CMLS - HW1 Report

Armas Monroy Claudio César
Cicognani Roberto Leone
Lorenzi Davide
Sartori Matteo

April 26, 2021

Contents

Chapter 1

HW1 - Genre Classification

1.1 Homework

The aim of Assingnment 1 is to create a Python script able to classify a song dataset into four different musical genres: Rock, Pop, Reggae and Disco. The script must also be able to plot a confusion matrix and other metrics.

The GitHub repository of the code is available at this link:

https://github.com/DavideTommy/Music_Categorizer-CMLS_HW1.git

The repository is Private so, if needed, just send an e-mail to us and we'll give immediate invite to the repo.

1.2 Pre Processing

Our dataset was downloaded from the GZTAN database. In order to make it as uniform as possible we processed it using MediaHuman Audio converter. We turned the .au audio dataset in to a modern .mp3 one with the following audio specs:

- **Sample Rate:** 22050 Hz, like the original.
- **Bitrate:** 160 kbits, like the original.
- **Channels:** from undefined to mono. Listening to the tracks we assumed also .au tracks were in mono.

After that we used wxMP3gain in order to level all tracks at 89 dB, because several of them featured clipping phenomena.

1.3 Idea

We decided to use an SVM classifier to extract the confusion matrix. For four genres, we implemented 6 non-redundant SVM, in order to cover all possible genre combinations, which are:

- 0-1 Rock - Pop
- 0-2 Rock - Reggae
- 0-3 Rock - Disco
- 1-2 Pop - Reggae
- 1-3 Pop - Disco
- 2-3 Reggae - Disco

In this way we are able to obtain a full confusion matrix.

To improve our feature analysis, we designed a feature dictionary composed by:

- MFCC
- Chroma mean
- Spectral Roll
- Spectral Centroid

1.4 Implementation

We implemented the code on ipython notebook, but we also encoded it on a .py file into PyCharm IDE, setting as default environment the same (CMLS) used on ipython.

We extended the code from the "Multi Genre SVM" seen in class with the same computation order:

1. computation of mfcc for all files, beginning with the elaboration of the spectrogram and the application of mel filtering using the Librosa library.
2. training of the algorithm, setting up the dictionary of features.
3. testing of the dictionary and normalization of results.

4. cross comparison of tracks for each SVM unit, with a majority vote system.
5. computation and printing of the multiclass confusion matrix

1.5 Feature Extraction

Our script aims at classifying tracks based on specific audio descriptors. In this case we used the capabilities of python libraries to extract specific low-level features from the audio files in the GZTAN database. These descriptors are:

- Mel-frequency cepstrum (MFCC): This descriptor is useful to describe timbral characteristics of audio. Its implementation on a digital audio track can be summarized as:
    1. Windowing of the signal
    2. Fast Fourier Transform (FFT) of each window
    3. Applying a mel filter bank on the resulting spectrum Base-10 logarithm
    4. Discrete Cosine Transform
- Chroma frequencies
- Spectral roll-off
- Spectral centroid

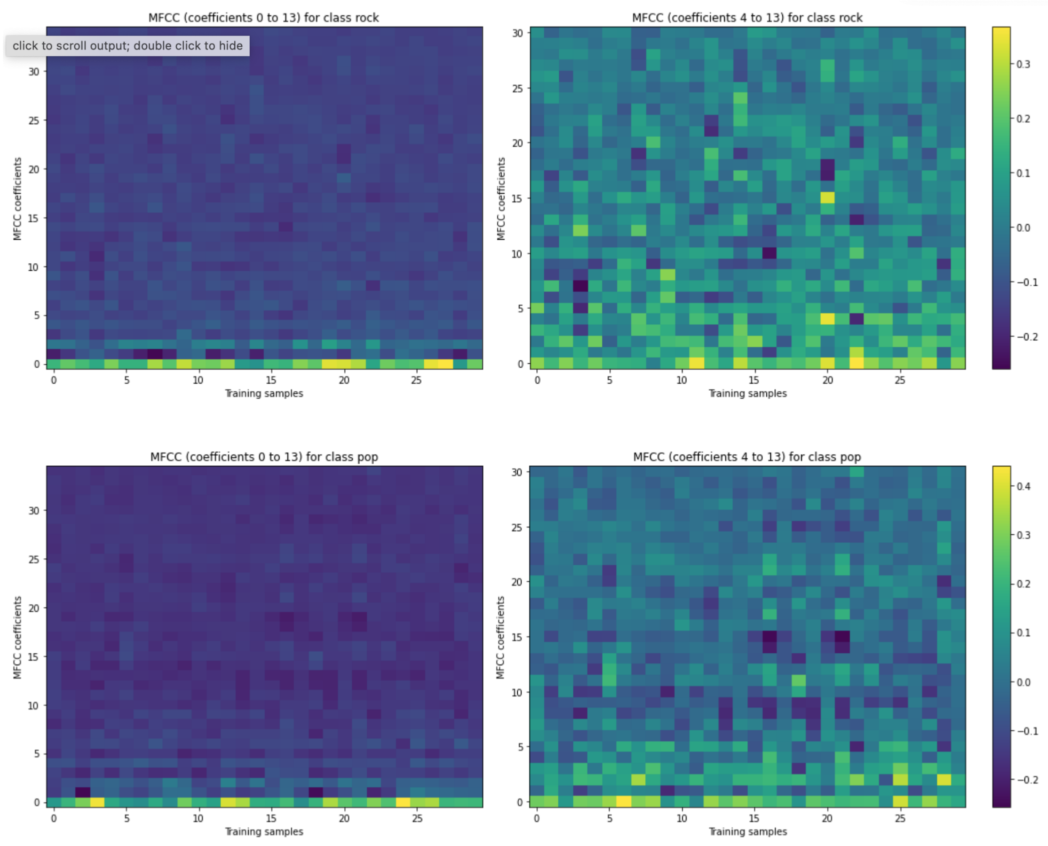
In particular, the mel filter bank consists of a series of triangular filters whose centers are based on the mel scale in order to better approximate the human perception of the spectrum. Chroma frequencies: Lets visualize the tonal content of a musical track. Spectral roll-off: Represents the frequency under which 85% of the total spectral energy lies. Spectral centroid: Basic spectral descriptor that represents, for each window of a signal, the "center of gravity" in terms of frequency. May vary from genre to genre.

1.5.1 Code implementation

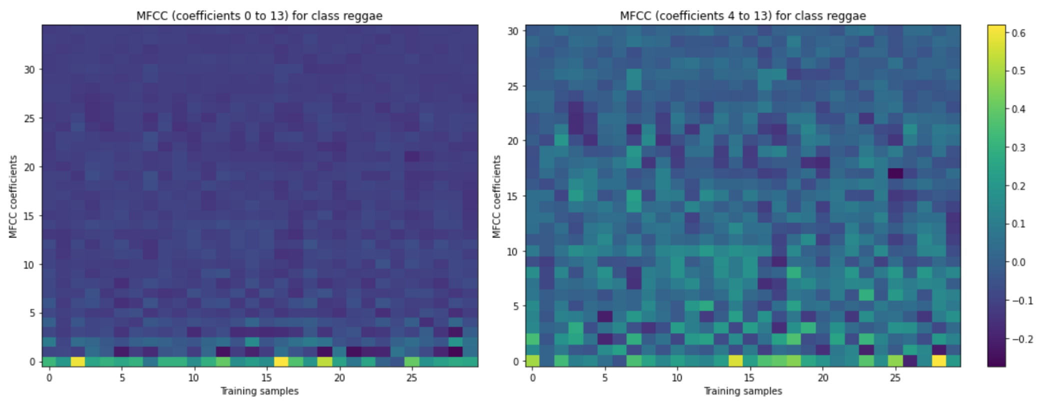
We decided to implement the MFCC feature from scratch. We defined the function `compute_mfcc` with the procedure described above; in particular, we used `librosa.stft` to apply hamming windowing and FFT. The other four features are found in the librosa library and we decided to import them directly into our code. After defining two sets of musical tracks (one for training set the other one for testing), the script performed the feature extraction for each of them and stored the data values in dictionaries containing four matrices, one for each musical genre. Each track produced a vector of 49 elements: 35 MFCC factors, 12 chroma factors, 1 spectral roll-off and 1 spectral centroid. Each factor is the mean value computed on all the windows of the audio track.

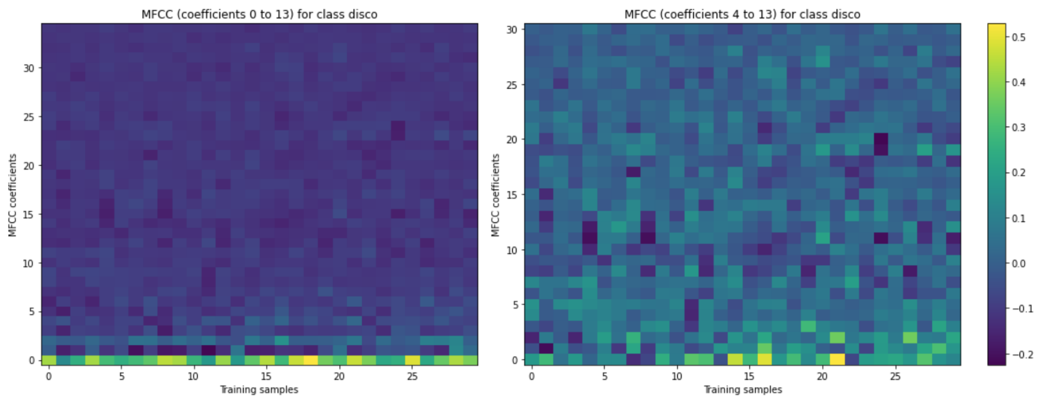
1.6 Feature Graphical Description

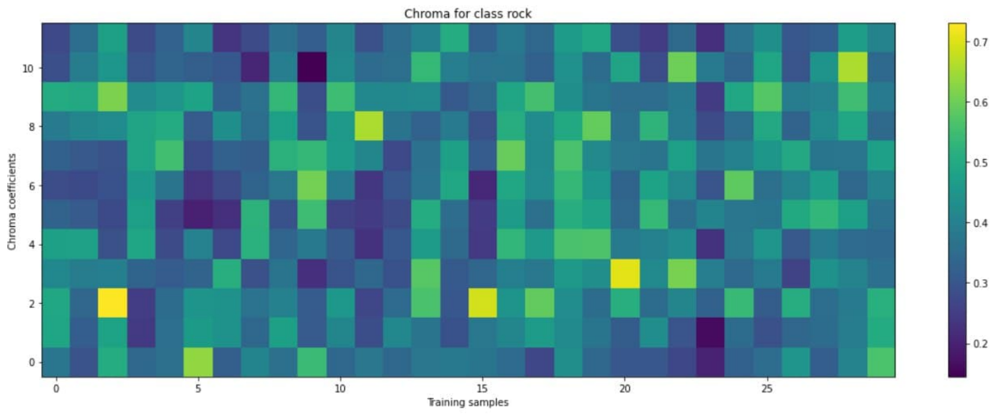
While having computed the Mel Frequency Cepstrum Coefficients and the Chroma frequencies we wanted to have a graphical representation of them. However, doing this exploiting the matrix `train_features` , in which all the different features were concatenated, was not the correct choice, making difficult to isolate them from spectral centroid, spectral roll-off and chromas. Thus, we decided to work with separate matrices for MFCC and chroma, called `train_mfcc_plot` and `train_chroma_plot` respectively. In the following lines we will describe the procedure regarding the MFCC plot which is analogous to the one for the chroma. After its initialization as `n_train_samples` rows and `n_mfcc` (equal to 35 for the MFCC case and 12 for the chroma) columns of zeros, we were ready to fill it in the same "for loop" (the internal one) used for computing the features. For each iteration, so for each audio file, we plot the graph. Meanwhile, in the external loop, we associated the corresponding values contained in this vector to each one of the four classes, so the four music genres, and we did this exploiting a dictionary matrix `mfcc_plot` pre-inizialized in such a way it is totally equal to the one used for computing the features. In fact in each one of the four iterations we filled a row of `mfcc_plot`. In this way we avoid representing the mean value of the MFCC for each song, which would have led to too many pictures appearing and we definitely speeded-up our code in this way. Moreover, the subdivision by class is a clever criterium of representation showing us how this feature changes for each different musical genre. Then, in the section "feature visualization", with another loop iterating on the classes, we finally plotted the MFCC matrix. The `transpose` operator is necessary because `mfcc_plot` has inverted axis with respect to the plot. Matching with the dimensions of the matrix, on the x-axis of the plot we have the training samples and, on the y-axis, the MFCC. On the left we have the representation of all coefficients, while on the right of the coefficients starting from the fourth and going on. This was possible with the function `subplot`. We obtained following MFCC

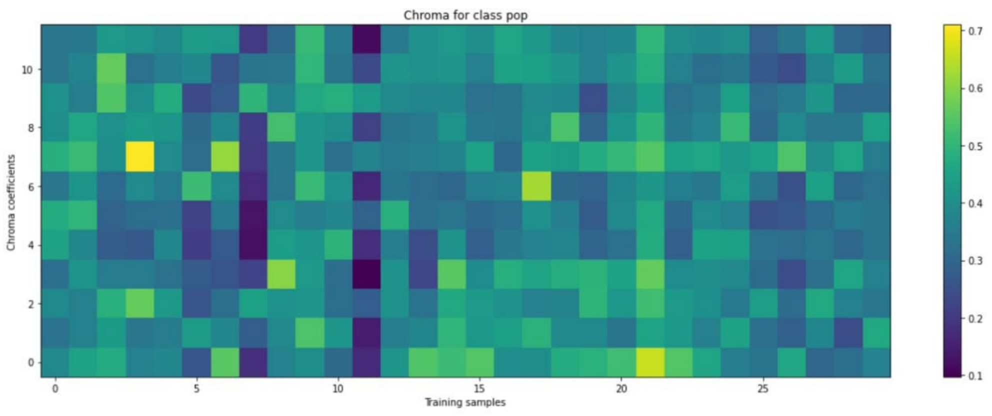


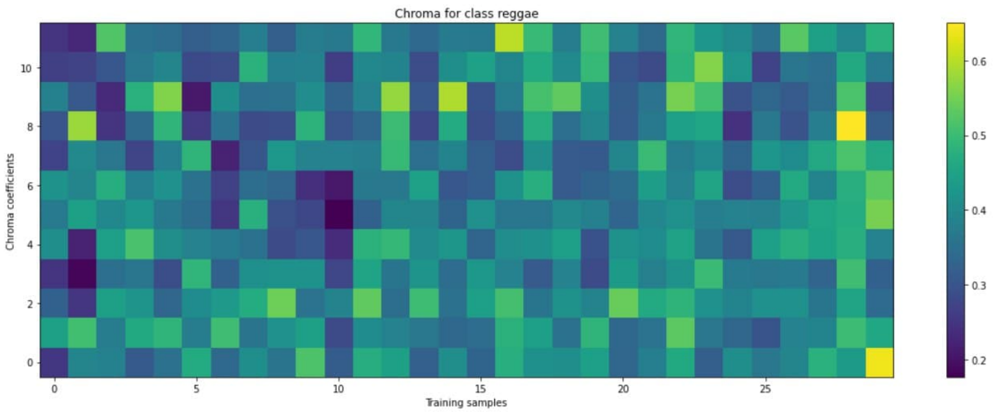
graphics (first four images) and the following Chroma Spectre Rapresentation (last four images)

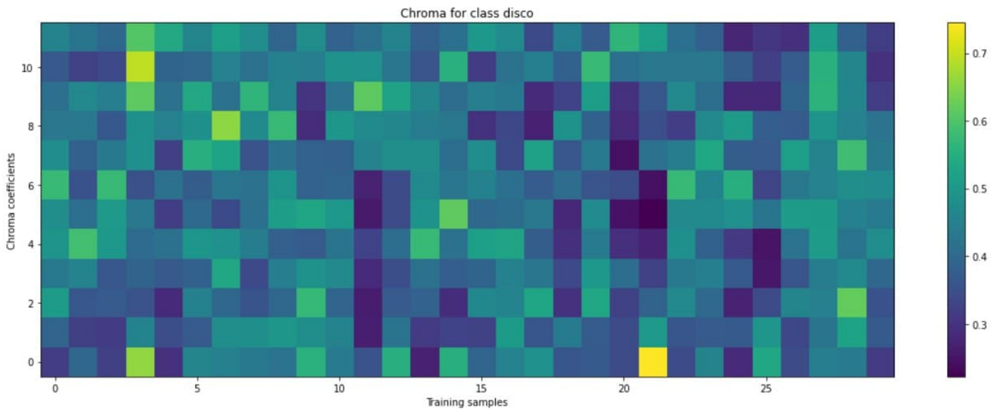

MFCC (coefficients 0 to 13) for class reggae
MFCC (coefficients 4 to 13) for class reggae
MFCC coefficients
Training samples


MFCC (coefficients 0 to 13) for class disco
MFCC (coefficients 4 to 13) for class disco
MFCC coefficients
Training samples


Chroma for class rock
Chroma coefficients
Training samples


Chroma for class pop
Chroma coefficients
Training samples


Chroma for class reggae
Chroma coefficients
Training samples


Chroma for class disco
Chroma coefficients
Training samples

1.7 SVM Classification

Support Vector Machine (SVM) is a binary classifier that learns the boundary between various items belonging to different classes, and in this case, we have used this tool to classify various audio tracks into four different musical genres, respectively: rock, pop, raggae and disco. Then, we have to see how accurate the classification is. In particular, in order to achieve an optimal classification, we used the SVN to classify the audio signals, according to the specific parametric function used to cluster the feature space. Each cluster refers to a single acoustic event (in our case is the musical genre). The algorithm listens and analyzes each song belonging to the testing set to recognize the musical genre based on the training set that has been performed a priori, and the feature space is partitioned by hyperplanes. For reaching our goal, we subdivided the four musical genres into four classes. Then, we created four respective training set for each genre. Then we created four different matrixes whose dimensions are the number of elements in each dimension of the training features ones, i.e. `dict_train_features` and respectively we obtained:

- one matrix of zero for the rock genre
- one matrix of zero for the pop genre
- matrix of zero for the reggae genre
- matrix of zero for the disco genre

This procedure has also been made for the testing set, obtaining same matrices. So we linked all the four training features matrixes to establish the minimum and maximum values (`feat_min` and `feat_max`), in order to perform the normalization of the features for succeeding in our mission. To normalize both training features and testing features matrixes we have subtracted (`feat_min`) from every genre features matrix, and then divided for the subtraction between `feat_max` and `feat_min`. Notice that SVM is a binary classifier in nature and we have four classes, so we needed to rearrange the problem. Thus, we needed to obtain a tree model, to work with binary classifiers and we succeeded in it thanks to the `sklearn.svm.SVC` function. We used the `model.fit` function to train our model and then we evaluated each classifier. We had three predictions, from which we extracted the most present.

1.8 Conclusions

From the original 100 song-dataset we decided to use 30 tracks for training and 70 tracks for testing.

The confusion matrix presents predominantly high values on the major diagonal, as it has to be for a correct identification of the genre of the songs.

$$\begin{bmatrix} 34 & 2 & 12 & 22 \\ 9 & 42 & 10 & 9 \\ 37 & 0 & 19 & 14 \\ 16 & 8 & 8 & 38 \end{bmatrix}$$

We achived an acuracy of:

- 48.58% for rock
- 60% for pop
- 27% for reggae
- 54.28% for disco

Using a 20 - 80 partition we obtained very similar results in terms of percentage. This means that the algorithm has a pretty good accuracy.

We obtained low values for Reggae because we didn't find the right features to identify it. The algorithm identified reggae song as rock song in most of case. This should be explained by the instruments that compose the sound of reggae and rock. In both of them are guitar and voices are the main features, so the algorithm can be fooled by this.

We can obtain better results with:

- **Better Datasets**: datasets containing longer tracks or of better quality can help the algorithm in recognising song. We tryied this adding songs of our personal collection to the train dataset with an improvement of results.
- **Deep Learning Algorithms and Neural Network**: looking around for inspiration we've found that the implementation of Deep Learning Algorithms and Neural Networks allows to obtain an accuracy very high, more than 84%.
- **Bigger Datasets**: With a bigger dataset is possible to define with more accuracy all genres, allowing a better recognition of the songs.
- **Enhanced Pre-Processing**: cleaning and elaborating tracks in a proper way, removing noises, can help the algorithm not to confuse noise and sound, with better defined characteristics.