UNIVERSITAT DE BARCELONA

Treball final de grau

GRAU DE MATEMÀTIQUES

Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

Supervised and unsupervised learning and clustering of audio tracks

---

Autor: Ángel Bergantiños Yeste

Director: **Dr. Sergio Escalera Guerrero**

Realitzat a: **Departament of Applied Mathematics and Analysis**

Barcelona, **18 de juny de 2018**

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.

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 classificar cançons pel seu gènere 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.

Contents

Chapter 1

Introduction

Small explanation of the project

1.1 History of 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 of audio classification

First methods used to classify audio

1.3 State of the art of audio classification

Methods that are being used now.

1.4 Sumary of the proposal

Chapter 2

Method

In this chapter, we will explain the methods we are going to use later to try to classify our dataset, which will be presented later.

The songs we get can't be used as raw data, thus we need to treat the tracks to be able to work with them. This will be accomplished using MFCC, to extract audio features; PCA, to reduce the dimensionality of the matrix given by MFCC; and Decision Trees, to get the relevance of each feature and know which ones is more useful.

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:

4. **Take power logs of each mel frequency.**

5. **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:

$$C_n = \sum_{k-1}^{k} (logD_k) \cos\left[m(k - \frac{1}{2})\frac{\pi}{k}\right]$$

   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 Naive

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 Component 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 (Principal Component Analysis) 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 works by orthogonally transforming a set which may have correlation into a new one linearly uncorrelated. This procedure will be done following these steps:

1. **Standardize.**
   First, we want to have all our data to be standardized, in order to make the following step easier to calculate.

2. **Calculate the covariance matrix.**
   Now, we have to create a matrix which will be composed of the covariance of each one of the features, following this diagram:

$$\begin{bmatrix} cov(x_1, x_1) & \dots & cov(x_1, x_n) \\ \vdots & \ddots & \vdots \\ cov(x_n, x_1) & \dots & cov(x_n, x_n) \end{bmatrix}$$

3. **Find the eigenvectors and eigenvalues of the matrix.** [Explanation]

4. **Re-arrange data.**
   Once we have the new matrix, we multiply the original by the eigenvectors, which will re-orient the data, having the original matrix converted to a less dimensional one.

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

2.4 Feature relevance

Even after reducing the dimensionality of our feature vectors, we can end up having information that doesn't give us meaningful information, so we will want to focus in the features that will help our program to give the best results, which will be measured using the accuracy of the predictions, as we will explain later.

In order to detect the most important MFCC we want to get, we will use algorithms based in decision trees, more precisely, Extra Trees.

Extra Trees algorithm (Extremely Randomized Trees) works similar to Random Trees but instead of choosing the best split from a random subset of the training set, they are chosen at random from the random subset for each tree. Apart from that difference, both help reducing variance, in expense of higher bias.

Given the number of features and how different two samples of the same genre can be, we don't really mind the increase in the bias if we can get, in return, a smaller variance, which can help training our program.

The algorithm works as follows: [Explanation]

Chapter 3

Evaluation design

Here, all the experiments we have done will be explained in detail, from how we get the features to the experiments we realize with each one of them.

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).

The genres in the dataset are some of the most common genres in music, which are as follow:

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

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.

3.2 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, once we have trained our model, we will check if each one of the samples in the test set can be predicted correctly. With that, we will create a confusion matrix that will help us identify what genres are often mixed up.

The number of songs correctly classified will tell us how good our program is working.

3.3 Mehtods 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.

From this library, we will only use two methods:

- **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.
- **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.
  From all the parameters that can be tuned, we only care about n_mfcc, which is the amount of MFCCs the algorithm will return. All the order parameters modify the properties of the song but, as we did already tune them with the "load" method, we don't need to do it now.

Knowing this, the experiments that we will carry out will be determined by the number of MFCC we calculate.

By default, MFCC returns 20 features for each interval, but this can lead to having data that won't give relevant information, as well as take a lot of time to compute in a laptop. For this reasons, we will use the following values for our experiments: 5, 10, 15 and 20 (in case it works best and we decide to keep using this amount). We wanted to use values in between, but the time it will take to perform the test versus the improvement we could get makes it purposeless.

All the experiments will be done using these 4 values but, if we find that one of them has far better results than the others, we will stick to that value, to make experiments faster.

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

The functions we will use from sklearn library are the following:

- **PCA:** this method will help us apply the method we explained before, to reduce the dimension of the matrix of values we have.
  The main parameter we are going to tune will be as follow:
  - n_components: number of components there will be after we apply PCA to our set.
- **SVC:** (Support Vector Classifier) this is the algorithm we are going use in most of the experiments to create a model which we can use to classify our dataset. It is part of the SVM module of sklearn and is the one we think will give the best results, considering our problem.
  To make it work properly with our dataset, we will have to tune the following parameters:
  - **kernel:** the kernel type we are going to use. We will perform an experiment with all the kernels sklearn offers us: rbf, linear, poly, and sigmoid, but will be explained later.
  - **C:** the penalty parameter. This will be the most important when using the 'linear' kernel, as gamma has not effect in it.
  - **gamma:** the kernel coefficient. It will allow us to tune the variance of the classifier.
- **fit:** fits the model
- **predict:** given a model and a sample, predicts its value.

- **ExtraTreesClassifier:** this class will help us classify the features by its relevance, using the method we explained before.

There are others methods from the library that we will use, but most of them are implementations of algorithms we will use in our experiments, so we are going to only show them:

- **GaussianNB:** implements Naïve-Bayes
- **AdaBoostClassifier:** implements AdaBoost
- **cross_val_score:** will be used to check the accuracy of the AdaBoost Classifier.
- **LeaveOneOut:** implements Leave One Out.

Chapter 4

Results

Once we have all the experiments set up, we can already start them.

In this chapter, we will show the results of each experiment we conduct, in which we will have a value based on the accuracy, which will be given by a percentage, as well as a confusion matrix that will show us which genres are most commonly confused.

The order of the experiments will be in the same order as the one we used to explain them, ignoring the results we get.

4.1 Classification results

The first experiment we will perform will be done using the naïve representation of the data.

For that, we will load all songs from our dataset, calculate the MFCC and store them in a numpy array, where each component will be a tuple containing the matrix that we have converted into a 1-dimensional array in the first component, and the genre in the second, which will be an unsigned integer going from 0 to 9, following this:

0. Blues
1. Classical
2. Country
3. Disco
4. Hip hop
5. Jazz
6. Metal
7. Pop
8. Reggae
9. Rock

The extraction of the data will be done 4 times, to have 5, 10, 15 and 20 features, using the following code:

[CAPTURA DEL CODIGO char1()]

Now that we have our dataset ready to work with, we have to create the train and test sets that we will use for the experiment.

sklearn needs two different train and test sets in order to work properly: one of them, will have the arrays of features, while the other will have, for each position of the first array, its genre.

As we said before, we are using 10-fold Crossvalidation, but we want all experiments to give us results we can compare. For this reason, we will divide the original dataset using the following division:

[CAPTURA DE LA FUNCION TRAIN_TEST]

Once we have our train and test sets, we can start classifying.

First, we will show a matrix with the best results we get from each experiment. Then, we will show each one of them and how we ended up having them.

n_mfcc	**rbf**	**Linear**	**Poly**	**Sigmoid**
5	15	38	38	10
10	15	44	41	10
15	22	47	39	11
20	24	46	41	11

As we can see, Linear gives us the best results with the default parameters,

with both 15 and 20 features giving the highest accuracy, so these are the ones that we will use for the following experiments using this dataset representation.

The best accuracy comes from having 15 features and using linear kernel. The following confusion matrix shows us better detail of it:

5	1	1	0	0	0	2	0	0	1
0	10	0	0	0	0	0	0	0	0
2	0	4	0	1	1	0	1	0	1
1	0	1	3	1	0	0	0	0	4
0	0	2	2	0	0	2	2	1	1
1	3	1	0	0	5	0	0	0	0
1	0	0	0	0	1	8	0	0	0
0	0	0	1	0	0	0	9	0	0
2	1	0	1	1	2	0	0	2	1
4	0	1	1	1	0	1	0	1	1

As expected, classical music is the genre that gives us best accuracy, given its difference between more modern genres, and it's also one of the genres other songs aren't usually confused to.

Pop also gives us great results, but genres such as rock and blues are the ones that most songs are predicted into, given that both are genres other genres evolved from.

Now, we will try tweaking gamma for rbf, poly and sigmoid methods and C for linear. The values will be taken by trial and error, trying different values until we find the best for each method and number of features. This will be done by creating different values, first, by powers of ten, between 10^{-10}) to 10^{5}, once we find the best, we should calculate the accuracy in a range of values around it until the accuracy remains constant. Given that we are doing all the experiments in a laptop, the time it will take to do it would be too long, so we will only get the value from the first iteration.

n_mfcc	**rbf** $\gamma = 10^{-7}$	**Linear** $\gamma = 10^{-7}$	**Poly** $\gamma = 10^{-7}$	**Sigmoid** $\gamma = 10^{-7}$
15	50	47	42	34

n_mfcc	**rbf** $\gamma = 10^{-7}$	**Linear** $\gamma = 10^{-7}$	**Poly** $\gamma = 10^{-7}$	**Sigmoid** $\gamma = 10^{-7}$
20	47	46	44	34

As we can see, linear gave us the same results before and after tweaking its C, which made that, although its accuracy was higher at the beginning, one we start changing the gamma value, rbf gives us better results.

Now that we have found the best result using naïve representation, we will try to reduce its dimensionality using PCA, to find if we can improve it.

Given that, in all cases, poly and sigmoid kernels give us worst results, from now on we will only use rbf and Linear kernels, which will allow us to test more experiments with more values.

A we explained before, this will be done using the function given by sklearn library, which reduces the size of each song's vector to the amount of features we want. Once we have reduced the size of each vector, we can apply the same method as before, which gives us the following results:

n_mfcc	**rbf**	**Linear**
5	12	34
10	12	42
15	12	44
20	12	41

[FALTAN RESULTADOS]

Once we have tried with all the data, we are going to try the same experiments, but using the histogram representation, as we explained before.

The code of the features extractions will work similarly to the experiment before, but taking both, the minimum and maximum value of each MFCC, as we can see here:

[CAPTURA DE CHAR3()]

Once we have all the min and max values of each song, we take the minimum and maximum value of all songs, which will be used to calculate the variable "step", which will be the size of each interval of the histogram, dividing the range of all values by the amount of intervals we want.

The histograms we will end up having will be similar to these:

[GRAFICAS DE HISTOGRAMA]

As we explained before, we will only use rbf and linear kernel, dividing the results in two different tables, each one for a different kernel, and using intervals in tens.

Using rbf kernel with default parameters, we get these results:

n_mfcc	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
5	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0
15	24	37	37	35	37	33	31	31	29
20	0	0	0	0	0	0	0	0	0

Using linear, we get these:

n_mfcc	**10**	**20**	**30**	**40**	**50**	**60**	**70**	**80**	**90**
5	25	41	41	45	41	0	0	0	0
10	25	42	44	50	50	0	0	0	0
15	25	42	44	53	51	50	54	57	59
20	25	42	44	53	52	51	54	55	57

Once we have tried the default values, we can start tweaking rbf's gamma and linear's C. In this case, we will only tweak the value with the amount of intervals that gave us the best results, because it would take too long to tweak those parameters for each one of the possibilities:

[FALTAN RESULTADOS]

With these experiments, we have found that the best accuracy is given using the histogram representation, without PCA, and with around 90 intervals in each histogram. For classification, rbf kernel using a gamma = 1.4, we have the best possible accuracy of all the methods we have tried.

To confirm this result, we will try using other train and test sets, to see if it is consistent.

Now, having this in mind, we will try other methods to get other approximations to the problem.

[FALTAN RESULTADOS]

4.2 Feature relevance analysis

Chapter 5

Conclusions

Bibliography

[1] M.F. Atiyah, *Vector bundles over an elliptic curve*, Proc. London Math. Soc., **7**, no. 3, (1957), 414–452.

[2] R. Buchweitz, G. Greuel and F.O. Schreyer, *Cohen-Macaulay modules on hypersurface singularities, II*, Invent. Math. , **88**, no. 1, (1987), 165–182.

[3] M. Demazure, *A very simple proof of Bott's theorem*, Inventiones Math. **33** (1976) 271-272.

[4] D. Eisenbud and J. Herzog, *The classification of homogeneous Cohen-Macaulay rings of finite representation type*, Math. Ann. **280** (1988), 347–352.

[5] D. Eisenbud, F.O. Schreyer and J. Weyman, *Resultants and Chow forms via exterior syzygies*, J. of Amer. Math. Soc., **16** (2003), 537–579.

[6] D. Faenzi, F. Malaspina, *The CM representation type of homogeneous spaces*, Preprint 2014.