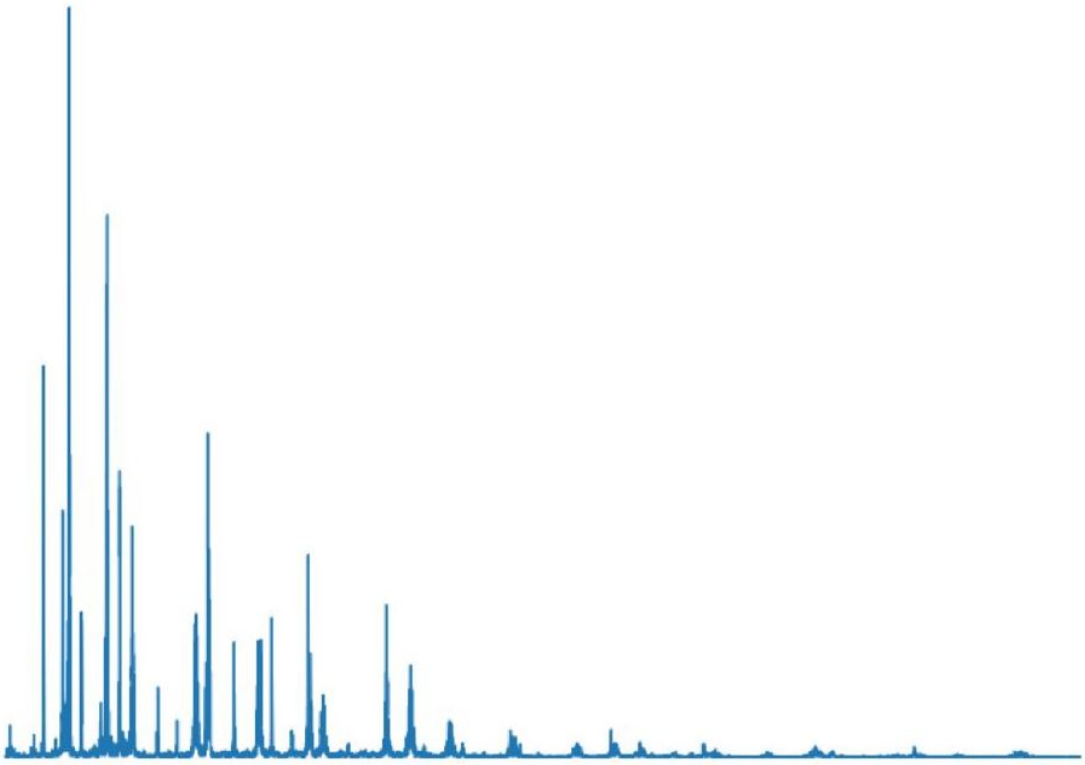
université
PARIS-SACLAY

FACULTÉ
DES SCIENCES
D'ORSAY

Musical Instruments Classifier

Clément GILLI & Louis-Alexis PENELOUX

Avril 2023



1 Résumé

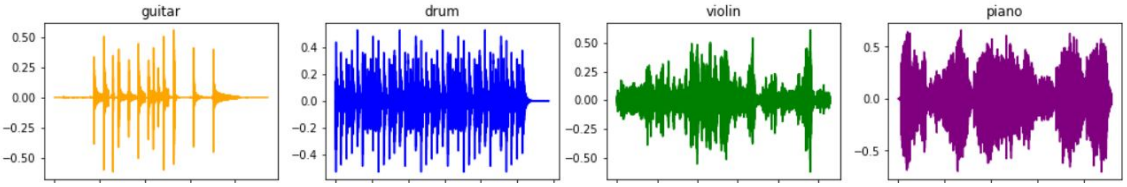
On cherche ici à construire un programme de classification de fichiers sonores en fonction de l'instrument de musique utilisé. On se limite ici uniquement à des violons, guitares (électrique ou acoustique), piano et percussion. Chaque fichier peut être traité grâce à la bibliothèque python *librosa*. Les features utilisées sont basés sur le calcule du spectre de fréquence, rendu possible grâce au module fft de numpy. Les différents aspects techniques d'apprentissage statistique utilisés sont le pré-processing, l'extraction de features, la réduction en composantes principales, la cross-validation, et l'utilisation d'un classifier de type SVM en faisant varier les hypers-paramètres. On revient dans ce rapport sur chacun de ces points, en expliquant les choix qu'on a dû faire en fonction des difficultés rencontrées.

2 Pré-processing

Le jeu de donné utilisé est disponible ici. Noter que deux jeux de données sont disponibles ici, un *train* (1600 fichiers utilisables en pratique), un *test* (80 fichiers). Nous avons tout fait depuis le *train*, même les tests, pour des raisons pratiques. Un CSV reliant chaque fichier à sa classe ($Sound\_Guitar$, $Sound\_Violin$, ...) était fourni. La première problématique à traiter avant d'ouvrir python a été de corriger un bon nombre de fichiers mal classifiés (mauvais label sur une série de sons donnée). Comme il y en avait plusieurs centaines, il a fallu passer par un script.
Le seconde problème de pré-processing à gérer a été le chargement des fichiers. Il est relativement long pour librosa de charger chaque fichier audio en entier. Nous avons fait le choix de ne considérer que 1.5 seconde de chaque fichier, tirer au milieu si la longueur du fichier est suffisante. Pour charger les 1600 fichiers, et faire les calculs de fréquences nécessaires (on y reviendra après), comptez entre 1 ou 2 minutes.

3 Visualisation de données et extraction de features

Après avoir chargé un fichier audio avec librosa, on peut très vite obtenir l'intensité du signal au cours du temps.



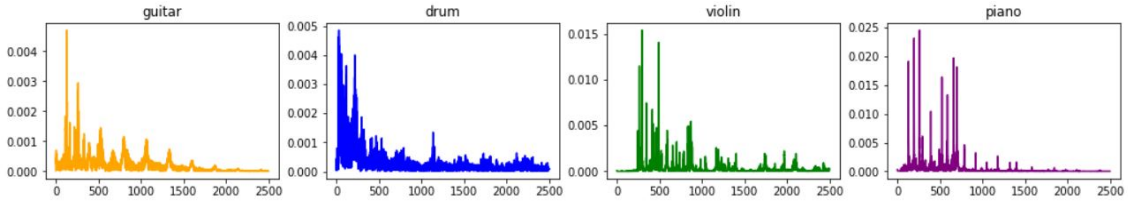
Ce n'est pas très intéressant, car d'un côté on ne charge que 1.5 s de chaque fichier audio, d'un autre, l'intensité n'a rien à voir avec le timbre d'un instrument. On va plutôt s'intéresser à l'analyse du spectre de fréquences.

La théorie de l'analyse de Fourier affirme que le signal sonore d'un son peut se décomposer en somme de signaux de fréquences multiples.

Théorème (Dirichlet) :
Soit g une fonction T-périodique continue presque partout. Alors elle admet un développement en série de Fourier, c'est à dire que pour la fréquence fondamentale $f = \frac{1}{T}$, il existe une suite d'amplitude $(A_n)_n$ et de déphasage $(\varphi_n)_n$ tel que

$$g(t) = A_0 + \sum_{n=1}^{\infty} A_n \cos(2\pi(nf) + \varphi_n) \tag{1}$$

Autrement dit, un signal est défini par une fréquence de départ f, et de différentes intensités sonores correspondant aux fréquences multiples de f. Certaine fonctions du modules fft de numpy nous permette de calculer directement cette décomposition du spectre des fréquences. Voir la 1ère figure de la couverture de ce rapport qui illustre bien ce principe sur un son de violon. En testant sur des fichiers de natures différents, on obtient :



En fait, on sait que le timbre d'un instrument se caractérise, indépendamment de la fréquence de la note fondamentale, par les différentes intensités de vibration sur les fréquences multiples. C'est la piste que nous avons exploré durant tout ce projet. Noter que nous n'avons pas utilisé de fonction d'extraction de features fournis par librosa.

4 Gestion des données

Toutes les fonctions sont déposés dans le fichier *utilities.py*.
Une première fonction charge les données, calcule le tableau de spectre de fréquence, et renvoi un dataframe. Après tâtonnement et pour des questions de performance, on a choisi de prendre le tapissage de fréquences suivant : [0 Hz, 5 Hz, 10 Hz, ...], qui nous donne un certain nombre d'information sans être trop lourd à stocker en mémoire.
Attention, le nom de la fonction est ambigu : elle s'appelle *load_train* car ce sont les fichiers provenant du dossier *Train_submission* que nous avons utilisé. Cette fonction sert à la fois à charger les fichiers du train **et** du test.

```
def load_train(nb_per_class,duration=2,maxfreq=5000):
    """
    Charge les donnees.
    Parametres :
        - nb_per_class : nombres de fichiers audios utilises pour chaque
    classe d'instrument
        - duration : duree pris en compte pour un fichier audio
        - maxfreq : nombre de tranche de frequences utilisees pour la
    decomposition de Fourier
    Return:
        DataFrame dont les lignes sont les fichiers audios, les colonnes
    le label et l'intensite pour chaque frequence
    """
```

utilities.py

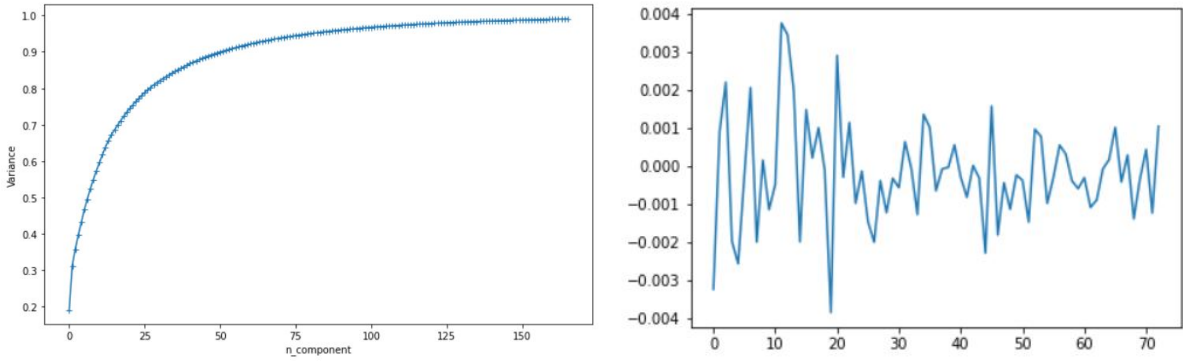
Ce n'est que par la suite qu'on sépare nos données en deux jeux : l'un pour l'entraînement (cross validation comprise), l'autre pour les tests. La séparation des données est faite aléatoirement, d'où la variabilité mineur, mais bien présente, des résultats d'une exécution du code à une autre.

```
def load_subsets(df,coef_train=0.6, coef_test=0.4):
    """
    Fait la separation des donn es entre un jeu pour les entrainements
        (cross-validation comprise) et tests.
    Return : X_train, y_train, X_test, y_test
    """
```

utilities.py

5 PCA

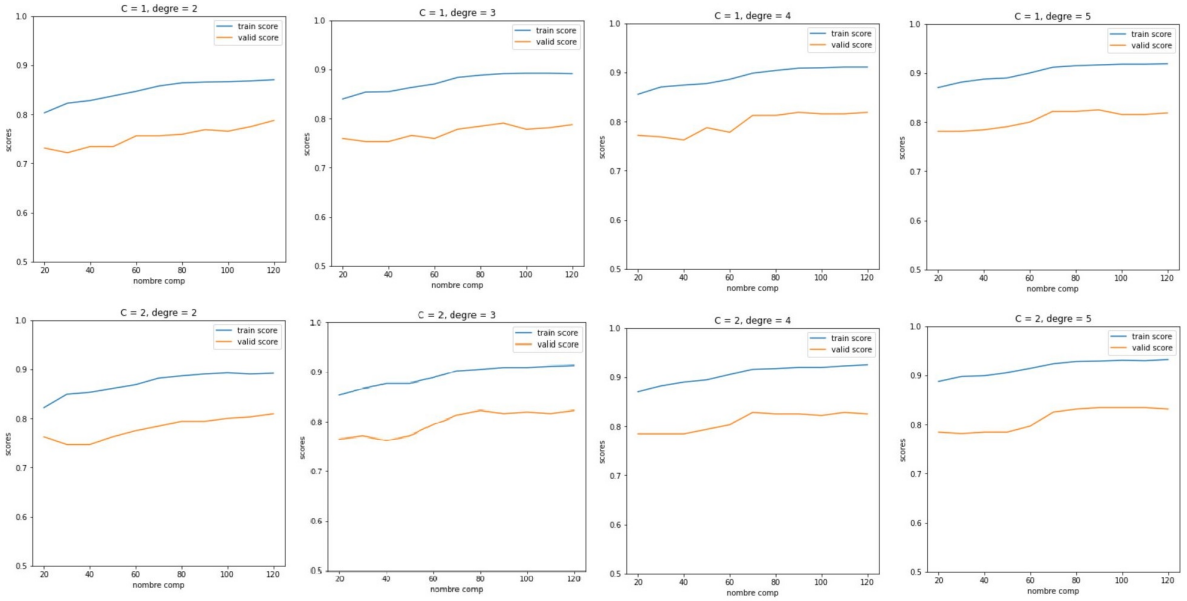
La difficulté intrinsèque à nos features est principalement dû à la grande variabilité des données. En effet, en général, on a un grand tableau (5000 cases environ) contenant beaucoup de valeurs très faibles, et quelques fois des valeurs plus hautes. On va choisir de conserver une variance importante, en conservant au moins 98% de la variance. Pour les 1600 données : à gauche l'évolution de la conservation de la variance en fonction du nombre de composants préservés, à droite l'allure d'une donnée après PCA.



En petite dimension (quand on travaille avec 40 fichiers par exemple), le nombre de features gardé est si faible que les tableaux de données sont transformés en tableaux quasiment uniformes de petits nombres. Il faut donc travailler avec un jeu de données relativement important pour garder des informations significatives sur chaque donnée.

6 Choix du classifier, premier ajustement des hypers-paramètres

Cherchant un classifier multi-class, notre premier choix d'exploration s'est naturellement tourné vers un classifier de type SVC polynomial. La répartition chaotique des données post-PCA nous amène à considérer un polynôme de degré assez haut. Les tests numériques donnent, pour une répartition 80% train, 20% validation :



On conserve le paramètre d'erreur $C = 2$ et le degré du polynôme $D = 5$. Maintenant que nos hypers-paramètres sont fixés, on va appliquer une cross-validation pour détérminer le meilleur jeu de données sur lequel baser l'apprentissage.

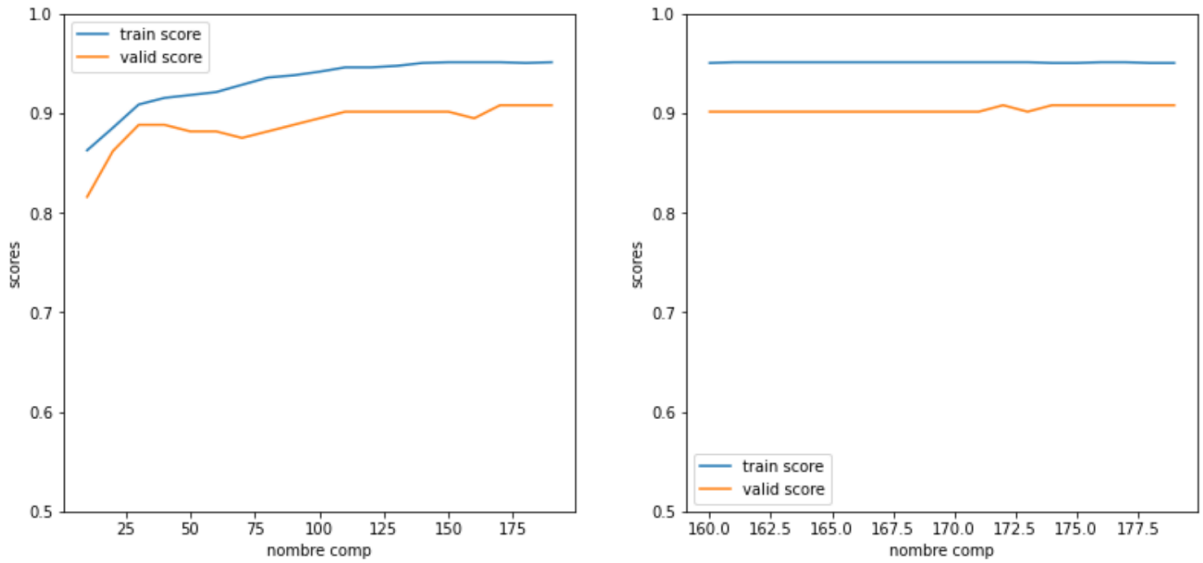
7 Cross-validation

Comme nous ne sommes que deux à travailler sur le projet, et que la problématique de performance est importante, nous avons fait le choix de ne pas implémenter la cross-validation nous-même. Les hypers paramètres sont fixés à $C = 2$ et $D = 5$ Nous avons fait une première cross-validation entre 10 et 200 composantes en allant de 10 en 10, puis nous avons raffiner cette recherche en un second temps pour trouver le classifier avec le meilleur score. On trouve finalement de meilleurs résultats en moyenne avec $D = 6$, hyper-paramètre qu'on conserve jusqu'à la fin. Tout le code est condensé dans la fonction suivante :

```
def cross_val(X,y,cv,C,degree,begin,end,step):
    """
    Realise la cross-validation.
    Parametres:
        - X, y les donnees
        - cv : le nombre de boucle pour la cross validation
        - C, degree : parametres du classifier
        - begin, end, step : debut, fin, et pas pour le nombre de
    composants
    Return :
        - Meilleur classifier entraine, avec le pre processing associe
    """
```

utilities.py

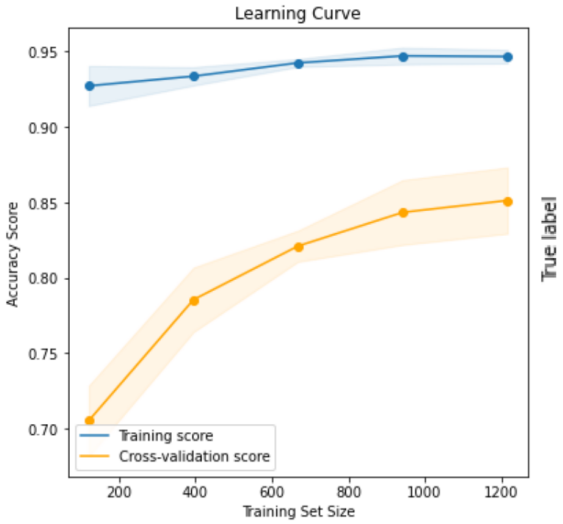
Un résultat possible :

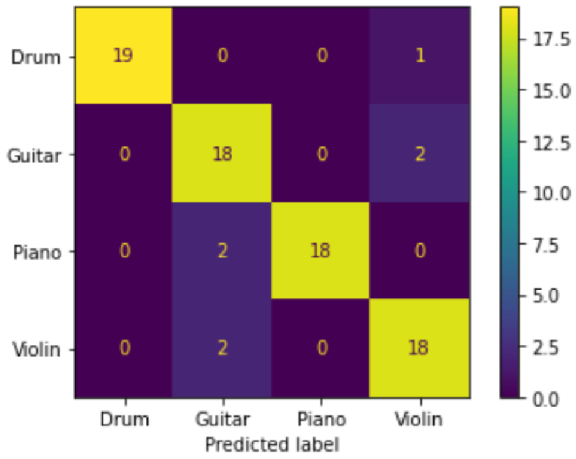


A partir de là, on fixe un classifier et un pré-processing.

8 Résultats

On a la learning-courbe à gauche, le résultat du test sur 80 fichiers à droite. Nous sommes assez satisfait du résultat. Nous avons testé le programme sur nos fichiers personnels, et les résultats sont concluants.





Comme on s'y attend, aucun son n'est confondu à tort avec une percussion, cette catégorie ayant un spectre très particulier.
En regardant les fichiers mal classés, on remarque que souvent, ce sont des fichiers non fiables (exemple: un piano qui joue avec une rythmique), ou des sons d'instruments tellement désaccordé que même une oreille entrainée aurait du mal à distinguer. On a aussi des fichiers qui durent quelques secondes mais où le son n'est joué qu'au tout début : comme on coupe au milieu, on analyse juste un fichier vide. C'est un risque qu'on est obligé de prendre, car vérifier que le son est présent quand on charge le fichier demande un temps de calcule beaucoup trop important. Nous sommes globalement très satisfait de nos résultats. Il est à noter que si on regarde, pour chaque fichier testé, non pas le résultat mais son pourcentage d'être dans une des catégorie, on peux voir qu'il y a une grosse marge d'amélioration (quand un fichier est prédit dans la bonne catégorie, le programme donne en fait une proba d'environ 50/60%, c'est améliorable.