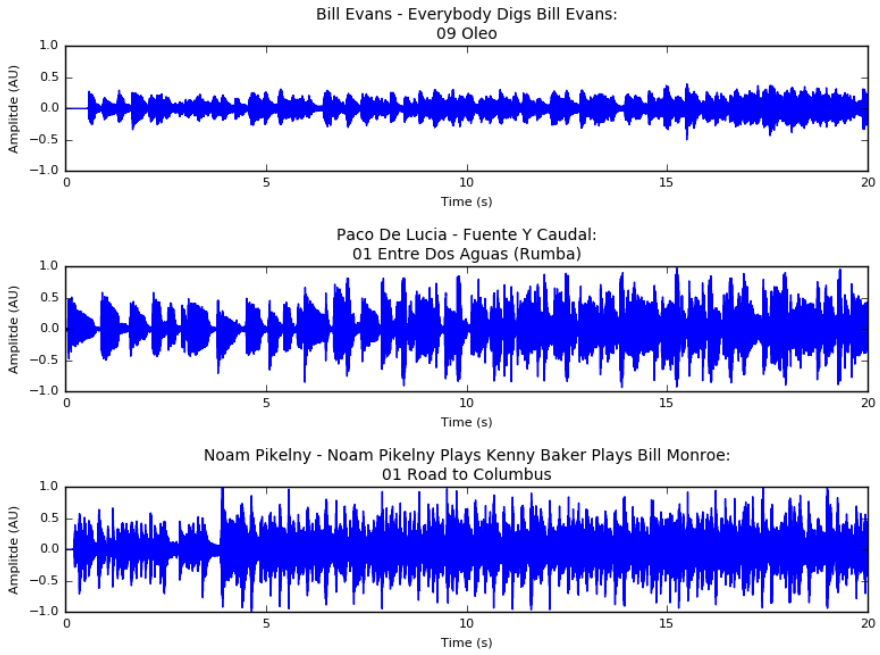


Figure 1: First 20 seconds of three different waveforms

We see that pure waveforms are not readily informative. If shown these three different graphs separately it would be hard to tell which genre corresponds to which. One could naively consider feeding these waveforms directly into our machine learning models but that would be impractical: each of these waveforms is composed of millions of 2-byte integers!

Music information retrieval

Time domain features

Instead we extract relevant and clever features from this waveform, dramatically reducing its dimensionality while still conveying information. One of this features is the ***zero crossing rate*** or ***ZCR***. The ZCR is a simple measure of the rate of change of the numbers of zero-crossings. It can be used to detect onsets, usually of percussive sounds and for monophonic sounds it can be used to estimate pitch, since the zero-crossings are related to frequency. It more generally describes the noisiness of a signal since noisy samples would have a higher zero crossing rate.

In order to visualize this and subsequent features over time, we divide the song into small windows (in the order of tens of milliseconds) and then compute the average over that frame. In order to loose as little information as possible we introduce a 75% overlap between successive frames. Figure 2 shows the zero crossing rate for Paco de Lucia's

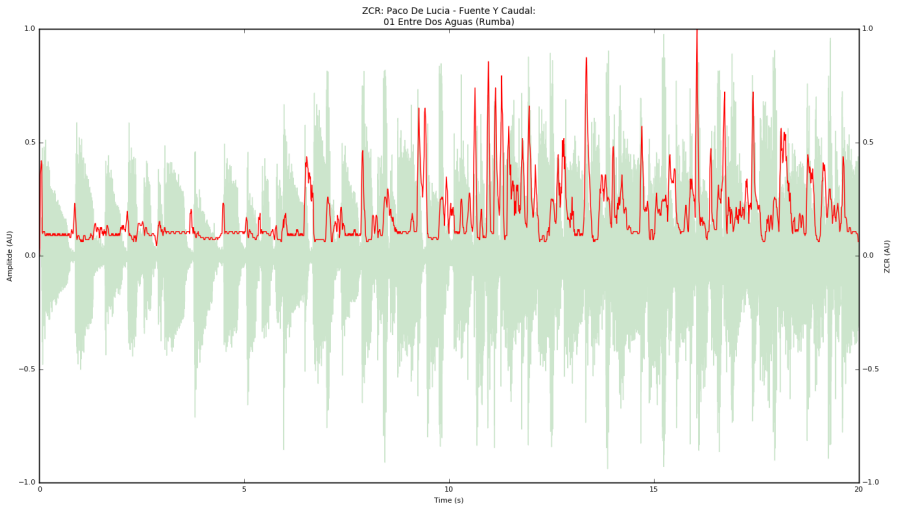


Figure 2: ZCR superimposed on waveform for the first 20 seconds of a song

"Entre dos aguas" song, which begins with a few solo notes on the bass. As we expect we see sharp peaks at the beginning of each bass note and then becomes a little bit harder to interpret.

The next natural question to ask ourselves is whether there are substantial differences between values of this first feature across our genres. To do that we will pick six songs we consider representative of each genre and calculate the aforementioned quantities, as shown in fig 3.

We see that there is enough difference between the values of the *ZCR* among genres for this feature to be useful. It is very interesting to see that the values for flamenco and bluegrass are close to equal. This may be due to a coincidence since at first sight the two genres are very different. However from a wider perspective, flamenco and bluegrass are

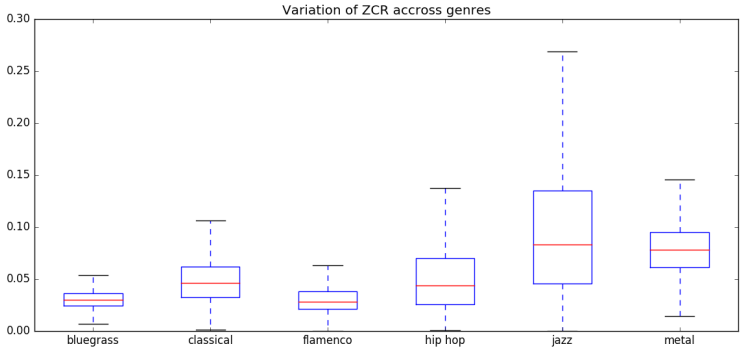


Figure 3: Variation of zero crossing rate across genres

Figure 4: RMS superimposed on waveform for the first 20 seconds of a song

both folklorical genres based upon similar harmonic structures, and with similar harmonic content and a not too different instrumentation.

The next feature we look at is the ***Root mean square*** or ***RMS***. The *RMS* is a measure of the wave's energy, and can be roughly translated into the overall loudness of the track. Once again, let's look at the *RMS* for Paco de Lucia's tune, and how its values vary across genres:

From figure 4 we see that the *RMS* actually follows pretty closely our wave, but that is what we would expect since all the *RMS* does is square the amplitudes and then take the square root of the mean. The graph of the variations across genres, as shown in figure 5 fits our intuition perfectly: we would expect metal and hip hop to be more energetic or loud. We also expect classical to exhibit a relatively larger standard deviation. Once again, this feature is fairly similar in both bluegrass and flamenco which is both exciting

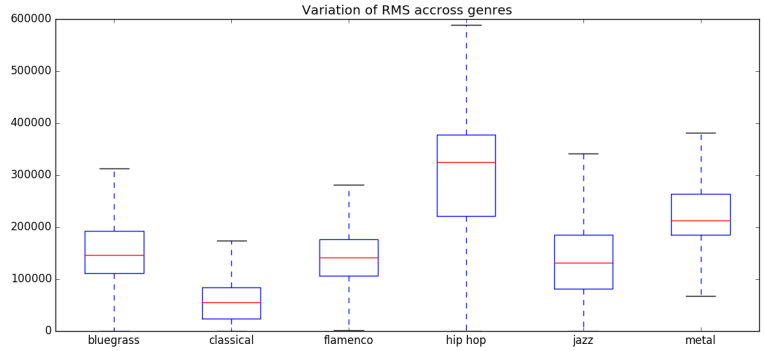


Figure 5: Mean and std of RMS across genres

(the machine is better at understanding genres than most of us) and worrisome (it may have a hard time differenciating between the two).

Frequency domain

For now we have been concentrating exclusively on features extracted from the time domain, that is, the pure waveform, therefore focusing mostly on amplitude. We would now like to focus our attention on frequency but it is quite hard to study the frequencies from just the waveform. To do that we will make use of the Fourier Transform to go into the frequency domain. The Fourier Transform allows us to jump back and forth between the time and frequency domains. The time domain is simply our waveform, in which we see the evolution of our sound over time, at the expense of specific frequencies information. In the frequency domain we can see the different frequencies, as well as their relative weight, at the expense of time evolution. If we played a pure tone at A4 for a second and then a pure tone an octave higher, A5 for another second the frequency domain would show two sharp peaks, one at 440Hz (corresponding to the A4) and one at 880Hz (corresponding to the A5), but there would be no way of knowing which frequency came first or how long each tone was played for.

To overcome this complete absence of time information in the frequency domain we make use of Short Time Fourier Transform, or STFT. Instead of taking the Fourier Transform of the entire song, the STFT divides the tunes into short windows (in the order of milliseconds) and then takes the Fourier transform of each thus gaining some rich time information. We can then view the result of the STFT in what is called a spectrogram, which conveys the presence, relative strenghts, and evolution of frequencies over time.

Let's pause for a second and study these spectrograms one by one. Percussion such as cymbals and snare drums contain a lot of frequencies. In the first tune, Oleo by the Bill Evans trio, the drums come in around second 7 by marking the beat with the cymbals. We can clearly see around ten hits of the hi-hat before the full drums join in around after 15 seconds. The other two tunes both start with a solo instruments. In "Entre dos Aguas" the bass plays the first 5 seconds or so, and we can see that in the spectrogram only low frequencies are preeminent at the beginning. After that the Cajon (a percussion instrument) joins in and we see a lot more frequencies present. The bluegrass song, "Road to Columbus", starts with the banjo. The banjo has a plunky, almost percussive like sound and we can see the difference in timbre and frequencies between the solo bass and solo banjo.

The power (and beauty) of spectrograms is most striking in classical music. Let us show the full waveform and spectrogram of the prelude to Act III of Richard Wagner's "Meistersinger von Nurnberg". Without even having to listen to the tune we can get an idea of the general colors and loudness from both this graph. Particularly, during the quiet fourth minute we can see the change in color from the deep rich horns to the strings.

Now that we are in the spectral (or frequency) domain, there is a whole new set of features we can extract. We begin by looking at the ***Spectral Centroid*** or *SC*. Figure 8 shows the spectral centroid superimposed on the spectrogram for the first 20 seconds of Paco de Lucia's song. The spectral centroid gives us "center of mass" of the spectrum at a specific instant. It translates into the darkness or brightness of sound, with dark having a lower center of mass and bright having a higher one. The first few bass notes of

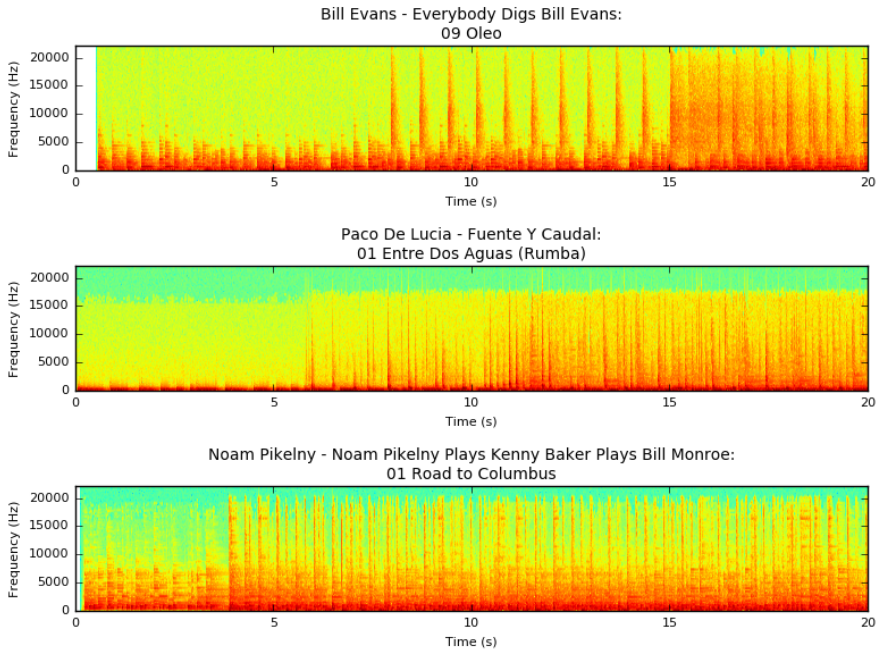


Figure 6: First 20 seconds of three different spectrograms

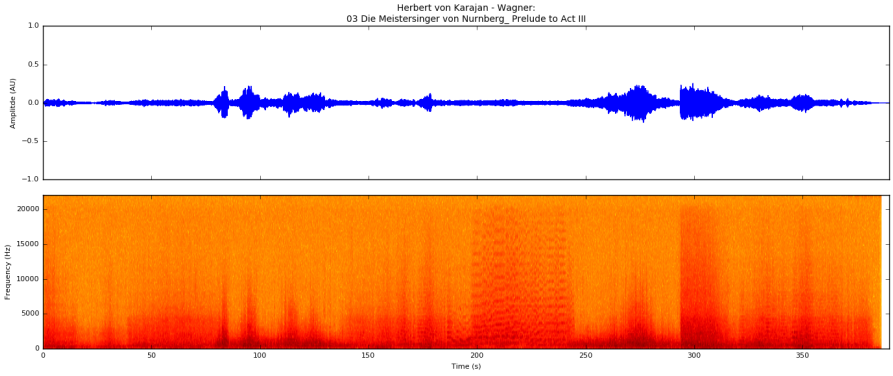


Figure 7: Full waveform and spectrogram of a classical music piece

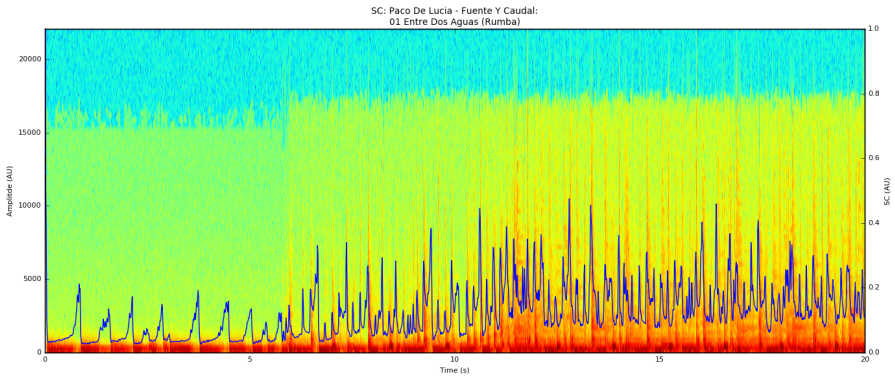


Figure 8: SC superimposed on spectrogram for the first 20 seconds of a song

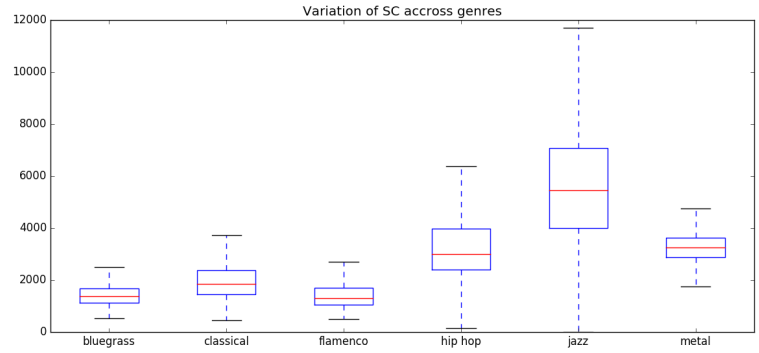


Figure 9: Mean and std of SC across genres

the song have a low spectral centroid. Once all instruments join in, with the guitar and percussion, the spectral centroid takes higher values.

Figure 9 below show the variation of the spectral centroid across genres. As we may expect, jazz (specially Art Blakey, the drummer and band leader chosen as example) is pretty heavy on the cymbals, and the general instrumentation makes the mean value of the spectral centroid pretty high. Once again bluegrass and flamenco have similar values, this time joined by classical music since all three genres have a fuller, darker sound.

Psychoacoustic features

We now extract a series of what are called psychoacoustic features. The human auditory system does not exhibit the same sensitivity across all frequencies and thus does not treat all frequencies equally. The Mel scale is a way to properly account for this. The scale is based on human pitch perception. At low frequencies (typically 500Hz) or less, the Mel scale follows linearly the frequency scale. That means that a pitch at double the frequency is perceived by humans as being an octave higher. However as we go past 1000Hz we need

increasingly large frequency gaps for humans to perceive a pitch difference. We can say we are more sensitive to low frequencies and less sensitive to higher frequencies.

It is common practice then to split our frequency range into bands, known as Mel bands which are fairly narrow for low frequencies but become larger and larger as we increase the frequency. With this bands we can extract what are known as the ***Mel Frequency Cepstrum Coefficients*** or ***MFCC***. The specific procedure to obtain these coefficients is beyond the scope of this report as it involves switching back and forth between time and frequency domains using "regular" Fourier as well as linear cosine transforms. But broadly speaking we start with the spectrogram, as seen for example in fig 6. Then we rescale the y or frequency axis to follow the Mel scale, and split it into the Mel bands. Taking the log of the resulting spectrum and taking what would be another Fourier transform we get a series of coefficients (typically 13, although the first or zero coefficient contains no relevant information) that show us the evolution of the content of the different Mel bands. We can also study the first order Δ, which represent the frame to frame variation of the coefficients. Figure 10 below show the first 13 MFCCs coefficients along with the first order and second order deltas for Paco de Lucia.

The second order delta is shown for illustrative purposes only, its information content being too low to use it as a relevant feature. We can now take the mean across each of the twelve (we discard the first coefficient) Mel Cepstrum coefficients for both the regular and first order transformation.

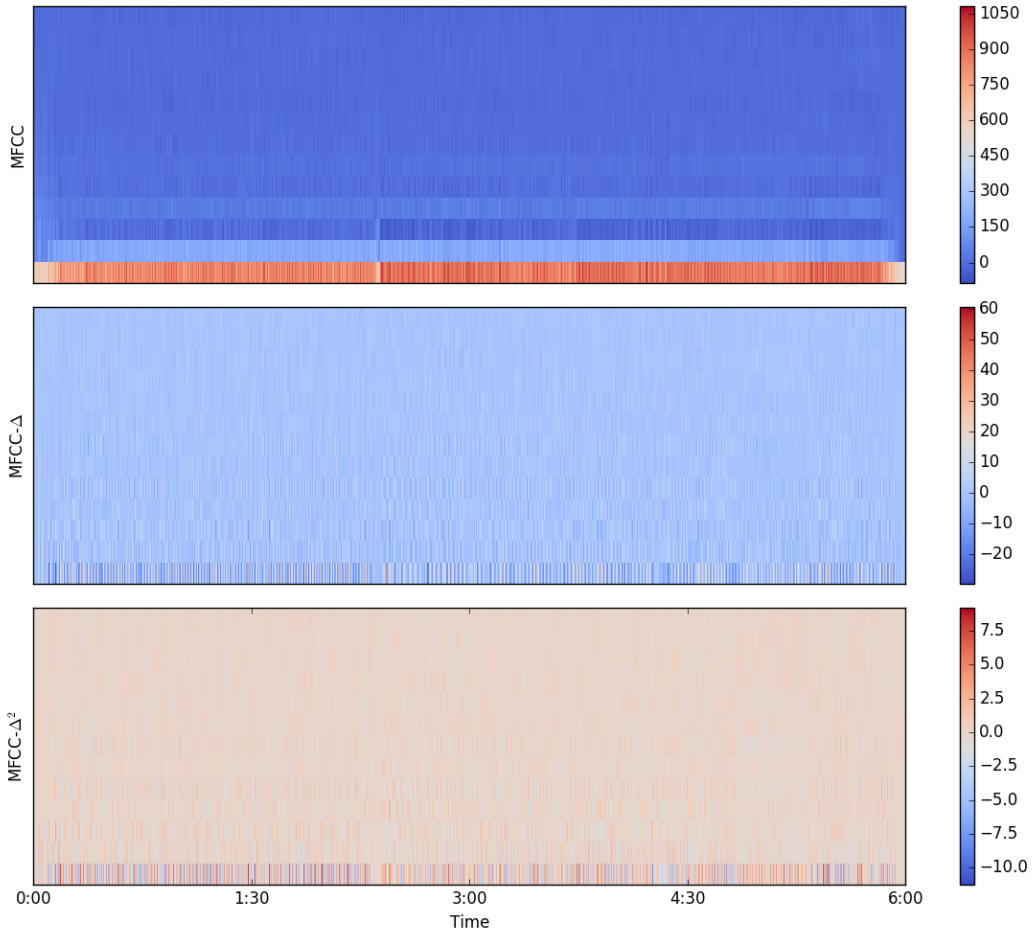


Figure 10: MFCC, along with first and second order delta

Genre Classification

Let us recapitulate and review all the features we have extracted. From the time domain we have the *Zero-crossing Rate*, describing the noisiness, and *Root Mean Square* for the loudness. From the frequency domain we have extracted the *Spectral Centroid*, describing the warmth or brightness of our songs. Finally from the spectral domain, but filtered through the Mel frequency banks we have the *Mel Frequency Cepstral Coefficients*, describing the frequency content adjusted for human perception, as well as its first order variations or delta.

We have successfully translated sound into something the computer can understand and work with. From tens of millions of integers we have gone to a set of 59 features (we calculate the mean and standard deviation for each feature and included the genre). We are now equipped to begin our exploration of Machine Learning techniques for music.

Preparing the data

We will start by extracting the features for all songs in our library and saving them in a dataframe and onto memory so that we do not have to extract the features every time we want to try something. For the time being we restrict ourselves for the sake of time to at most the first 5 minutes of each song. For different musical genres the reader may want to use a shorter window and speed things up but because we are using classical music as a genre we need to include as much time as convenience allows to not focus only on the relatively long introductions.

The next thing we want to do is get rid of the first MFCC and MFCC Δ coefficients which convey no useful information. We will also extract the genre labels as a separate dataset, and code them as numbers, with digits 0 to 5 representing genres: bluegrass, classical, flamenco, hip hop, jazz, and metal respectively.

Building the model

In this section we will explore two simple yet powerful algorithms: *Gaussian Naive Bayes*, and *K-Nearest Neighbors*, as well as ways to improve the latter. We will look into the pros and cons of both algorithms, as well as the assumptions they made and why we believe they are a good fit for the task at hand. We conclude the section with a non-exhaustive, short introduction to other possible methods for music genre classification.

We first must split our data into a training and testing set. The training test will be used to train our algorithm and we will use the testing set to test our results. It is common practice in machine learning to perform such a split since the ultimate goal is to be able to make predictions on unseen data. If we used all of the data for training we have a high risk of *overfitting*, that is making our model follow the training data too closely so that predictions on it are almost perfect but the results do not generalize well. Due to the limited amount of data we have (50 songs per genre for a total of 300 songs), we believe a train/test split of 76%/24% is appropriate resulting in 228 songs in the training set and 72 in the testing set. Because of the relatively small size of the test set, it is important that each genre is represented more or less equally in the testing set. We therefore choose to perform a stratified shuffle split, which keeps the class ratios in the testing set.

We should keep in mind that because we have assigned the genres to our songs beforehand by following the filesystem structure proposed this task would fall under supervised learning. Note however that our initial assignment is somewhat arbitrary. That is there are certain flamenco tracks that sound like jazz, but since they belong to a mostly flamenco album they have classified as flamenco. Similarly some jazz may sound like classical and some hip hop like jazz. Because the genres are not deterministic, perfect scores are virtually unachievable.

Gaussian Naive Bayes

One of the simplest but most robust and powerful family of algorithms for classification is *Naive Bayes*. Naive Bayes methods rely on **conditional independence** between every pair of features, that is the value of a particular feature does not have any effect (either positive or negative) in the value of all other features. Translated to our problem and features this would mean, for example, that the values of the *ZCR*, *RMS*, and *SC* are completely uncorrelated. Of course we know this condition does not apply to our set of features since intuition tells us that a higher value of the zero-crossing rate means we are dealing with a "noisy" signal which probably means it is also loud (high root-mean-square) and with a richer frequency spectrum (high spectral centroid). We can confirm our intuition by plotting the correlation matrix, shown on fig. 11, which shows the correlation between each pair of features.

We see that our first three features (*ZCR*, *RMS*, and *SC*) are somewhat correlated between them, as well as with the first few *MFCCs*, so it seems that the condition of conditional independence is not met by our features. However, as massive an assumption as it may seem, we will see that in practice it often yields surprisingly good results. This is in part due to the ability of Naive Bayes models to handle very large numbers of features, even for limited training points, as it is our case. It is this last attribute of Naive Gaussian models that makes them appropriate for this particular task and dataset. In addition, this family of models require very little data preprocessing, an issue we will get to later when exploring other families of algorithms.

We present our results in the form of a confusion matrix, shown in figure 12. Our confusion matrix consists of six rows and six columns, each representing a genre. The rows represent the true genres and the columns represent the predicted labels. Thus, a song falling into the "Jazz" row, "Classical" column is a Jazz song that was incorrectly classified as Classical.

Before we move to a more quantitative way of measuring performance let's study our results see if they make sense. First of all the results are really not bad for a first approach: we correctly predict all of the twelve classical and hip hop songs, eleven out of the twelve metal and bluegrass, and ten out of twelve classical and flamenco. The mistakes made are not terrible either: we misclassified jazz as classical twice, and we mistook flamenco for bluegrass and hip hop, just to name a few.

We now look at ways of quantitatively measure the performance of our model. Some common used metrics are the *accuracy, precision, recall,* and *F1 score*. Accuracy simply represents the percentage of correctly classified genres. All three other metrics are defined for each genre in our list. The precision measures the number of instances correctly classified as a particular genre divided the number of songs classified (both correctly and incorrectly) as that genre. Looking at the confusion matrix we can calculate the

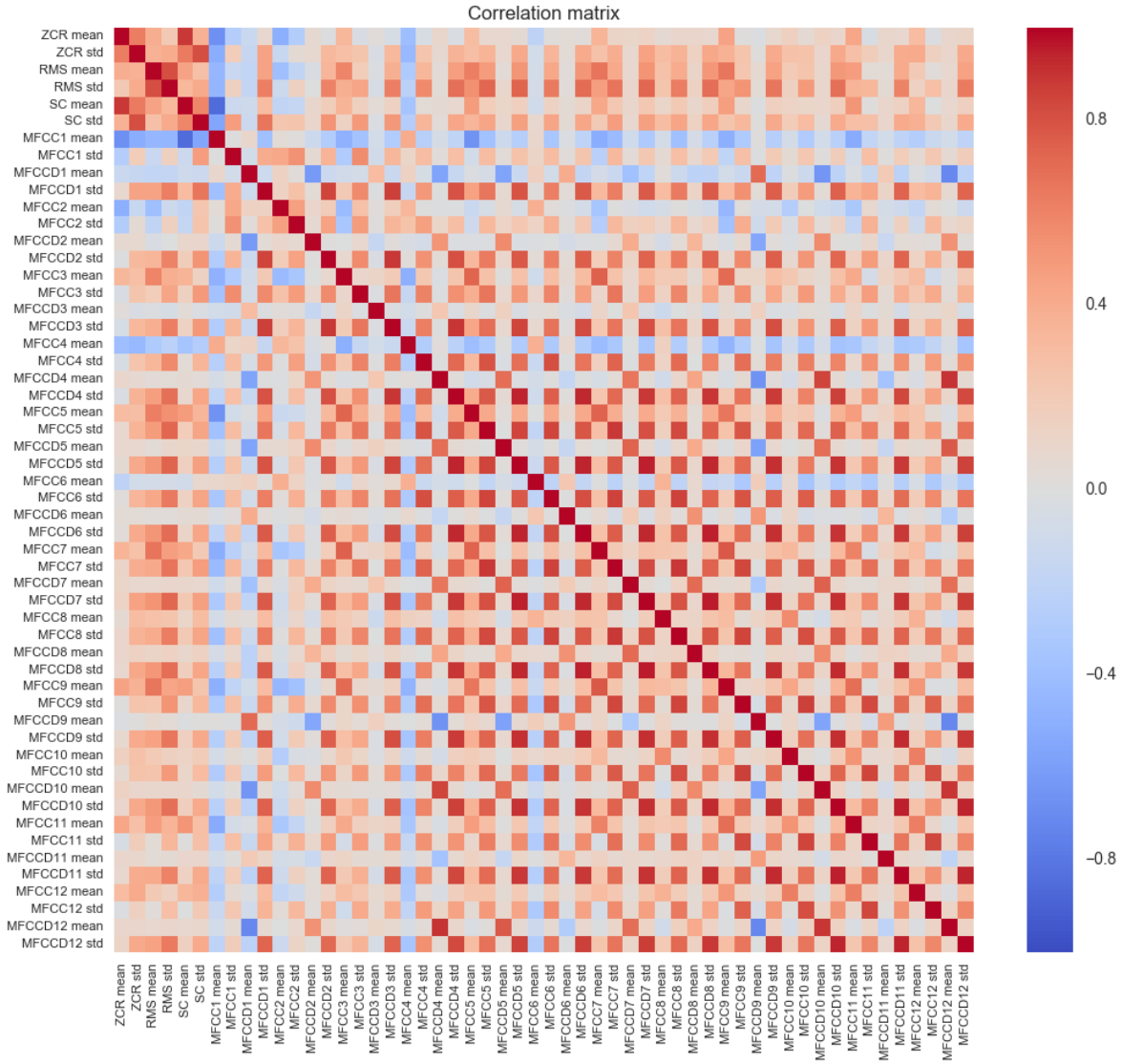


Figure 11: Correlation matrix as a heatmap

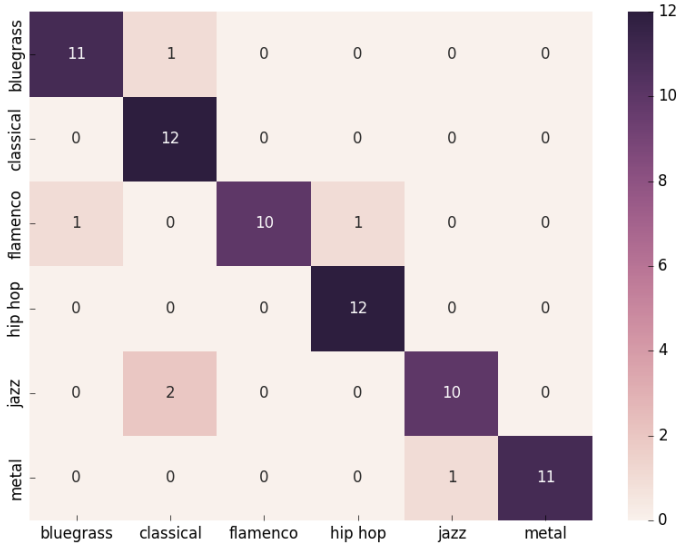


Figure 12: Gaussian Naive Bayes

precision for each genre by dividing the item in the diagonal by the total number of instances in the column. The recall is similarly defined as the number of instances correctly classified but divided by total number of true instances of that particular genre. So, again, looking at our confusion matrix, we can compute the recall for a genre as the element in the diagonal divided by the total number of songs in the row. Finally the F1 score summarizes both precision and recall in a number from 0 (meaning bad) and 1 according to $F1 = 2 * (precision * recall)/(precision + recall)$. The precision, recall, and F1 score of these metrics applied to our Gaussian Naive Bayes model for a particular train/test split are summarized in table 1. The accuracy of our model is 91.7%.

But this is just for a particular train/test split. How do we know we did not get lucky or unlucky? In order to extract meaningful scores for our model we will use cross validation to train and evaluate our model performance on seven different training and testing sets or folds. This is specially important in our case because we have limited data. The results are summarized in table 2. We see a slight drop in our average scores for all three scoring functions which means we had gotten "lucky" the first time. From now on, every table of results will be presented after 7-fold cross validation.

What can we compare this numbers against? Let's look at the simplest of baselines which consists of a model which picks a genre at random. What values of precision and recall can we expect from such a model? Since all genres have the same probability of being chosen our each cell on our confusion matrix would (on average) be equally populated. Since we have 72 test set samples and a 6x6 matrix, that means (again, on average) a "2" in each cell leading to a precision, recall and f1 score of 0.166 which incidentally is the probability if selecting a particular genre. Now we have a way of quantitatively measure

	precision	recall	f1 score
bluegrass	0.92	0.92	0.92
classical	0.80	1.00	0.89
flamenco	1.00	0.83	0.91
hip hop	0.92	1.00	0.96
jazz	0.91	0.83	0.87
classical	1.00	0.92	0.96
avg	0.925	0.917	0.917

Table 1: Scores of Gaussian Naive Bayes

	precision	recall	f1 score
bluegrass	0.85	0.8	0.82
classical	0.86	0.95	0.9
flamenco	0.89	0.86	0.87
hip hop	0.99	1.00	0.99
jazz	0.93	0.9	0.92
classical	0.98	0.93	0.95
avg	0.916	0.907	0.907

Table 2: Scores of Gaussian Naive Bayes after cross-validation with 7 folds

	bluegrass	classical	flamenco	hip hop	jazz	metal
blues	0.002	0.0	0.039	0.0	0.959	0.0
bossa nova	0.001	0.0	0.091	0.0	0.909	0.0
folk	0.599	0.0	0.401	0.0	0.0 0.00	

Table 3: Prediction probabilities for new genres

the performance of our model. In light of these new metric we confirm that our simple model already dramatically outperforms the baseline .

One other thing we can do to see how well our model performs is to test it on some genres that are outside of our original list and see how well it classifies them. In order to do that we can make use of one of the very attractive properties of *Gaussian Naive Bayes*: soft assignment. This means that for a given tune we can calculate the probabilities that it belongs to one of our six predefined genres. We can see how it performs on a blues, a bossa nova, and a folk song on table 3. It is remarkable that our model gives both blues and bossa a very high probability of being jazz, since this is certainly the semantically closer genre. It also classifies folk as an almost even mixture of flamenco and bluegrass, which again seems like a very appropriate mix!

One of the advantages of using a Gaussian Naive Bayes method for music classification is that is works right "out of the box". That means that the model has no parameters we can tweak and fine tune. Its simplicity with its remarkable results (see table 1) make it a very attractive model. However this lack of parameters means that with this model there is no room for improvement. We then move to a different family of algorithms, one in which we can modify the parameters to improve the model and learn more about our music library at the same time.

K-Nearest Neighbors

We now turn to another one of the most conceptually basic but powerful machine learning algorithms for classification: *K Nearest Neighbors* or *KNN*. *KNN* is a special kind of algorithm in that it does not require any training. It simply stores the data and label in memory and once asked to classify a particular instance, it computes the distance (with a specific distance function) to all points in the training data and assigns a label based on the label of the K nearest neighbors.

But before we can apply *K-NN* to our data we have to do some data preprocessing. As we have seen, *K-NN* works by calculating the distance between a point and its neighbors. But we are using different scales to measure each of our features! *ZCR* has very small units (between 0 and 1), whereas *RMS* is often measured in the tens of even hundreds of thousands, and *SC* is a measure of frequency, so it must lie in the range 0-22,000, just to name a few. We have to properly scale our features so that any notion of distance is actually meaningful.

One approach to achieve this unit normalization is **Standarization**. By subtracting the mean and dividing the standard deviation we effectively transform our features into quasi-Gaussian distributions with 0 mean and unit standard deviation. Once we have standardized our data we can proceed to using the model to predict musical genres.

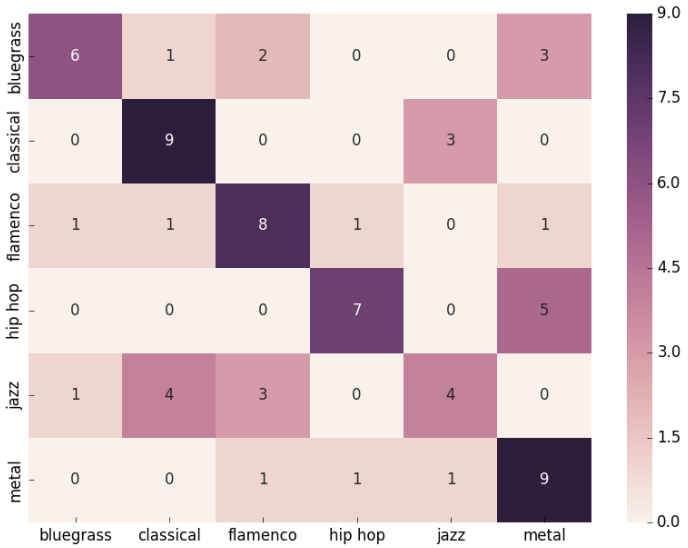


Figure 13: Confusion matrix for $K = 1$ and Euclidean distance

As a first approach to music genre classification with *K-Nearest Neighbors* we will apply the "simplest" form of *KNN*, that is without fine-tuning any parameters and using the raw data as our input. We will also look at the $K = 1$ case, that is, we only look at the closest point in feature space. Our distance function will be the well known Euclidean distance.

We can clearly see this model outperforms our previous *Naive Bayes* model, achieving an outstanding *F-1* score of 0.955 and an accuracy of 95.4%. Can we bring the score higher? It seems like a difficult task given the results quoted in this section but at least it can serve as confirmation that we are effectively doing our best. In the next subsection we will explore the parameter space of *K-NN*.

Improving the model

How can we improve our model? The KNN algorithm has a series of parameters we can play with and tune to improve accuracy. The first and most obvious parameter we have available is the number of neighbors K used to assign a label. In the previous example we limited ourselves to the most basic case of $K = 1$, but we could look at any number of neighbors, varying K from one to the number of points in our training set. Another parameter we can tune when considering more than one neighbor is the weighing of the points. When considering more than one point we can treat them equally, regardless of their distance to the sample we want to classify. Or we could give each neighbor a weight depending on how far it lies on the feature space. And finally we have our distance function. We have a series of distance functions available to us: the well known Euclidean distance, the Manhattan distance (in which the distance is the number of steps taken

	precision	recall	f1 score
bluegrass	0.96	0.99	0.97
classical	0.99	0.99	0.99
flamenco	0.86	0.96	0.91
hip hop	0.99	0.92	0.95
jazz	0.96	0.93	0.95
classical	1.0	0.94	0.97
avg	0.960	0.954	0.955

Table 4: Scores of KNN with $K = 1$ and Euclidean distance

in the direction of our vector space unit vectors), or the Chebyshev distance (in which the distance is the maximum distance along any one of the vector space directions), for example. We will explore varying the number of neighbors between 1 and 25, changing the weights, and using both Manhattan and Euclidean distance.

We perform our exploration of parameter space using grid search. This type of strategy takes a series of possible values for the parameters and a scoring function (in our case the F1 score) and then fits and evaluates a model with each possible combination of parameters, returning the ones that obtain the highest scores. Now, the way we tune this parameter is by looking at the results on the test set. But we may now be at risk of overfitting the testing set with our model parameters! One way out of this is to use cross validation: we first split our data into k folds. We then train our model on *k-1* folds and evaluate on the remaining fold, and repeat this procedure for each of different fold groups. In order to study the stability of our model we perform a nested cross-validation, where before estimating the best model parameters, we hold out a portion of the data for model evaluation. So first we split our data into a train and a test set, then using cross-validation on the training set we estimate the best model parameters and a model with such parameters is evaluated on the test set. Table 5 summarizes our findings. The outer cross-validation (used for model evaluations) gives us five different models with different sets of parameters. We see that 4 out of the 5 models suggest using weights based on distance, and all models use Euclidean distance. In terms of the number of neighbors used the results vary between 1 and 5. We settle on the model with the average parameters, that is 3 nearest neighbors, distance-based weights and Euclidean distance. This model offers a slight improvement over our unoptimized original model. It uses three nearest neighbors, which reduces the possibility of overfitting. It also assigns weights based on distance, which means that out of the three neighbors, those "farther" away contribute less to the final label assigned. This variation on the number of nearest neighbors for different folds, however, suggests that further tests of the stability of our model are required.

In order to test the stability of our model, we will once again use cross-validation. We split our data into train and test sets, train the model on the train set and evaluate on the test set. We repeat this procedure for different train/test sets and look at the variation on f1 score. If the scores don't vary too much then we have arrived at a stable

	n neighbors	weights	distance	f1 score
model 1	4	distance	euclidean	0.97
model 2	5	distance	euclidean	0.99
model 3	3	distance	euclidean	0.99
model 4	3	distance	euclidean	0.97
model 5	1	uniform	euclidean	0.96
avg	3	distance	euclidean	0.976

Table 5: Parameters and f1 scores for different folds

model. However if the f1 scores vary widely then our model is unstable. Again, we use our final model with $K = 3$, distance-based weights and Euclidean distance, and split our data into 7 different train/test groups. The mean f1 score obtained is 0.960, and the standard deviation of the f1 scores is 0.024, with a minimum of 0.930 and a maximum of 0.986. This is a relatively high standard deviation, indicating that we obtain different f1 scores depending on which test subset we evaluate the model on. However, all scores stay above 0.93 which means we are still facing a useful and accurate model. The main cause of the variation on the accuracy of the model depending on the test set is due to the limited data available: only 50 songs, coming from a limited number of artists are present in each genre. If the training set contains a good proportion of songs of each artist, then it will have a relatively high accuracy when evaluating on the test set, which includes tracks from the same album the model was trained on. If, however, songs of a particular artist end up being unrepresented in the training set, then the accuracy will drop when evaluating the model. This suggests (as is often the case) that we need more data to achieve a more generalizable model. A wider variety of artists and albums will surely drop the overall accuracy but will also reduce the variation in scores across folds. But again, despite the somewhat large variation in accuracies, these stay comfortably in the range 0.930-0.986, which means the model is still useful for genre classification.

We have now achieved a fully functional and accurate model. It is able to differentiate correctly between the genres and with precision and recall scores much higher than the baseline. By exploring the parameter space we have confirmed the best choices of parameters for the task at hand. Before we conclude this section and present a simple way to visualize your music library it is worth exploring other models.

Benchmarking and Exploring other models

One of the problems we encounter when trying to compare our model to the literature is that for the task of Music Information Retrieval and Genre Classification there is not a well studied and standardized dataset. Some papers rely on the "Million Song Dataset" (consisting of a million western songs) but this dataset is already populated with a specific set of features and does not allow for the download of the music itself to extract other features. It is pointless to compare our results from those derived from the "Million Song

Database" since the features studied vary considerably (the "MSD" contains features such as tempo, fade-in and out timings, etc).

We are thus left to the task of gathering, and more importantly labeling the data ourselves. For this report we labeled albums as a whole belonging to a genre. For example, all the songs in Bill Evans' "Everybody digs Bill Evans" were labeled as Jazz, including "Peace Piece", a solo piano tune that will have been most likely labeled as Classical when taken separately.

Yet another element making bench-marking difficult is the choice and number of genres. The number of genres chosen and their musical proximity can have a huge impact on performance. It will be much more difficult to classify 10 genres than 3 and it is easier to classify between, say, jazz and metal than between pop and rock.

Despite all these differences we can turn to the existing literature to get a general idea of the effectiveness of our model, as well as to learn about other approaches to music genre classification other than KNN and Naive Bayes. All papers and projects cited on this paragraph appear in the references section. Cast et al. used only the MFCC features and achieved outstanding 98% and 100% accuracy using Naive Bayes and KNN respectively, although they limited their data to three distinct genres. Haggblade et al. also used MFFC features but used a wider range of methods on four different genres. They achieved an accuracy of 80% using KNN but improved that figure significantly with other methods such as Multiclass SVM (87% accuracy) and Neural Networks (96%). Yaslan and Cataltepe offer an thorough exploration of the topic by reporting accuracies for different models and on different subsets of data. Their accuracy for Naive Bayes peaks at around 60% and for KNN at 80%. Ezzaidi and Rouat used Gaussian Mixture Model for an outstanding 99% recognition accuracy using, again, the MFCCs as features. Another example using neural networks and a growing neural gas from Clark et al. report an accuracy of 68%.

We see that the results vary considerably between the different work cited. We have to look at those number in context, and look at how many genres were they classifying, and how much data they had available. An 80% accuracy using KNN for 4 different genres may seem under par but if the genres are very close together or if they manage to predict using only one second of audio then the results are remarkable.

Overall, both our models achieve relatively high accuracies taking into account the limited number of features available (many of the existing literature use data such as year released, tempo, and key), and the number of genres classified although more reliable results could be obtained by using more data or at least by selecting a richer variety of artists in our dataset.

Visualizing your music library

We conclude this report with a fun way of visualizing your own music library in a two dimensional plane. We recall we have been working in 58 dimensions throughout this report, and the task of visualizing such a space in two dimensions may seem daunting at first. But there is a very powerful dimensionality reduction technique that can accomplish this task, while retaining most of the information.

Principal Component Analysis or ***PCA*** looks at our data in however many dimensions it lives and changes the coordinate system to follow the directions where data exhibits the most variation. Let's think about what this means with an example: suppose you start off with two dimensional data that lies in an ellipse, but that is not centered or aligned with the axis. The directions of highest variance of the data are of course the major and minor axis of the ellipse. *PCA* moves our original coordinate axis so that our new "x" is aligned with the major axis and our new "y" is along the minor. FIgure 14 illustrates this procedure.

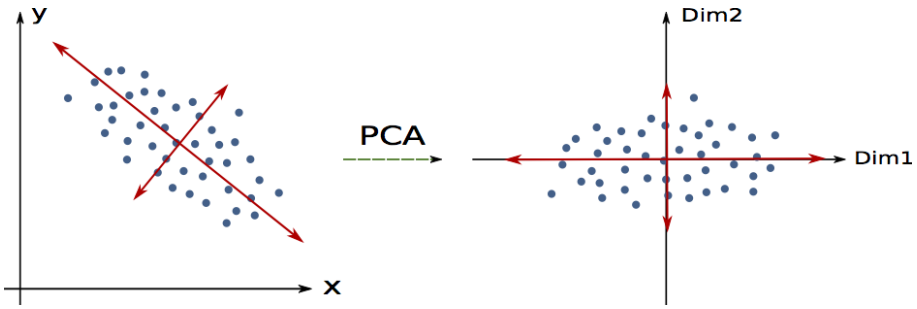


Figure 14: Example of Principal Component Analysis

Before performing PCA analysis on our data, and in order to facilitate interpretation of the PCA components, we manually reduce the dimensions of our data by only considering the *MFCC* means and standard deviations. Since this features relate more closely to our human perception of the sound, while still conveying useful information we consider this set of 26 features to be sufficient to describe and differentiate our genres. We are now in a position to perform PCA on our reduced, 26-dimensional data. Figure 15 shows the particular linear combination of original dimensions making up our PCA components, as well as the variation in the data explained by each. We see that together the two principal components selected explain 75% of our data.

We can interpret our new dimensions in the light of the relative weight of each *MFCC* in our new dimensions. We recall that the *MFCC* describe the presence and evolution of frequencies in particular bands, these bands being related to our perception of sound. We also recall that the human auditory system has a better sensitivity at relatively low frequencies. We see that our first component reflects songs which carry more weight in the higher frequency bands, at the expense of the first two bands. The same goes for their standard deviations: our first dimension looks for higher variation in the higher frequencies than in our first two bands. Our second component can most easily be described as songs with a very strong presence in the second Mel band, right around where our sensitivity peaks, at the expense of both our first band and higher frequency bands. We thus expect darker and heavier songs to lie towards the bottom left corner while warmer and more percussive (remember percussion has a very rich harmonic spectrum) to lean towards the upper right. Figure 16 shows our music library plotted along our new axis, colored based on genre.

We do not see clusters as delimited as we would expect from the results of our genre classification but that is natural, after all we have effectively reduced the dimension by a factor of more than 20 and our PCA components only explain 75% of the variance. We expect our data to form more distinct clusters in higher dimensional space. However we

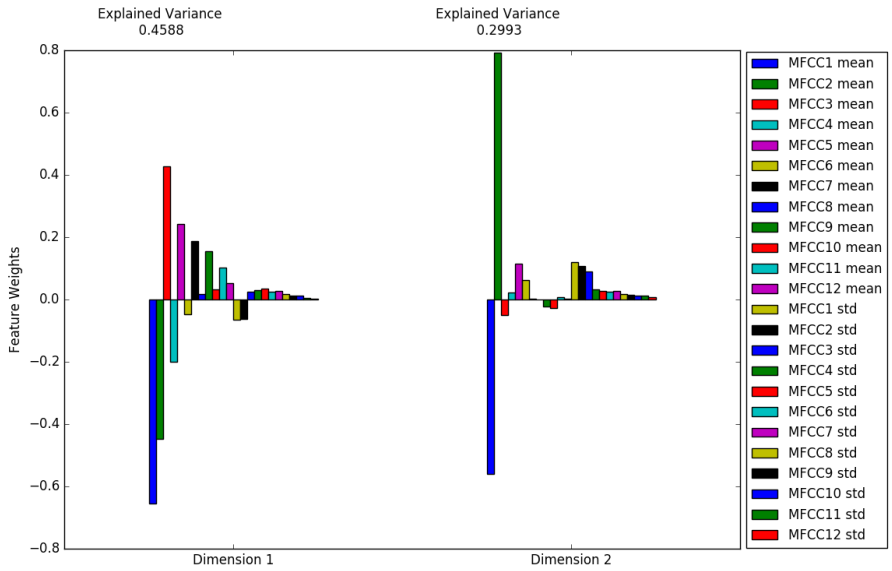


Figure 15: Principal Component Analysis on our music data

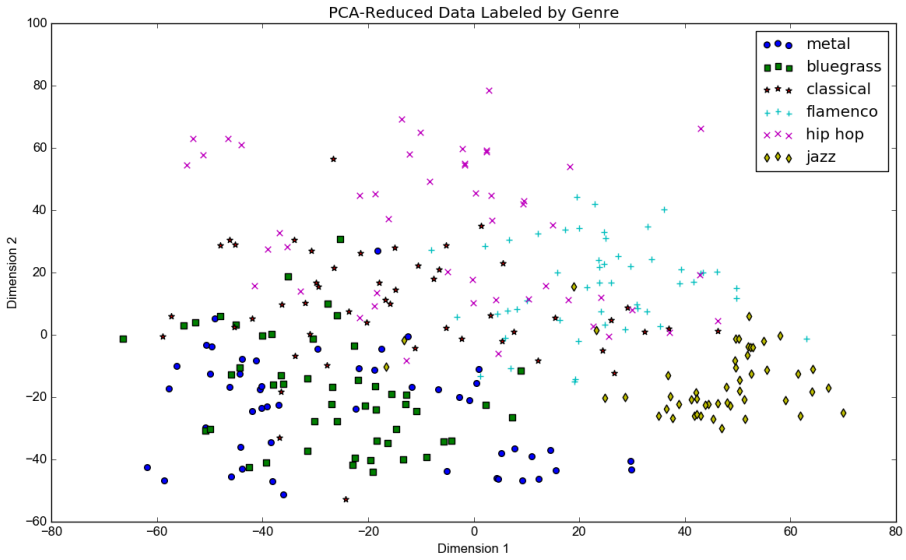


Figure 16: Musical library plotted along PCA components

can already confirm visually some of the results obtained during classification. We see, for example that Jazz is much more spread out than other genres with most of the songs clustered to the lower right but some other living in the classical and flamenco spaces. We also see that both classical and metal, our "darker" genres lie in the bottom left, as we would expect.

Of course, this graph on its own has little information content, but paired with our genre classification algorithms, it offers a simple and fun way to visualize our music library and to get an rough idea of which genres share more characteristics than others, and which genres lie in opposite sides of the spectrum.

Conclusion

This short report has offered a glimpse into the complex and fascinating world of music information retrieval, whose task is to extract meaningful features from both the time and frequency domain, as well as psychoacoustic features with the goal of bridging the semantic gap between our understanding of music and the computer's. Features such as *zero-crossing rate, root mean square, spectral centroid*, and *Mel Frequency Cepstral Coefficients* have been explained and extracted.

Armed with this new set of features, machine learning techniques have been applied for the task of genre classification. Simple but powerful models such as *K-nearest neighbors* and *Gaussian Naive Bayes* as well of some common metrics to judge the quality of the output of such models have been examined and applied, fine tuning our model parameters using cross-validation.

Finally, with the purpose of offering some insights on dimensionality reduction techniques, we have plotted our music library along the dimensions of most variation. This graph offers a simple but attractive way of comparing genres and offers some explanation for the results of our classification models.

Personal evaluation: This project was really, really fun. One of the issues that had troubled me for a while was that a lot of the cool things you can do with machine learning rely on having really good data, so it was really cool to jump that hurdle and turn my own music library into useful data for machine learning. It also gave me the excuse to go back and relearn a lot of concepts related to music such as Mel Cepstral Coefficients. I was really surprised to get such relatively good results from these simple models, but they are two of my favorite models and I was happy to see how powerful their simplicity is once again.

Things to improve: As we've seen on the section on model parameters and cross-validation, our model is still dependent on the particular train/test split due to the very limited amount of data available. Not just in the number of songs per genre, but more specifically in the number of artists and albums represented. The same number of observations but with each song being drawn from a different artists would certainly be able to generalize more than the current model.

Future work: I would really like to explore the limits of this model in terms or granularity. We have seen it can differentiate fairly well between classical, jazz, flamenco, metal, hip hop, and bluegrass. But how well would they perform on much closely related genres? Could this model be used to distinguish classical music genres such as a symphony from a concerto from a string quartet? One other really cool thing would be to try

to classify performers. Could these simple models tell Wilhelm Kempff and Grigory Sokolov's renditions of Beethoven concerti apart? How about Magdalena Kozena and Nancy Argenta singing Bach? This task seems much more difficult than simple genre classification and it may need more sophisticated models and features, but by slowly making the problem more and more complex perhaps useful insights appear.

References

- Udacitiy's Machine Learning Nanodegree material
- `https://github.com/librosa/librosa`
- `http://scikit-learn.org`
- George Tzanetakis and Georg Essl, "Automatic Musical Genre Classification Of Audio Signals", IEEE Trans on Sp and Audio Proc., 2001
- `http://wwwnew.schindler.eu.com`
- `http://www.ifs.tuwien.ac.at`
- `http://musicinformationretrieval.com/`
- `http://www.christianpeccei.com/musicmap/`
- `http://stackoverflow.com/`
- Luis Pedro Coelho and Willi Richert, "Building Machine Learning Systems with Python 2 ed.", Packt Publishing, 2015
- Cast et al. Music Genre Classification
- Haggblade et al. Music Genre Classification
- Ezzaidi H, Rouat J. Automatic musical genre classification using divergence and average information measures. Research report of the world academy of science, engineering and technology; 2006.
- Clark et al. Music Genre Classification Using Machine Learning Techniques

Appendix A: Installations

The code accompanying this report is built in python 2.7. Below is a list of the different modules and utilities that need to be installed:

- In order to convert our mp3 or m4a collection to wav from the command line we need librosa, along with the mp3 encoder lame.
- Alternatively, if the our library is exclusively composed of mp3 we can use the faster mpg123.
- **Python modules**
  - numpy v1.11.0 for basic algebraic calculations.
  - matplotlib v.1.5.1 for plotting.
  - pandas v.0.18.0 for storing the data in dataframes.
  - librosa v.0.4.3 for music feature extraction.
  - sklearn v.0.17.1 for machine learning.
  - seaborn v0.7.0 for prettier plots.