*Projet Python*

Réalisation d’un outil de reconnaissance musicale

\*\*\*\*\*

**Solution GALY**

M2 SISE 2019-2020

**G**uillaume Martial

**A**urélien Dautois

**L**ong Nguyen Phuoc

**Y**asmine Medkour



**INTRODUCTION :**

Dans le cadre du cours de machine learning dans notre Master 2 SISE et afin de compléter nos connaissances et notre pratique du langage python, nous avons réalisé un projet de reconnaissance musicale. L’objectif est de mettre en place un outil permettant de retrouver un fichier audio sur la base d’un extrait dans une base de fichiers audios existantes.

Nous avions à notre disposition une bibliothèque de 62 fichiers sons, et le but final est donc de retrouver le son qui correspond à un extrait enregistré via un microphone. Les étapes intermédiaires ont été de reconnaître un son complet puis un extrait d’un son sans passer par un microphone et finalement de passer par un microphone.

Notre outil doit renvoyer à minima le nom, l’artiste, l’album et la date de sortie de la chanson.

Ce rapport présente donc le résultat de notre travail :

* [Une définition du son et du traitement d’un signal](#_Son_et_traitement)
* [L’environnement de travail et les librairies utilisées](#_Environnement_Colab-Librairie_libro)
* [Présentation du code](#_Présentation_du_code)
* [Exemple avec un son enregistré](#_Exemple_avec_un)
* [Problèmes rencontrés, solutions et pistes d’améliorations.](#_Problèmes_rencontrés,_solutions_1)

# Son et traitement du signal

Dans un premier temps, il est essentiel de définir ce qu’est un son. Le son est une onde produite par une vibration. L’amplitude est une caractéristique importante du son. L’intensité perçue dépend de l’amplitude : le son peut être fort ou doux. Dans l’air, l’amplitude correspond aux variations de pression d’onde.

Pour le traitement numérique du son : traitement par un ordinateur, il faut procéder par une conversion. Cette opération consiste à transformer les variations de pression du son en une suite de nombres.

Un microphone convertit les variations de pressions de l’air en signaux électrique qui, reliées à un convertisseur va le transformer en une suite de nombres. Ce travail est réalisé par les cartes son des ordinateurs.

Le traitement du signal peut se faire de différente manière, nous avons utilisé la transformée de Fourier ainsi que les spectrogrammes pour cela.

La transformée de Fourier permet de décrire un signal continu dans l’espace des fréquences. Elle va ensuite nous permettre d’obtenir le Spectre d’un signal sonore.

Un spectre de son représente l’intensité acoustique en fonction de la fréquence ou de la longueur d’onde d’un son. On parle de spectre de bande quand la répartition en fonction de la fréquence est continue. Le Spectrogramme permet d’analyser un signal sur des fenêtres successives de temps et de faire une représentation en temps/ fréquence.

# Environnement Colab-Librairie librosa

Google Colaboratory est un outil pluggé sur google drive qui nous a permis de programmer notre outil de manière collaborative sur le drive, à l’image d’un google docs par exemple.

Au départ, nous n’avions pas choisi de travailler sur le cloud, mais nous avons fait ce choix car il nous paraissait avantageux sur deux points :

Dans un premier temps, nous n’avons pas réussi à faire fonctionner la librairie Librosa sur nos ordinateurs personnels, sur une version locale de python. En passant par Google Colaboratory, nous n’avons rencontré aucun problème à ce niveau-là, et tous les packages sont installés pour tous les utilisateurs. Dans un second temps, plutôt que de démultiplier les codes, cette solution nous permettait de centraliser notre travail, ce qui nous aidait dans un souci de partage des fichiers en temps réel.

Finalement, il semble aussi plus simple de partager un outil fonctionnant entièrement, et de manière indépendante, sur le cloud plutôt que de partager des fichiers python qui dépendent des packages de chaque utilisateur.

Pour mener à bien ce projet, nous avons utilisé le package LIBROSA qui permet de faire de l'analyse de signaux sonores. Il fournit les éléments de base pour récupérer des informations musicales.

D’autre part, nous avons utilisé les librairies classiques suivantes :

*import numpy as np*

*import pandas as pd*

*import matplotlib.pyplot as plt*

*import os*

*from google.colab import drive*

# Présentation du code

## Nous allons voir le code exploité par notre outil dans l’ordre d’apparition des fonctions dans la fonction principale nommée “recognize”. Nous aurons ainsi une image pas-à-pas du fonctionnement de notre algorithme de reconnaissance musicale.

## Fonctions liées au traitement du son de la libraire librosa

## Signal audio

Récupérer un signal audio avec la fonction load :

*x, sr = librosa.load(“chemin\_d’un\_fichier.mp3”)*

→ sr, fréquence d’échantillonnage (ici 22050 points par seconde).

→ x, signal audio : vecteur de sr \* 30s points pour un morceaux de 30s.

## Transformée de Fourier, amplitude et spectrogramme

Effectuer la transformée de Fourier avec la fonction stft et passage en amplitude avec amplitude\_to\_db :

*I = librosa.stft(i)*

→ on obtient une matrice : chaque colonne correspond à la transformée de fourier (spectre) sur plage de 2048 points du signal audio, soit 93 millisecondes.

*Idb = librosa.amplitude\_to\_db(abs(I))Colaboratory*

→ matrice de la même taille : “amplitude” de chaque spectre en décibel

## Création d’une empreinte

## Max de l’amplitude

La fonction amplitude récupère l’amplitude maximum des spectres en prenant en entrée un signal audio x et en renvoyant un vecteur (de 1200 points environ selon la fréquence d'échantillonnage et le nombre de points de la transformée) du spectre créé par la transformée de Fourier.

*def amplitude(i):*

*I = librosa.stft(i)*

*Idb = librosa.amplitude\_to\_db(abs(I))* # "Module" de la transformée

*max\_ampl = []*

*for i in range(Idb.shape[1]):*

*max\_ampl.append(Idb[:,i].max())*

*return(max\_ampl)*

## Lissage

Afin de réduire au maximum les bruits extérieurs sur nos sons enregistrés, nous sommes passés par un lissage de nos courbes.

*def lissage(Ly,p):*

*'''Fonction qui débruite une courbe par une moyenne glissante sur 2P+1 points'''*

*Lyout=[]*

*for index in range(p, len(Ly)-p):*

*average = sum(Ly[index-p : index+p+1]) / (2\*p + 1)*

*Lyout.append(average)*

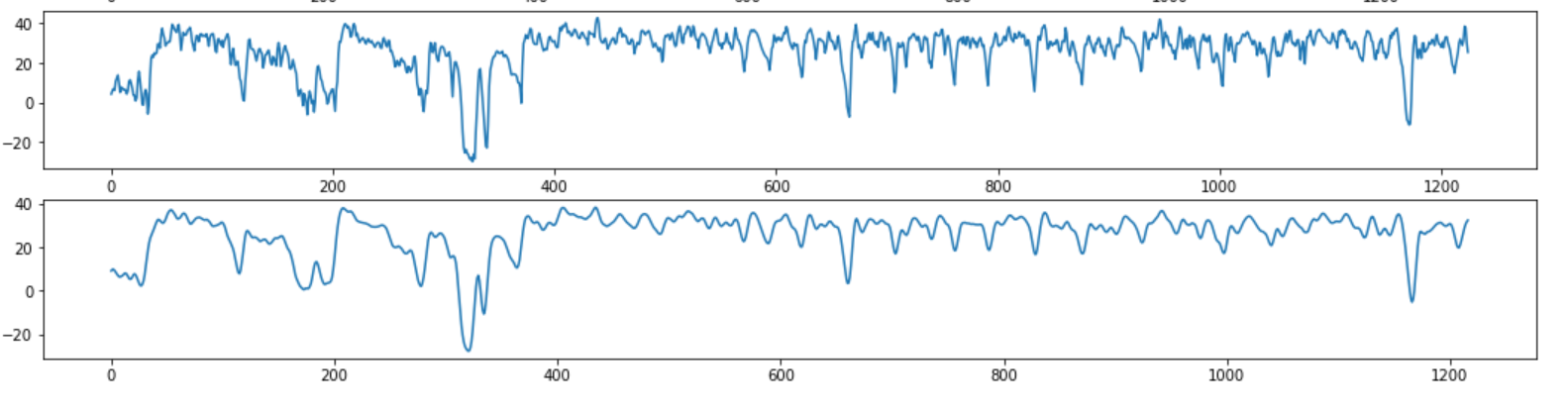
*return Lyout*

→ empreinte = max amplitude + double lissage

*x, sr = librosa.load(file\_path)*

*x\_empreinte = amplitude(x)*

*x\_empreinte\_lisse = lissage(lissage(x\_empreinte, 2), 2)*

La première courbe ci-dessous est non lissée tandis que la seconde est lissée.

Nous pouvons constater qu’il y a moins de fluctuations sur la seconde courbe. Cela permettra de mieux reconnaître les sons enregistrés avec un micro et ayant potentiellement du bruit.

## Création de la base d’empreintes

Ensuite, nous appliquons ces fonctions sur les 62 chansons de notre bibliothèque afin de créer une base de vecteurs d’empreintes : des valeurs maximales par unité de temps.

Une fois la chanson transformée en un vecteur d’empreinte, nous l’avons lié avec ses informations comme le titre et le nom de l’auteur afin que notre outil donne ces informations lors de la reconnaissance. Pour cela, nous avions un fichier avec les métadonnées de toutes les chansons de notre base. Puisque notre base s’est restreinte au dossier 000, nous avons créé une nouvelle base de données issue du fichier des métadonnées comprenant 62 lignes (une par chanson) avec les colonnes suivantes :

*track\_id; album\_title; artist\_name; track\_title; track\_date\_created; track\_date\_recorded; track\_duration; track\_genres\_genre\_title; track\_language\_code; track\_listens*.

*meta = pd.read\_csv("metadata\_V2.csv", sep = ";", header=0)* #Importer les métadonnées

*dir\_path = '.../Music files/000/'*

*dico\_empreinte = {}* #Initialiser un dictionnaire vide

#Faire un boucle qui traite chaque chanson

*for element in os.listdir(dir\_path):*

*file\_path = dir\_path + element*

*x, sr = librosa.load(file\_path)*

#Extraire les éléments des métadonnées pour créer les clés

*num = int(element[-7:-4])*

*cle\_df = meta.loc[meta['track\_id']==num, ]*

*cle = ""*

*for i in range(0 , cle\_df.shape[1]) :* #Une boucle pour créer des clés du dictionnaire

*cle = cle + " - " + cle\_df.columns[i] + " : " + str(cle\_df.iloc[0,i])*

#Affecter le vecteur des valeurs maximales au dictionnaire

*dico\_empreinte[cle] = energie(x)*

## Reconnaissance audio

Nous allons maintenant voir le cœur de notre algorithme de reconnaissance musicale, avec la comparaison de deux empreintes, la comparaison d’une empreinte à la base et finalement les sorties.

## Comparaison de deux empreintes

Nous avons utilisé un coefficient de corrélation pour comparer deux empreintes et savoir s’il s’agit de la même musique.

*def correl(x, y):*

*if len(x) <= len(y) :*

*extrait = x*

*ref = y*

*else :*

*extrait = y*

*ref = x*

#L’empreinte la plus courte est considérée comme étant l’extrait et l’autre empreinte devient la musique de référence.

*borne\_inf = 0*

*max\_coef = -1*

*while borne\_inf <= (len(ref) - len(extrait)):*

*ref\_court = ref[borne\_inf : (borne\_inf + len(extrait))]*

*max\_coef = max(max\_coef, np.corrcoef(extrait, ref\_court)[0,1])*

*if max\_coef == np.corrcoef(extrait, ref\_court)[0,1]:*

*instant = borne\_inf*

*borne\_inf = borne\_inf + 1*

*return(max\_coef, instant)*

On utilise le coefficient de corrélation de numpy pour calculer la similarité entre notre extrait et notre référence.

Le coefficient de corrélation permet de mettre en évidence une liaison entre deux séries quantitatives de données. Il est compris entre -1 et 1, et la corrélation est forte lorsqu’elle est proche de ces deux valeurs et faible lorsqu’elle est proche de 1.

Tant qu’on peut parcourir la référence, on calcule toutes les similarités avec un pas de 1 dans la référence (une colonne parmi celles créées par la transformée de Fourier).

Nous avons aussi une variable instant qui nous donne le moment dans la référence où l’extrait est le plus corrélé avec la référence (par exemple pour un instant de 4,5 secondes, notre extrait commence à 4,5 secondes de la référence).

## Comparaison avec la base d’empreintes

*# boucle de comparaison entre l'empreinte du signal d'entrée et la bibliothèque de sons*

*max\_biblio = -1*

*for cle,valeur in dico\_empreinte.items():*

*cor, inst = correl(x\_empreinte\_lisse, valeur)*

*max\_biblio = max(max\_biblio, cor)*

*if max\_biblio == cor :*

*ref = cle*

*sec = inst / 30*

*print("Similitude : ", max\_biblio)*

*if max\_biblio > 0.6 :*

*print("L'extrait écouté correspond au titre : ")*

*print(ref)*

*print("À l'instant (s) : ", sec)*

*elif max\_biblio > 0.4 :*

*print("Le morceau le plus proche :")*

*print(ref)*

*else :*

*print("Aucune correspondance n'a été établie.")*

Ici, nous allons appliquer la fonction de comparaison de deux empreintes à l’ensemble des empreintes présentes dans la base et retourner le coefficient de corrélation le plus élevé retrouvé entre deux empreintes ainsi que toutes les informations stockées dans le dictionnaire de notre base.

Nous avons choisi deux seuils de valeur pour la corrélation : si le coefficient le plus élevé trouvé est de 0.6 ou plus, le morceau recherché est trouvé, s’il est compris entre 0.4 et 0.6, nous avons un morceau proche, et s’il est inférieur à 0.4, nous n’avons pas trouvé de correspondance.

## Enregistrement du son à partir d’un microphone

Pour cette partie de notre outil, nous avons rencontré plusieurs problèmes liés à notre méthode de travail. En effet, travaillant sur Google Colab, nos commandes pythons s’effectuent sur une machine virtuelle et pas sur nos ordinateurs. Nous avons donc une fonction python pour enregistrer du son depuis un microphone qui marche en local sur nos machines mais qui ne fonctionne pas sur les Google Colab car nos micros ne sont pas sur la machine python. Nous avons ensuite essayé d’utiliser un code mélangeant du javascript et du python pour faire en sorte que nos navigateurs internet accèdent aux micros des ordinateurs, et ce code marche.

Pour autant, il ne nous a pas semblé possible de régler la fréquence d’enregistrement et les objets retournés par ce script ne nous permettait pas de travailler avec librosa, nous avons donc abandonné cette idée.

Finalement, nous avons choisi de passer par un plug-in de google drive qui permet d’enregistrer des sons et de les stocker directement dans le drive. Cette solution facilite l’utilisation du drive puisque tout se fait directement depuis ce dernier.

## **Exemple avec un extrait enregistré**

Tout d’abord un test sur le morceau 000255 avec un extrait du morceau directement depuis le fichier .mp3 et non depuis le micro :

*path\_test =* *'/content/drive/My Drive/Projet final - Python/Music files/000/000255.mp3'*

*recognize(path\_test)*

Similitude : **0.9999145833206274**

L'extrait écouté correspond au titre : - track\_id : 255 - artist\_name : Bird Names - album\_title : On Opaque Things - track\_title : We Want to be Old - track\_date\_created : 11/26/2008

L'extrait correspond à l'instant (s) : **1.4333333333333333**

## Ensuite un test avec un extrait du morceau enregistré via le micro :

*path\_test = '*/content/drive/My Drive/Projet final - Python/Livrables/Sons\_micro/255.mp3'

*recognize\_freq(path\_test)*

Similitude : **0.7400981384195056**

L'extrait écouté correspond au titre : - track\_id : 255 - artist\_name : Bird Names - album\_title : On Opaque Things - track\_title : We Want to be Old - track\_date\_created : 11/26/2008

L'extrait correspond à l'instant (s) : **2.1666666666666665**

## **Problèmes rencontrés, solutions et pistes d’améliorations**

Comme dit plus tôt dans ce rapport, Google Colab nous a permis d’éviter plusieurs problèmes en nous en posant un autre en particulier : nous avons pu travailler librement avec librosa, les fichiers .wav et .mp3 ainsi que de façon collaborative grâce à cet outil, mais il nous a empêché d’utiliser le script que nous avions écrit pour enregistrer notre micro.

Une piste d’amélioration importante serait donc de réussir à utiliser un micro directement dans notre programme.

A l’origine, nous devions exploiter une base de 2050 sons pour notre reconnaissance musicale et ce grand nombre de son aurait pu poser problème. En effet, la création de la base d’empreinte sur 62 sons dure déjà entre une ou deux minutes, la bibliothèque entière prendrait donc en théorie plus d’une heure. De plus, il faudrait stocker ces informations pour ne jamais avoir à recréer la base.

Finalement, notre algorithme est exhaustif au niveau de la comparaison d’empreinte, c’est-à-dire qu’il parcourt tous les sons de la base avant de donner un résultat. On peut imaginer qu’avec une très grande bibliothèque, le temps de traitement serait long, il faudrait donc peut-être modifier la méthode de comparaison, ou l’optimiser pour qu’elle ne soit pas exhaustive.

Pour conclure sur ce projet, nous avons fait un outil de reconnaissance musical porté sur 62 chansons et nous arrivons à retrouver un extrait d’une des chansons.