POLITECNICO
MILANO 1863

MSc. Music and Acoustic Engineering

Computer Music - Languages and Systems

Homework 1
Genre Classification

Authors:
10504151
10813836
10426992
10754748
[Group 16]

Link to the repository:
https://github.com/IronZack95/CMLS_First_Project

1 Music Genre Classification

Musical genres are labels created and used by humans for categorizing and describing the different pieces of music. They have no strict definitions and they are characterised by common features typically related to the instrumentation, the rhythmic structure and the harmonic content of the music. So the musical genre is a high-level descriptor and it is subjective. In fact it is not easy to define exactly all the possible genres that are present in the musical universe and asking to different people to classify some pieces of music not all of them will assign a certain piece to the same musical genre with certainty.

The idea of this project is to write a code which automatically classify different musical genres from a dataset of audio files. To classify these audio files, we will use low-level features of frequency and time domain and we will build a relationship between these low-level features and the high-level features of music.

We start from a dataset of audio tracks having similar size and similar frequency range. We use GTZAN genre classification dataset which is one of the most recommended datasets for the music classification. This collection was built in 2000/2001 and consists of 1000 audio files in .wav format, each lasting 30 seconds, at 22050 Hz sampling rate. There are 10 classes, each containing 100 audio tracks, but for the aim of this project we select only four genres among the 10 available: Blues, Metal, Hip-Hop and Reggae.

To fill the semantic gap between the low-level features and the high-level characteristics of music, we will use *Support Vector Machine* (SVM). We do a first training phase and then a testing phase.

In the training phase, we extract the features from the musical pieces contained into the dataset. Feature extraction is the process of computing a compact numerical representation that can be used to characterise a segment of audio. We choose a particular set of low-level features and then we train a SVM to learn the correspondence between the low-level features and the genre annotation. The features we will use are the *Mel Frequency Cepstrum Coefficients* (MFCCs) that are perceptually motivated features based on the STFT. So we start from a frame of the audio signal and we end up with n coefficients.

After that, we do a testing phase in which we test the trained model on new data to understand how good the performance of the model is. Since we consider four musical genres, we will use a methodology which is called Majority Voting. In practice we have six different SVM, one for each possible couple of the different classes we are considering. The first SVM will learn to distinguish Blues from Hip-Hop, the second one Blues from Metal, the third one Blues from Reggae, the fourth one Hip-Hop from Metal, the fifth one Hip-Hop from Reggae and the last one Metal from Reggae.

Once we have these training models, we analyse the audio tracks. For each audio track, the six different SVM return a certain class as a result. Then we apply the majority voting, so the music piece is classified according to the genre that is returned the most times. The scheme is represented in Figure 1.

At the end of the process, we build a confusion matrix to understand the performance of the model.

2 Features Description

Music genres classification is subjective. The best way to approach this problem is using the Mel Frequency Cepstrum Coefficient (MFCCs) timbral descriptor. This technique can

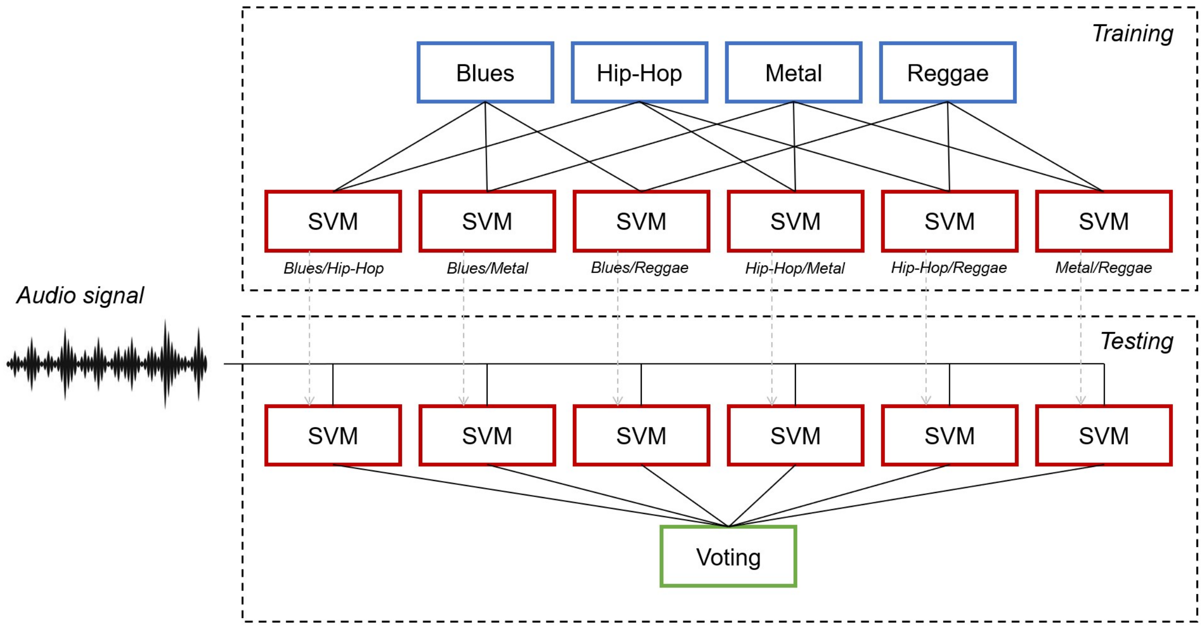


Figure 1: Multiclass music genre classification

be used as an excellent features vector representation of human hearing. Our auditory system is extremely non-linear and Mel Frequency Cepstrum Coefficients allow us a non linear representation of the frequency spectrum. In particular, in order to convert a frequency f in Hz into its equivalent in mel, we use this formula[1]:

$$\boxed{MelFrequency = 1127.0148 \log{(1 + \frac{f}{700})}} \tag{1}$$

The MFCCs computation involves, in geral, 6 steps:

1. **Signal Sampling** in order to obtain a discrete signal[2]

2. **Discrete signal windowing** in order to divide it in frames containing a certain number of samples. Usually one applies a regular spaced window of the same length through the audio signal. The most common used window is the *Hamming Window* which has smoothed borders.

3. **Compute the DFT** passing in the frequency domain in order to obtain the signal spectrum and then take its magnitude.

4. **Apply the Mel Filter Bank to the spectrum**. The Mel Bank at point 4. is used to map the powers of the spectrum obtained above onto the mel scale defined with the formula (1). A Mel Filter Bank is composed by Triangular Overlapping Windows mapped in according to the mel frequency scale, as shown in Figure 2. The result of this filtering process is a Mel signal spectrum divided in mel-scale spaced bands.

[1]Note that Python Librosa libray allow us to compute the frequency scale transformation in a single command without specifying the formula (1)

[2]In our case this step is not necessary because the music tracks are already in the digital domain

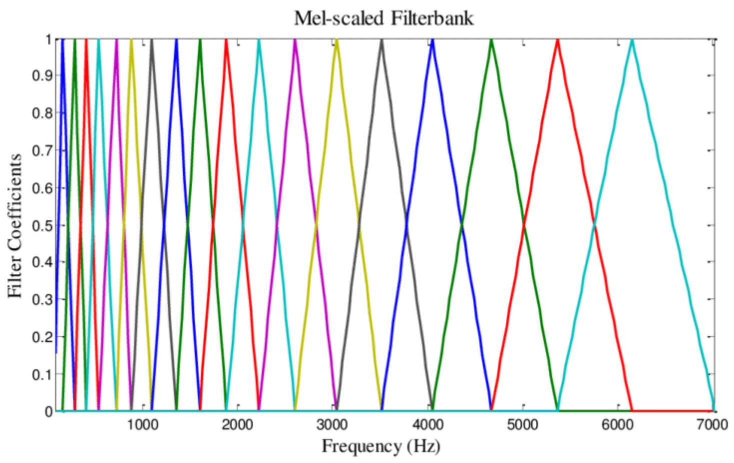


Figure 2: Mel Filter Bank

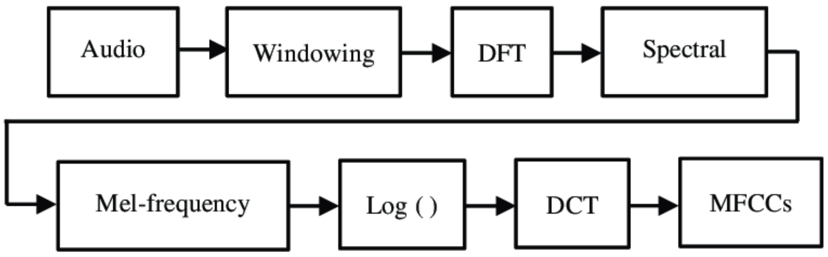


Figure 3: Flow chart of extracting Mel Frequency Cepstral Coefficients MFCCs

5. **Compute the logarithm of each band of the Mel Spectrum**

6. **Discrete Cosine Transform (DCT)** of the list of Mel logarithm powers.

The Mel Frequency Cepstrum vectors are formed by the coefficients of the DCT. In our case we are considering 18 Mel coefficients. This means that the original spectrum of the signal is divided in 18 bands during the Mel filtering step before the logarithm and the DCT computation.

We defined a function that compute the FFT of the input signal and window it with an *Hamming Window*. We filter the spectrum using a Mel Filter function contained in the Librosa library and we find the logarithmic value of the result before applying the Discrete Cosine Transform.

3 SVM Classification

3.1 Computing the Training and Testing Features

Once the feature computation step is over, one has to retrieve those features from the samples, applying the feature computation code to its own "sample basket".
At this point, two steps are needed: the "Training-definition" step and the "testing-definition" step. These are two preliminary steps which prepare the machine for the actual training and testing step which will be dealt later on in this paper.
During this procedure the feature-computation-code is iterated all over the sample basket in order to extract the features (which in this case are just the MFCC) from the samples

and then load it into two separate lists called “dictionaries”, one for training operation and the other one for testing operation.
These two dictionaries, which respectively go under the name of “training dictionary” and “testing dictionary” (also known as “Training set” and “Test set”), are the lists of extracted features which the machine will later analyse to train and test itself.

Since the goal of this work is to create a machine capable of recognizing four different music genre, the sample basket has been filled with songs; in particular, 100 songs for each genre (Blues, Hip-Hop, Metal and Reggae) have been used, 70 of which were filled the Training set, while the remaining 30 were used for the “Test set”.
Moreover, since the "Training-definition" step and the "Test-definition" step are really similar to one another, only the former will be discussed; the only difference between the two is in the name of the variables (test / train).

Let us now dive into the "Training-definition" step passages.
Firstly, and array called “classes” is instantiated and filled with the four genres’ names.
Then, the number of mel’s coefficients is set to 18 and the “training dictionary” is instantiated as an array under the name of `dict_train_features[ ]`.
As it is possible to see in the code, this array gets filled with four other empty arrays (one for each class) each of which, in turn, will be filled with the MFCC. Afterwards, two nested for cycles are created, one to be iterated over the classes, and the other to be iterated over each sample.
For each class (first for cycle) the following operations are done:

- The file path where to take the samples from is set, under the name of train_root.
- An array called `class_train_files[ ]` is instantiated and filled with the samples.
- A variable called `n_train_samples` is defined as the number of samples of each class.
- A matrix filled with zeros `called train_features[ ]` is instantiated with dimensions [`n_train_samples` X `n_mfcc`].

For each sample (second for cycle), instead, the code does the following:

- Loads the “.wav” file through the function `librosa.load`.
- Computes the MFCC of each song via the user-created function discussed in the first section of this paper.
- The `train_features` matrix gets updated with the values of the MFCC for each song.

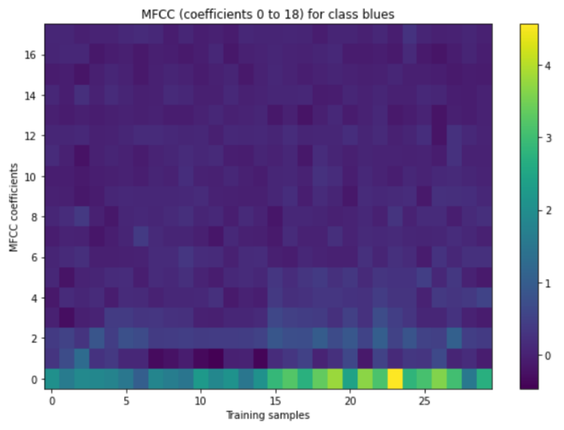
In the last line of code, the `dict_train_features[ ]` array is filled with the features and it is ready to be read by the machine during the actual training step.
Once the training step definition is done, the next operation is to code the Test step definition.
As mentioned above, the Test-definition step will not be covered in this paper since it is really similar to the one we have discussed until now. However, by having a look to the code, one does not find difficult to see that it is the same code of the Training-definition step, except for the fact that the word “train” has been swapped with the word “test” since this is the test step definition.

3.2 Feature Visualisation

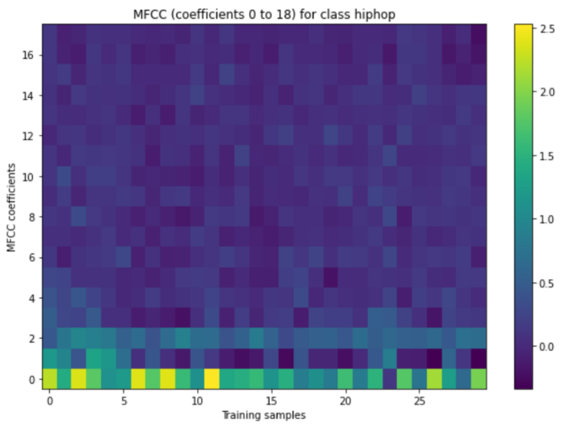
After having collected all the features and having stored them in the dictionaries, a feature visualization algorithm has been implemented. That is a piece of code which plots the features onto a graph and displays it on the screen.
This step is important for any kind of machine-learning application since having a viewable feedback from the calculator improves the research in many ways:

- Better understanding the workflow of the machine.
- Checking if the code is choosing the features appropriately.
- Having an intuitive insight about how the machine is making its decisions.

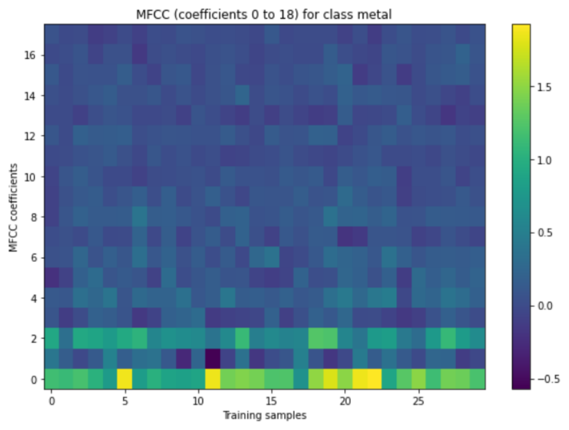
Since the algorithm developed during this work is only based on the Mel Frequency Cepstrum Coefficients (MFCC) feature, the only plottable feature is therefore the Cepstrum's distribution.
In particular, the code plots a graph for each class (genre); each graph represents the distribution and amplitude of the MFCC for each sample (song). The final results are shown in the pictures (a), (b), (c), (d).



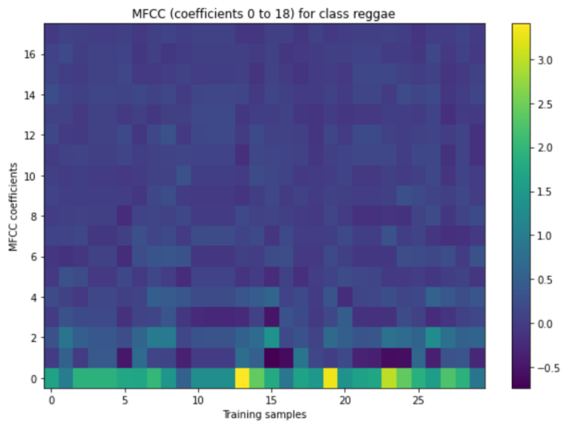
(a) Blues



(b) Hip Hop



(c) Metal



(d) Raggae

As it is possible to evince by looking at the images, the algorithm is printing the distribution of the cepstrum (y-axis) for each song (x-axis) taking into account also the magnitude for each coefficient (lighter colours correspond to higher coefficient's values).
Although at first glance they might look the same graph, some differences between the plots can be noted. For instance, at higher MFCCs the metal genre has higher coefficient's values than the other genres.
Furthermore, it is important to keep in mind that the step of visualising the features is only helpful to the user and not to the machine, which is capable of appreciating smaller variations in coefficient values than the human eye is capable of appreciating colour variations.

To conclude and better understand this part of the work, it might be useful to analyse the code, which can be found on the project's GitHub page intended to implement the visualisation.
By looking at it, the first thing one notices is that it consists basically of a for cycle iterated over the four classes; for every cycle there is therefore a plot.
Let us analyse which are the steps for each graph:

1. A new array called mfcc is created as the transposed vector of `dict_train_features`. This new vector will contain the order and the magnitude of the Coefficients.
2. A new figure gets instantiated under the name fig by the function `plt.figure()` which also sets the size.
3. The function `plt.subplot()` creates the layout for the plots of the other classes.
4. The function `plt.imshow()` actually plots the datas contained in mfcc.
5. All of the other lines of code are functions dedicated to add the details of the plots (i.e.: names of the axles, colorbar, etc.).

3.3 SVM Application

We finally arrive at the core of the machine learning classification process, the training of the model. First we have to do some considerations about the chosen of Machine type needed for classification problem. Like said before our intent is to make a pyramid scheme of binary decision distinguishing step by step which class an input belong to, so we need a chain of binary classificator machines, which under supervised learning process, are trained to best separated input features from a class to another.
We choose in particular a simple structured machine called Support Vector Machines, or SVM, which best fits that purpose.
A support vector machine constructs a hyper-plane or set of hyper-planes in a high or infinite dimensional space, which can be used for classification, regression or other tasks. Intuitively, a good separation is achieved by the hyper-plane that has the largest distance to the nearest training data points of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.
SVM capable to perform a binary or multi-class classification on a Dataset are also called C-Support Vector Classification or SVC.
We implement this model using `scikit` library for python using the function `clf = svm.SVC()`.

The matematical rappresentation of SVC is the following:
Given training vectors $x_i \in \mathbb{R}^p$, i=1,..., n, in two classes, and a vector $y \in \{1, -1\}^n$, our goal is to find $w \in \mathbb{R}^p$ such that the prediction given by sign $(w^T\phi(x) + b)$ is correct for most samples.
SVC solves the following problem:

$$\begin{aligned} \min_{w,b,\zeta} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \zeta_i \\ \text{subject to } y_i(w^T\phi(x_i) + b) \geq 1 - \zeta_i, \\ \zeta_i \geq 0, i = 1, ..., n \end{aligned} \tag{2}$$

Intuitively, we're trying to maximize the margin (by minimizing $||w||^2 = w^T w$), while incurring a penalty when a sample is misclassified or within the margin boundary. Ideally, the value $y_i(w^T\phi(x_i)+b)$ would be ≥ 1 for all samples, which indicates a perfect prediction. But problems are usually not always perfectly separable with a hyperplane, so we allow some samples to be at a distance ζ_i from their correct margin boundary.
The penalty term C controls the strengh of this penalty, and as a result, acts as an inverse regularization parameter. Another important parameter is $\phi(x_i)$ also called kernel it determine the shape of the hiperplane that divides features, the impact on the division is showed clearly in the Figure 4.

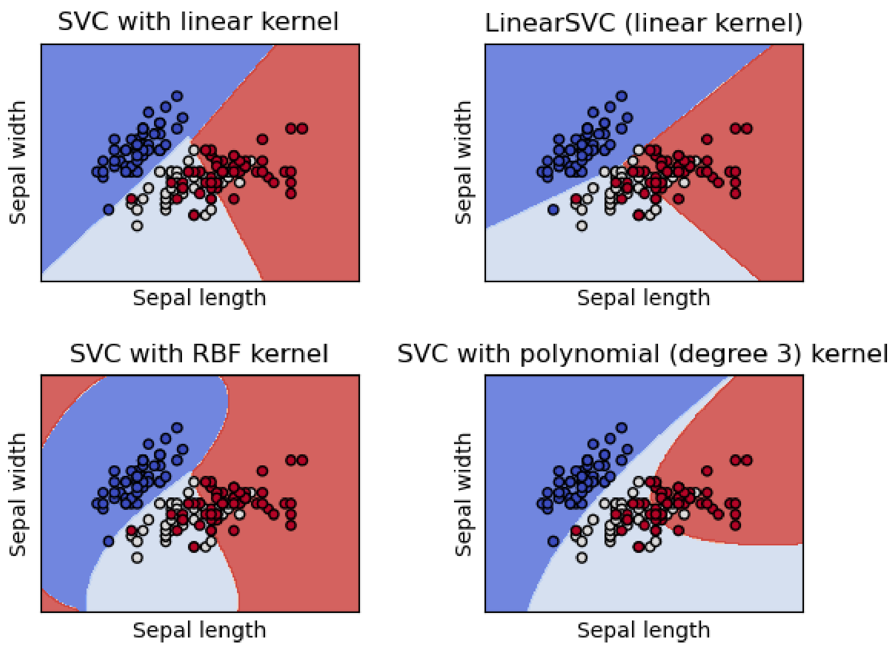


Figure 4: Example of different kernel impact on SVC

For setting up CVS we can interact with two main parameters:

- Set penality term C = 1
- Set kernel $\phi(x_i)$ = 'rbf', means Gaussian shape.

Once the optimization problem is solved, the output of **decision function** for a given sample x becames:

$$\sum_{i \in SV} y_i \alpha_i K(x_i, x) + b, \tag{3}$$

and the predicted class correspond to its sign. We only need to sum over the support vectors (i.e. the samples that lie within the margin) because the dual coefficients α_i are zero for the other samples. After being fitted, the model can then be used to predict new values with the function `clf.predict()`, which in binary classification case, set the predicted output to 1 if the prediction probability is greater than 0.5 and 0 viceversa.

4 Confusion matrix and accuracy

At the end of the testing process, it is important to evaluate how accurate the model is in recognising the musical genre of each analysed audio track.

In the case of multiclass configuration, the easiest way to understand the performance of the model is to build a confusion matrix. This particular matrix simply counts how many times each class is classified as one of the considered classes. In our case we obtain a four by four matrix in which the rows correspond to the actual genre and the columns to the predicted genre. The sum of each row corresponds to the number of samples considered.

To understand if the model is accurate, we observe the main diagonal of the confusion matrix. The higher the values on the main diagonal of the matrix, the more reliable is the model. The other numbers show that the misclassification of the system are similar to what a human would do.

We can calculate the percentage of correct classification by dividing the values on the main diagonal of the matrix by the number of analysed samples and multiplying by 100. The percentages will give us a more clear idea on the model accuracy.

	Blues	Hip-Hop	Metal	Reggae	**Accuracy**
Blues	15	4	1	10	**50%**
Hip-Hop	1	24	1	4	**80%**
Metal	1	10	15	4	**50%**
Reggae	3	17	4	6	**20%**

Table 1: Confusion matrix

We can see that not for all the considered musical genres the implemented SVM technique is able to make a precise classification. The SVM had a success rate of 80% when identifying Hip-Hop, 50 % when identifying Blues (frequently confused with Reggae) and Metal (frequently confused with Hip-Hop), but only 20 % when identifying Reggae which was largely misclassified as Hip-Hop.

The evaluation of the results from automatic genre classification systems is not as simple as it might seem, in particular if the considered musical genres have similarities in the typical instruments used or in their origins. Just because the classification of a given song does not agree with a given ground truth classification does not necessarily mean it is wrong. Given the subjective nature of genre classification, and how artists sometime merge two or multiple known genres, there are many situations where two or more genres might be appropriate for a given song.