NAML Project Report - Group 14

Music Genre Classification using
k-Nearest Neighbours
Nearest Centroid
Multiclass SVM
-

by Silvia Marino (codice persona) and Francesco Panebianco (10632465)
DEIB - Politecnico di Milano

February 2022

Contents

1 Introduction **2**
Scope . . . 2
The Dataset . . . 2
Feature Extraction . . . 4
Dataset Visualization . . . 5
Alternative Feature Set: Mel-Frequency Cepstral Coefficients . . . 7

2 Classifiers **9**
k-Nearest Neighbours Classifier . . . 9
Nearest Centroid Classifier . . . 9
Multiclass SVM Classifiers . . . 9
Building Blocks . . . 9
One-To-Rest Classifier . . . 12
One-To-One Classifier . . . 12

1 Introduction

Scope

The scope of this project is to create a music genre classifier using the machine learning algorithms *k-Nearest Neighbours* and *Multiclass SVM*, which were assigned to our group. The project is part of the evaluation of the "Numerical Analysis for Machine Learning" course, which is part of the first semester of the first year of Master's Degree in Computer Science and Engineering at Politecnico di Milano. Given the similarities between *k-Nearest Neighbours* and *Nearest Centroid*, we chose to implement the latter as well, comparing its performance to the former, even though it is outside the specification of the project.

The Dataset

The dataset assigned to our project is the notorious *GTZAN Genre Collection*[3], which contains 100 different extracts from 10 different music genres, provided in .wav (Waveform Audio File Format). The genres considered are:

- Blues
- Classical
- Country
- Disco
- Hip Hop
- Jazz
- Metal
- Pop
- Reggae
- Rock

As it can be seen, genres that share similarities are included in the dataset (e.g. Blues and Jazz), but also dramatically different types of music such as Rock and Classical, which we expect the algorithms to classify with higher precision.

The dataset provides audio samples as *.wav* files. WAV is the most common uncompressed audio file format in Microsoft Windows systems. It was developed by IBM and Microsoft, for storing an audio bitstream on PCs[1].

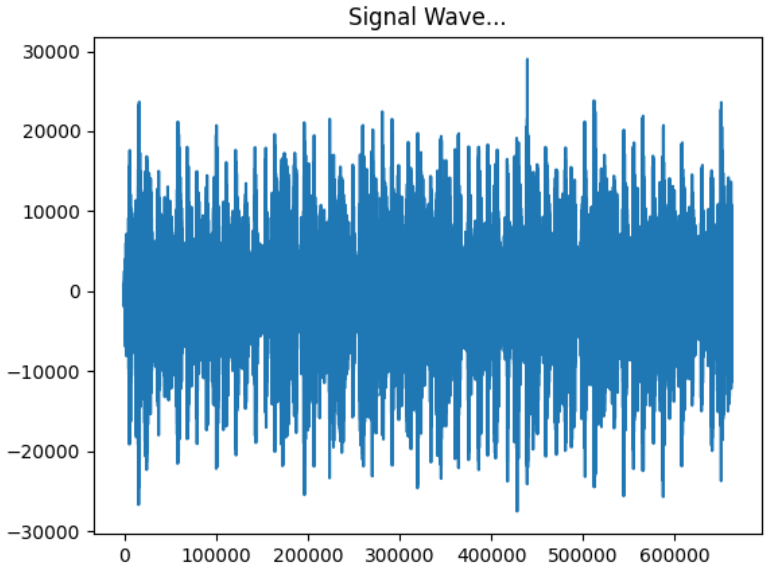

Figure 1: Plotted waveform of blues.00000.wav

This kind of files are managed in Python by using the *wave* module, which provides convenient functions to work with the WAV sound format. In our case, the audio samples are provided as 22050Hz Mono 16-bit 30 second tracks[3], which are easily transformed into a numpy array of *int16*.
The picture above is the result of the following code:

```
import wave
import matplotlib.pyplot as plt
import numpy as np

test_file = wave.open('genres/blues/blues.00000.wav', 'rb')
# Extract Raw Audio from Wav File
signal = test_file.readframes(-1)
signal = np.frombuffer(signal, dtype='int16')
test_file.close()

plt.figure(1)
plt.title("Signal Wave...")
plt.plot(signal)
plt.show()
```

Feature Extraction

For what concerns the features required to classify the genres, our first choice was the triplet *Zero Crossing Rate*, *Average Energy* and *Silent Ratio* as defined in one of the papers provided by the project specification [2]. The extracted features were collected and exported in a csv file to avoid having to extract the feature multiple times, as the complete process took approximately 45 minutes on *Google Colab*.

Zero Crossing Rate
It indicates the frequency of signal amplitude sign change, which is in some way related to the average signal frequency. In practice, it is a key feature to classify percussive sounds as it's often correlated with the beat. ZCR records how many waves have passed for a certain time, by giving a positive amplitude a positive value (1) and a negative amplitude a negative value (-1). The implemented formula is as follows:

$$\mathrm{ZCR} = \frac{\sum_{n=1}^{N} \left| \mathrm{sgn}\ x(n) - \mathrm{sgn}\ x(n-1) \right|}{2N} \tag{1}$$

where sgn x is the *sign function*.

Average Energy
It indicates the loudness of the audio signal as a whole, being the average of the square amplitude of the audio signal.

$$\mathrm{E_{avg}} = \frac{\sum_{n=0}^{N-1} x(n)^2}{N} \tag{2}$$

In practice, it corresponds to the momentum or the force of the music within the time slice where it is measured.

Silent Ratio
It indicates the proportion of the sound piece that is considered to be *silent*. Silence is defined as a period within which the absolute value of amplitude is below a certain threshold. If we indicate

$$SR = \frac{\sum_{x(n)<\mathrm{thr}} 1}{N} \tag{3}$$

In our implementation, silence threshold is derived from the average energy as follows...

$$\mathrm{thr} = 0.8\sqrt{E_{\mathrm{avg}}} \tag{4}$$

Dataset Visualization

After exporting the feature dataset, we visualized it using built-in functions from *pandas* and *seaborn*. Full dataset visualization is available in a the Jupyter Notebook called *NAML_Project_Data_Visualization.ipynb*.

	ZCR	AVERAGE_ENERGY	SILENT_RATIO
count	1000.000000	1.000000e+03	1000.000000
mean	0.103768	2.631159e+07	0.827613
std	0.041886	2.396840e+07	0.024517
min	0.021714	6.178678e+04	0.767521
25%	0.070328	9.781419e+06	0.810039
50%	0.099618	1.834412e+07	0.824969
75%	0.132136	3.768007e+07	0.842175
max	0.275001	1.777716e+08	0.926341

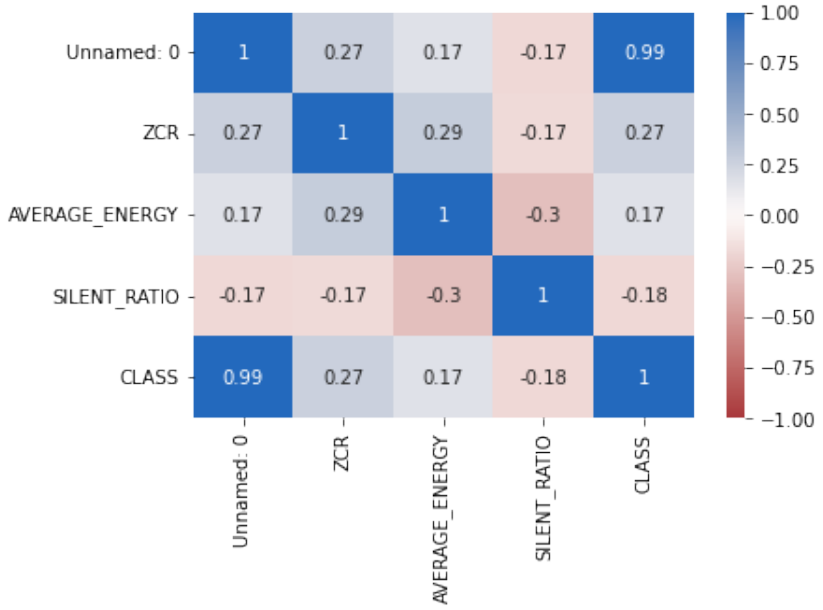

Figure 2: Annotated Heatmap of the correlation of extracted features

From this visualization, it is evident that the three features are independent enough to be non-redundant in the classification.

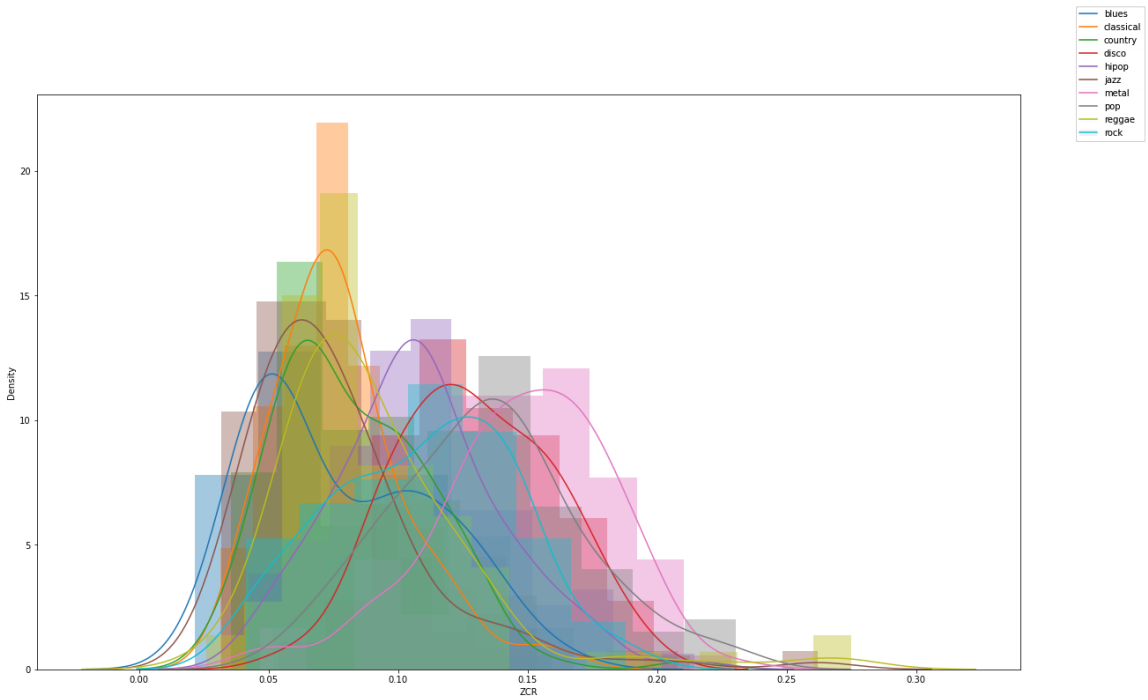

Figure 3: Distribution Plot of ZCR feature

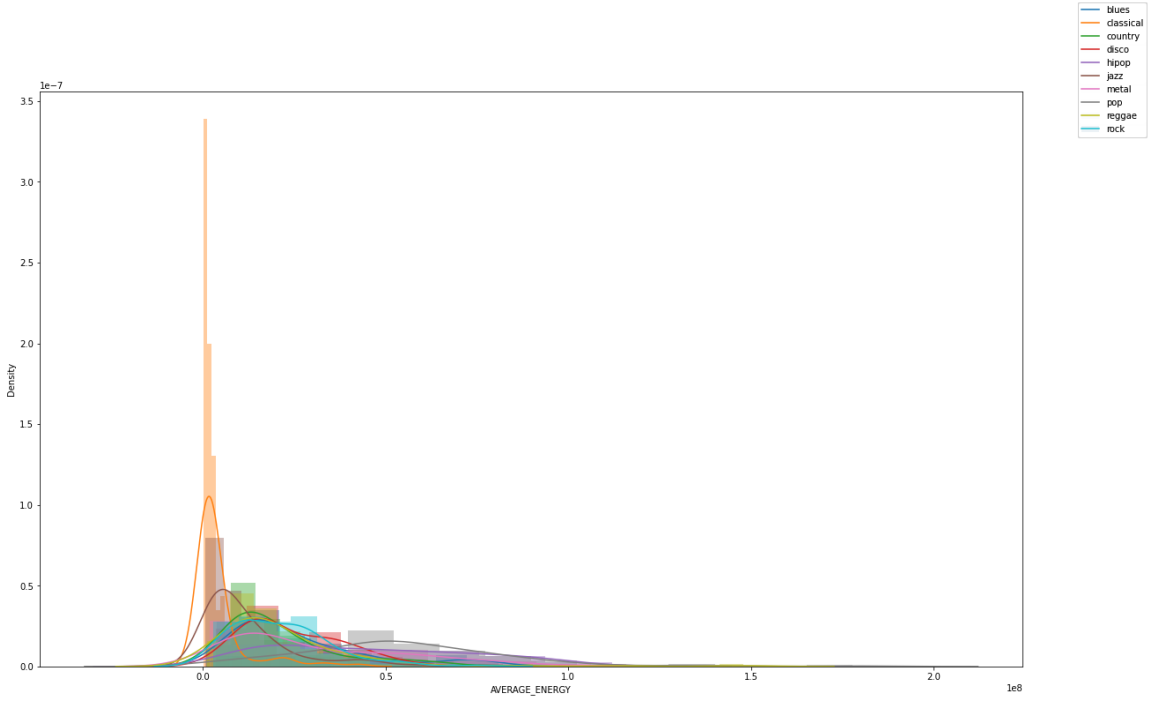

Figure 4: Distribution Plot of Average Energy feature

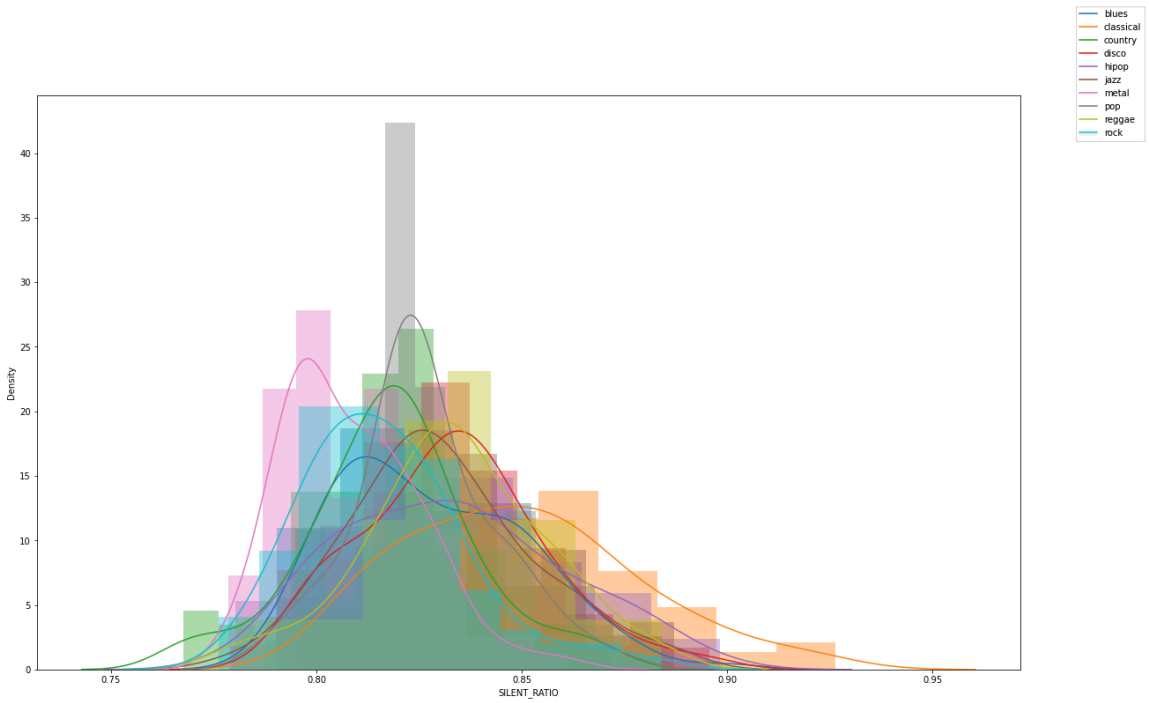

Figure 5: Distribution Plot of Silent Ratio feature

Alternative Feature Set: Mel-Frequency Cepstral Coefficients

The second paper provided by project specification [4] mentioned the possibility of using *Mel-Frequency Cepstral Coefficients* as features for music genre classification.

The **Mel-frequency Cepstrum** (MFC) is a representation of the short term power spectrum. The **Mel-frequency Cepstral Coefficients** (MFCCs) collectively make up an MFC. A Cepstrum is defined as a non-linear spectrum-of-a-spectrum.

The peculiarity of an MFC with respect to a generic cepstrum is, as the name suggests, the usage of the *Mel-frequency Scale*, which emulates the human hearing response more closely than the linearly spaced frequency bands used in the normal spectrum.

MFCC extraction is a relatively standardized algorithm, so we've chosen a popular python library to aid us in the extraction: *librosa*. As the library source code shows [5], the implemented extraction procedure is as follows:

1. Since a time series is provided, the spectrogram of the signal is computed.
2. The spectrogram is mapped to the mel scale
3. The log of the result is taken (dB conversion)
4. The *Discrete Cosine Transform* is applied

The Librosa implementation also supports a final processing step: Sinusoidal Liftering, which is reportedly considered beneficial to speech recognition applications. Since this is outside the scope of the classifier, no Liftering was applied.

The input signal is automatically divided into a certain number of windows, each with their MFCCs, most applications can deal with the first 13 MFCCs, as they carry the most relevant information for the section of the spectrum that can be heard by the human ear.

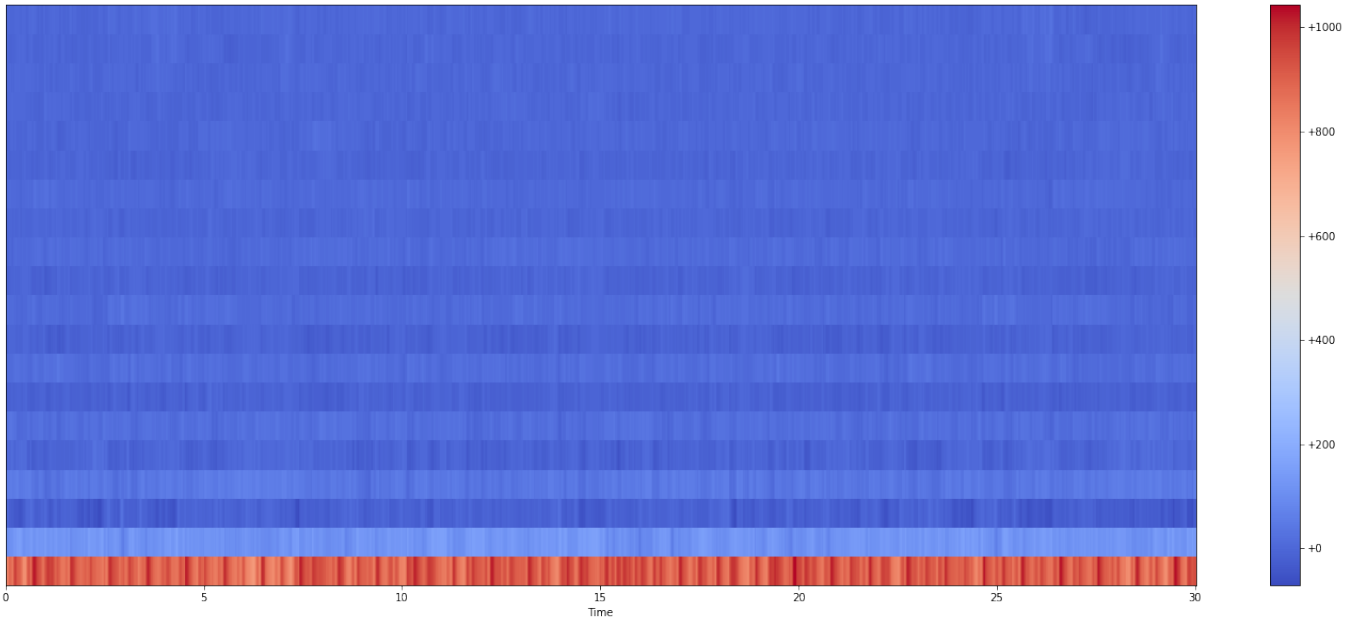

Figure 6: Mel-Frequency Cepstral Coefficents spectrogram of *blues.00000.wav*

Given that the resulting MFCCs of the signal are returned in a matrix with rows corresponding to the 13 coefficients and columns corresponding to all windows of the original signal, we chose to keep the mean and the variance of each MFCC across the signal as a feature for classification.

Librosa also allows for the computation of local estimates of n-order derivatives for MFCCs using the *librosa.feature.delta* function. Mean and variance of this such features (namely order 1 and 2) have also been kept for the classifier.

A specific Jupyter Notebook named *NAML Project Mel-frequency cepstrum.ipynb* handles the visualization and generation of the MFCC data for the dataset assigned to this project (the exported feature set is called *mfcc_dataset.csv*).

2 Classifiers

k-Nearest Neighbours Classifier

In this section it will be explained which methodologies have been used to implement the music genre classifier and how the code has been structure to achieve the results exposed at the end of the report

Nearest Centroid Classifier

Multiclass SVM Classifiers

The implementation of a *Support Vector Machine* classifier can be accomplished in a variety of different ways. To explore them before moving to the full Multiclass application, we created a Jupyter Notebook called *NAML_Project_Experiments.ipynb*. After testing the binary classification performance, we adapted the code for the multi-class case, exploring both the *One-To-Rest* strategy and the *One-To-One* strategy.

Building Blocks

The implementation of a Support Vector Machine problem can be either derived from the primal or the dual formulation.

Primal Formulation - Hard Margin
The problem of computing the *maximum-margin hyperplane* exploiting *support vectors* (samples from either class on the "gutter" that separates the two classes) can be formulated as follows:

$$\min_{w,b} |w| \qquad \text{s.t.} \quad y_i(w^T x - b) \geq 1 \tag{5}$$

Where $\mathbf{y_i} = \pm\mathbf{1}$ is the label of the class corresponding to sample $\mathbf{x}$.

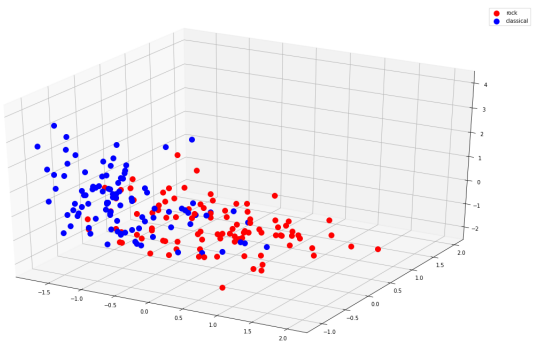

Figure 7: 3d scatteplot of rock and blues samples

Unfortunately, our samples are not linearly separable using a hard-margin, so this basic form of classification was not chosen.

Primal Formulation - Soft Margin
The soft-margin formulation involves the usage of the **hingeloss** function, which makes the optimization problem become

$$\min_{w,b} \quad \frac{1}{n}\sum_{i=1}^{n} \max\Big(0, 1 - y_i(w^T x - b)\Big) \tag{6}$$

The optimization problem is unconstrained, which enables us to apply *Gradient Descent* method, particularly appropriate considering the convexity of the function. Most soft-margin implementations also feature a *penalization* term, but in our implementation such parameter ended up decreasing the accuracy. Hence, we decided to remove it.

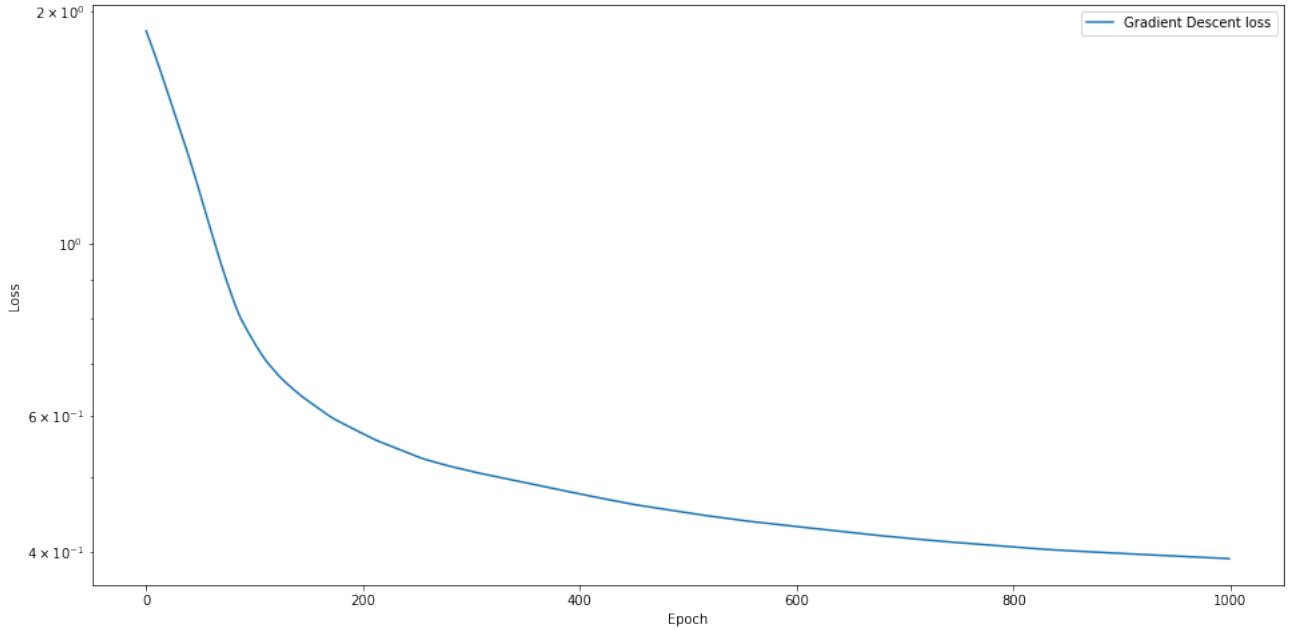

Figure 8: Gradient Descent hingeloss history on rock/classical margin

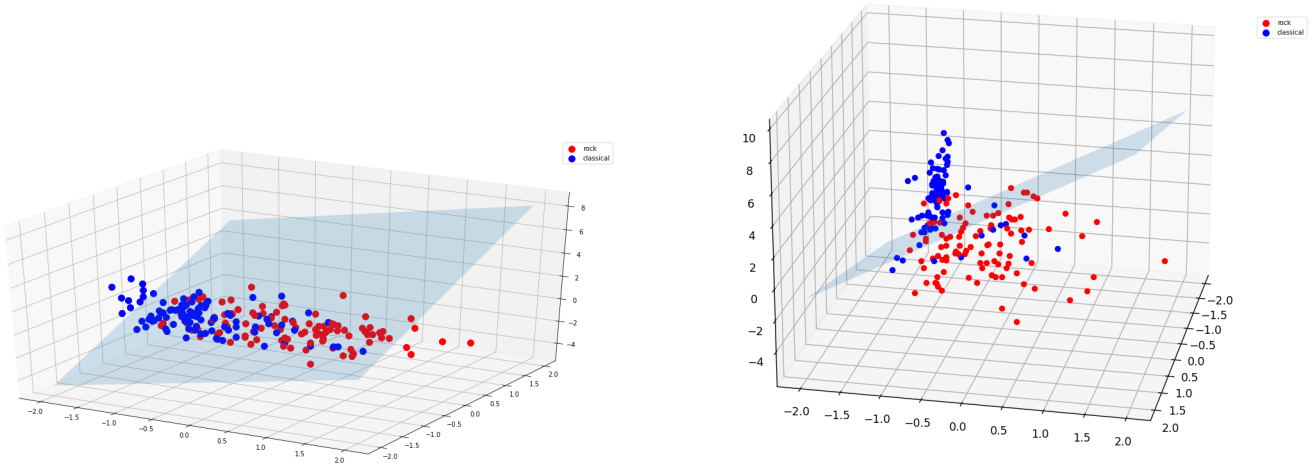

Figure 9: Plots of the hyperplane corresponding to the optimal parameters (two different rotations of the point of view)

Primal Formulation - Soft Margin with Feature Map

Before trying the dual formulation, which allows the usage of the Kernel Trick for non-linear classification margins, we tried a simple quadratic *feature map* to be applied to the primal formulation. This way, the three samples (ZCR, average energy and silent ratio) are mapped to a higher dimensional feature space.

$$\phi(x_1, x_2, x_3) = \begin{bmatrix} x_1 & x_2 & x_3 & x_1^2 & x_2^2 & x_3^2 & x_1x_2 & x_2x_3 & x_1x_3 \end{bmatrix}^T \tag{7}$$

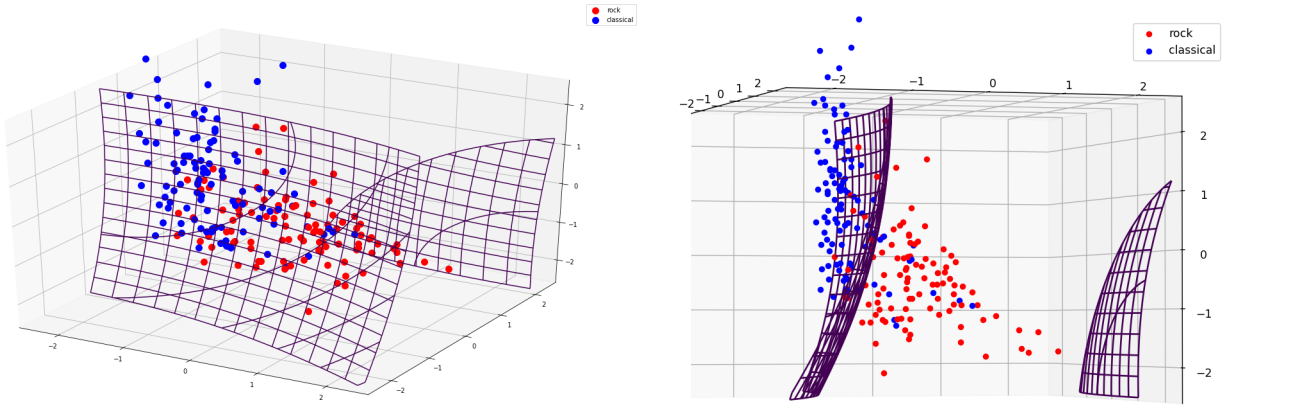

Figure 10: Plots of the non-linear margin corresponding to the optimal parameters (two different rotations of the point of view)

Dual Formulation - Linear Margin

The dual formulation of the SVM is easily obtained from the soft-margin primal formulation by using *Lagrangian Multipliers...*

$$\max_{c1,\ldots,c_n} \quad \sum_{i=1}^{n} c_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} c_i c_j y_i y_j x_i^T x_j \qquad \text{s.t.} \quad \sum_{i=1}^{n} c_i y_i = 0 \quad \wedge \quad c_i \geq 0$$

This time the optimization problem is constrained, so we had to use a *Quadratic Programming* library available for python. We ended up using *scipy.optimize*. Since the only constraint on the decision variable is linear, the computation of an iteration did not require the hessian, which was hardcoded to zero. The parameters of the optimization were method='trust-constr' (constrained minimization of scalar function) and jac='2-point' which enables the calculation of the jacobian of the constraint using finite differences [6]. Of course, since the dual formulation is a maximization problem, the objective function has to be multiplied by $-\mathbf{1}$ before being fed to the *scipy.optimize.minimize* method.

After running the computation, weights and bias can be derived from the following expressions

$$w = \sum_{i=1}^{n} c_i y_i x_i \qquad b = w^T x - y_i \ \text{ for any } i \text{ such that } x_i \ \text{ lies on the margin } (c_i > 0) \tag{8}$$

Dual Formulation - Kernel Trick

The kernel trick can intuitively be applied to the objective function by replacing the dot product of two samples $\mathbf{x_i^T x_j}$ with the kernel function $\mathbf{K(x_i, x_j)}$. The choice of the kernel function itself, unfortunately, was constrained by the performance of its computation. While a polynomial kernel was fast due to the matrix optimizations of *numpy*, a gaussian kernel was found expensive. The following kernel had the best results in terms of classification accuracy:

$$K(x_i, x_j) = (x_i^T x_j + 1)^3 \tag{9}$$

The computation of the weights (multiplied by the kernel) and the bias is as follows:

$$w_j^\phi = \sum_{i=1}^{n} c_i y_i K(x_j, x_i) \qquad b = w_i^\phi - y_i \ \text{ for any } i \text{ such that } x_i \ \text{ lies on the margin } (c_i > 0) \tag{10}$$

One-To-Rest Classifier

One-To-One Classifier

References

[1] Fleischman E. *WAVE and AVI Codec Registries*. 1998. URL: https://datatracker.ietf.org/doc/html/rfc2361.

[2] Tamatjita Elizabeth Nurmiyati and Mahastama Aditya Wikan. "Comparison of Music Genre Classification Using Nearest Centroid Classifier and k-Nearest Neighbours". In: *International Conference on Information Management and Technology (ICIMTech)* (2016), pp. 118–123. DOI: 978-1-5090-3352-2.

[3] Leben Jakob. *Music Analysis, Retrieval and Synthesis for Audio Signals*. URL: http://marsyas.info/downloads/datasets.html.

[4] Cast John, Schulze Chris, and Fauci Ali. *Music Genre Classification*. Tech. rep. Stanford University - Computer Science Machine Learning course CS229, 2013.

[5] *Librosa Github Repository: MFCC extraction details*. 2015. DOI: 10.5281/zenodo.4792298. URL: https://github.com/librosa/librosa/blob/main/librosa/feature/spectral.py.

[6] *Scipy Optimization and root finding*. URL: https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html.