~~Abstract / Resum / Resumen~~

1. intro

1.1 history digital audio

1.2 history audio classif.

1.3 sota audio classification

1.4 summary of the proposal here (que vas a hacer y pq)

2. Method

~~2.1 MFCC~~

~~2.2 Feature vector representation~~

~~2.2.1 naive todo concatenado~~

~~2.2.2 histogramas por componentes~~

2.3 Dimensionality reduction – PCA

2.4 Feature relevance (dec. trees based)

3.Evaluation design

3.1 Dataset (cuantos generos)

3.2 Evaluation protocol (cuantos datos train, test fold, accuracy como metrica de evaluacion...)

3.3 Methods and parameters

4. Results

4.1 Classication results (graficas.... matriz de confusion y que generos se confunden más etc.)

4.2 Feature relevance analysis

5. Conclusions

References

**Resum**

La música és una forma d'art que ens acompanya dia a dia i, amb l'aparició de serveis en línia com Spotify o Tidal, l'anàlisi musical s'ha tornat crucial perquè aquests serveis puguin recomanar nova música als usuaris, així com classificar totes les noves cançons que són pujades cada dia.

En aquesta dissertació, construirem un programa capaç de clasificar cançons pel seu genere mitjançant algoritmes de machine learning.

**Resumen**

La música es un arte que nos acompaña a diario y, con la aparición de servicios online como Spotify o Tidal, el análisis musical se ha convertido en algo crucial para que estos servicios puedan recomendar nueva música a los usuarios, así como clasificar todas las nuevas canciones que son subidas cada día.

En esta disertación, construiremos un programa capaz de clasificar canciones por su género mediante algoritmos de machine learning.

**Abstract**

Music is a form of art that accompanies all of us every day and, with the appearance of online services such as Spotify or Tidal, music analysis has become crucial to these services to recommend new music to users and to classify all new tracks uploaded every day.

In this dissertation, we are going to build a program capable of classifying audio tracks by its genre using machine learning algorithms.

**1. intro**

**1.1 history digital audio**

Even though digital audio became available in 1938 as telephone technology, it wasn’t until the 60s that mankind was able to record digital audio and store it in a computer.

Digital audio became possible after Harry Nyquist and Claude Shannon discovered what was known as Nyquist-Shannon Sampling Theorem, which was also discovered by E. T. Whittaker, Vladimir Kotelnikov and others whose name hasn’t been catalogued.

This theorem was, and still is, used to convert an analog signal (continuous) into a digital signal (discrete), dividing the analog signal into smaller pieces called “samples” and analysing every sample to get a value, that will represent all frequencies in the signal.

Years later, in the 1950s and 1960s, the technology to record digital audio kept improving, but it was still too expensive to be used for the great public.

It wasn’t until the 70s, that digital audio started to become mainstream, thanks to Thomas Stockham who, in 1976, built which is considered the first digital audio recorder: a 4-channel, 16-bit system that sampled at 50KHz.

Years later, in 1982, Philips and Sony released the CD, which allowed audio to be distributed easily, but it wasn’t until the mid-80s, thanks to companies such as Mitsubishi and Sony, released the first digital audio recorder into the mainstream market.

In 1933, one of the most popular audio formats was invented: mp3, which allowed reducing audio size and making files more portable.

After that, and thanks to the release of the first iPod in 2001 and its success, digital audio became portable and easy to listen for almost everyone.

**1.2 history audio classification**

Even though digital audio has been around for quite some time, music classification started two decades ago.

**1.3 sota audio classification**

**1.4 summary of the proposal**

**2. Method**

**2.1 MFCC**  
After investigation, we found out that most of the projects involving audio analysis were using MFCC to extract features from the audio files.

MFCC (Mel Frequency Cepstral Coefficients) is usually used to extract features from human talk, but has been used lately for all kinds of sound.

MFCC were defined by Paul Mermelstein and S. Davis in 1980.

Although it was first developed to recognize monosyllabic words in spoken form, its characteristics make it useful for all kinds of sounds.

The algorithm works as follows:

1. Divide the signal in several same-sized intervals.  
   This step will take the audio file and segment it into frames of the same size. The size of the frame will depend on the characteristics of the file, but it usually uses a frame of 20 to 30 ms.
2. Take the Fourier Transform of each interval.  
   Fourier Transform will take the frequencies of the interval and decomposes it into a finite domain of components that form the original signal.
3. Convert the values to Mel Scale.  
   Once we have taken the Fourier Transform, we have to map the values into mel scale. This scale represents pitches which, when being judged by listeners, will be of equal distance. [link to example]  
   To convert the frequencies (Hz) we get from the last step to mels, we use the following formula:

m=2595log10(1+f/700)

1. Take power logs of each mel frequency.
2. Apply the discrete cosine transform (DCT) to all Mel logs.  
   Now, in order to convert the values back into the time domain, we need to apply the discrete cosine transform to all values.  
   This is done using the following formula:



The resulting values will be MFCC.

Using this, we will end up with a matrix which size will be determined by the number of coefficients we want and the length of the audio sample.

**2.2 Feature vector representation**   
Once we have extracted the features using MFCC, we have to decide what are we going to do with them, given that the amount of features we get will always be, in our case, bigger than the dataset we can work with.

We will work with two different representations of these features: using all the raw data and creating histograms of each component.

**2.2.1 Naïve**  
The first method we will try will use all the values we get from MFCC.

This method will take the matrix whole matrix and convert it into a 1-dimensional array, created by concatenating each row, which size will depend on the length of the song, one after another.

This way, we will have our dataset converted into a matrix of as many rows as songs it has by the length of each array.

The amount of information we will have to work with will be enormous, but we will use it to have a first approximation of the accuracy of our classifier.

**2.2.2 Histograms**  
As we said before, we want to reduce the amount of values we have, but being able to still have the most information we can, as well as remove the effect of time in our experiment.

In order to do that, we will have as many histograms as coefficients we use, and will be built following this procedure:

1. Take maximum and minimum values of all dataset.
2. Divide the interval in as many steps as you want.
3. Create a histogram for each coefficient.
4. Put every value of the corresponding row into its interval.
5. Divide every final value by the amount of values you have.
6. Concatenate each histogram into a 1-dimensional array.

This way, each song will be represented by an array with its size depending on the number of coefficients and the amount of steps we take.

**2.3 Dimensionality reduction – PCA**  
Once we have both representations of the feature vector, we will try one last modification of it.

This will be done by applying PCA to the matrix we have, which will reduce the size of it even more.

The objective of this procedure is to have a feature vector smaller than the size of the dataset, which we expect it will help classification.

PCA

This will be done using sklearn python library, but will be explained later on.

**3 Evaluation design**

**3.1 Dataset**  
For the realization of the project, we needed a large set of songs and genres to be able to train our algorithm in a proper way.

Initially, we wanted to use a relatively small amount of songs (100) of 4 different genres, all of them royalty free, taken from Free Music Archive. The problem was that the set we ended up with was too small to make the program work as intended.

We decided to change the set to an already made one, so we looked for data sets build for our purpose and ended up finding Marysas, a website in which we could find 1000 songs of 10 different genres (100 songs per genre), all of them 30 seconds long and with a similar set of properties (which will be explained later).

All the music in the data set is available for everyone and it can be used for investigation without any charge.

All songs are “.au” files, which is a format used by the program Audacity.

To work with them, we need to know a few basics of digital audio, so I will explain what each one of the terms we will need when we extract the features of each song.

* Audio frame: Contains information in a given time.
* Sample rate: Number of samples taken from a continuous signal in order to produce a discrete signal.
* Channels: Number of streams in which the audio is sent.
* Frame size: Size of each frame. Sample rate \* # of channels.
* Frame rate: Number of frames per second. Frame size / s.

In our data set, all songs have the following properties:

* Sample rate: 22050Hz
* Channels: 1 (Mono)
* Frame rate: 22050 fps

To make the program able to work with other formats and songs, we will take all this information when we extract the features.

This is accomplished forcing the load function from librosa to take the Sample Rate as 22050 and converting the signal to Mono-channel.

3.Evaluation design

**3.2 Evaluation protocol**

Now that we have our dataset, we will explain how we are going to divide it in order to train our program. We will only use train and test sets, because we think adding a validation set will be useless in such a small dataset.

Considering this, our train and test set will follow 10-fold Crossvalidation, which will divide the original dataset in two smaller sets: the train set will have 90% of the songs; the test set, will have the remaining 10%.

Although this method is supposed to create these sets at random, we will always use the same sets, to be able to compare results between different methods and find which one is the best. Once we find which one works best, we will try it with other sets, to find a more fitting value.

To test the results, […]

**3.3 Methods and parameters**

Some of the steps we mentioned before are quite difficult to program so, in order to focus in the main experiment, we will use two already existing python libraries: librosa and sklearn.

Librosa gives us the majority of audio analysis tasks already built in, so we only need to tweak the parameters we need to get the information we need out of every song.

Sklearn, on the other side, will be used for all the algorithms involving machine learning.

**4. Results**

**4.1 Classification results**

**4.2 Feature relevance analysis**

<https://newonlinecourses.science.psu.edu/stat505/node/51/>

<ftp://statgen.ncsu.edu/pub/thorne/molevoclass/AtchleyOct19.pdf>

<https://dsp.stackexchange.com/questions/28898/mfcc-significance-of-number-of-features?utm_medium=organic&utm_source=google_rich_qa&utm_campaign=google_rich_qa>

<http://mirlab.org/jang/books/audioSignalProcessing/speechFeatureMfcc.asp?title=12-2%20MFCC>

<https://www.sfu.ca/sonic-studio/handbook/Mel.html>

<http://aircconline.com/sipij/V4N4/4413sipij08.pdf>

**To be used later:**

The functions we will use for our project are the following:

* librosa.load(): This function loads the audio file, modifying the properties of the file we need to have all files following the same standards.  
  The most important parameters we need are:
  + sr: changes the sample rate
  + mono: converts the file to mono-channel
  + duration: crops the song into a smaller length.  
    The size of the matrix depends on the length of the file, so we need to make all songs last the same to work with them.
* librosa.feature.mfcc(): calculates the MFCC of the audio file we have loaded.  
  The function automatically tweaks all the parameters it needs to make a small enough matrix, but without losing huge amounts of information.  
  In this case, each interval is about 0.02 seconds long.