

## CONFÉRENCE PEIP : RAPPORT FINAL

---

Équipe: **InstrumentAll**

Antoine Brenaget

Augustin De La Bourdonnaye

Arthur Domergue

Dorian Massamba

Marin Murgue

Maxime Fournier

Glen Rousseau

Yann Salmon

PeiPA2, Année 2021-2022

# Sommaire

1	Introduction . . . . .	2
2	Résumé du projet . . . . .	2
3	Aspects techniques et scientifiques . . . . .	3
3.1	Contexte social, scientifique et technique . . . . .	3
3.2	Problématique et plan pour y répondre . . . . .	6
3.3	Algorithme de prédiction de l'instrument joué . . . . .	6
3.4	Programmation de l'accordeur . . . . .	23
3.5	Les composants de l'accordeur . . . . .	27
3.6	L'assemblage . . . . .	29
4	Conclusion . . . . .	30
	Bibliographie . . . . .	33
5	Aspects gestion de projet . . . . .	34
5.1	Déroulement des séances . . . . .	35
5.2	Les problèmes rencontrés et résolus . . . . .	35
5.3	Comparaison prévisions/réalité . . . . .	38
	<b>Annexes</b>	<b>41</b>
1	Prise en main : comment VOUS pouvez tester l'algorithme . . . . .	42
2	Évaluer la performance d'un algorithme de prédiction . . . . .	45

# 1 Introduction

Dans ce rapport, nous allons vous présenter les résultats de nos travaux dans le cadre de la Conférence PeiP. Le thème central du projet fût l'étude du *timbre*. Nous avons passé le plus clair de notre temps sur deux applications de nos recherches : la construction d'un accordeur (pour guitare) et la création d'un algorithme capable de prédire l'instrument joué dans un fichier son.

## 2 Résumé du projet

Avant de commencer, rappelons la problématique de notre projet ainsi que la définition du timbre :

### Problématique

Comment caractériser le timbre d'un instrument, d'une voix ? Ou, autrement dit, qu'est-ce qui fait que pour une même note jouée par deux instruments, nous entendons deux sons différents ?

En première approximation, le timbre d'un son est un ensemble de propriétés acoustiques, autre que la hauteur, l'intensité et la durée, qui font que nous percevons ce son différemment justement lorsque la hauteur, l'intensité et la durée restent invariantes. Ainsi définit, le timbre semble être un terme « poubelle » dans lequel on met tout ce qui n'est pas encore bien connu. En effet, c'est en partie vrai et cela fait de ce terme un terme vague. Mais ce n'est pas pour cela que c'est un concept inintéressant.

Notons que nous parlons ici du timbre d'un son, et non d'un instrument ou d'une voix. En effet, pour être précis, il faudrait parler du timbre d'un son ou d'un environnement sonore spécifique plutôt que du timbre d'un instrument ou d'une voix. Notre perception d'un même son produit par un même instrument peut varier en fonction du contexte. Mais nous sentons bien que les sons produits par un instrument se distinguent le plus souvent clairement des sons produits par un autre instrument. Et en effet, chaque instrument a une sorte de « signature acoustique » composée des caractéristiques acoustiques communes aux sons qu'il produit, et donc génère des sons qui ont, en large proportion, les mêmes timbres. Ainsi, ce n'est pas totalement dénué de sens de parler du timbre d'un instrument ou d'une voix.

Pour répondre à notre problématique, nous avons fait des recherches bibliographiques. Nous avons appris que le timbre d'un son est fortement corrélé à une certaine fonction des *coefficients cépstraux de fréquence Mel* (**MFCC** pour « Mel-frequency cepstral coefficients » en anglais) de

celui-ci<sup>1</sup>. Cependant, il aurait été dommage de s'arrêter là. Nous avons décidé de réaliser deux sous-projets nous permettant d'appliquer ce que nous avons appris :

1. Construire un algorithme qui prend en entrée un fichier son (son d'un instrument) et qui renvoie l'instrument qui est joué dans ce son.
2. Fabriquer un accordeur pour guitare, c'est-à-dire un outil capable, si nous lui donnons une note en entrée, d'identifier si cette note est juste pour l'instrument en question, et si ce n'est pas le cas, s'il faut accorder vers les aigus ou vers les graves.

## 3 Aspects techniques et scientifiques

### 3.1 Contexte social, scientifique et technique

#### Contexte social

Notre groupe de Conférence PeiP compte huit membres. Nous sommes tous regroupés dans le même groupe de TD, nous avons donc l'habitude de travailler ensemble notamment au cours des divers projets proposés durant les semestres. C'est un avantage clé puisque grâce au fruit de notre expérience, nous travaillons efficacement ensemble. De plus, nous nous retrouvons régulièrement en dehors du temps scolaire donc nous nous connaissons personnellement en plus d'avoir des habitudes de travail communes. Notre équipe *InstrumentAll* est donc une équipe dynamique et soudée. Les interventions de John Kingston ont notamment permis de cibler avec précision les méthodes de travail de chacun ainsi que notre rapport avec les travaux de groupe. Nous avons alors listé trois de nos défauts qui risquaient de compromettre le déroulement du projet. L'objectif était de les exposer à un membre du groupe afin d'être tous conscients des difficultés de chacun en terme d'organisation, de motivation, etc. En se confiant à un seul membre du groupe, il était plus simple d'évoquer d'éventuelles complications à la fois relationnelles ou professionnelles. À ce stade du projet, nous avons su éviter les embûches que sèment la réalisation d'un travail en groupe, effet sous-marin, effet 8=4, etc., et aucune dispute ou tension n'étaient à signaler. Nous avons enfin dressé notre propre profil du collègue idéal pour le groupe, des qualités et compétences acquises lors de cette ConfPeip, et notamment nos défauts corrigés. Ce profil du « collègue idéal » est un objectif ambitieux à atteindre, mais l'idée est de s'en rapprocher, de faire des efforts pour progresser dans certaines compétences. La communication au sein de notre groupe est un de nos atouts. L'ambiance y est très bonne, les échanges entre tous les différents membres du groupe sont simples et se font sans rapport hiérarchique. Le chef de notre groupe maintient un rythme de travail et une rigueur dont le groupe ne souffre pas. Chaque membre

---

1. Nous expliquerons plus tard en détail ce que sont ces coefficients

du groupe prend l'initiative d'organiser son temps de travail en dehors des horaires dédiés à la conférence Peip, si nous jugeons nécessaires de terminer certaines tâches avant une date donnée. Nous nous sommes également imposés de ne jamais faire l'impasse sur un créneau de Conférence Peip organisé le mardi dans l'après-midi afin de conserver une bonne dynamique. Les réunions occupent une place importante dans l'organisation des travaux de groupe, nous n'avons organisé que trois séances de réunion puisque simplement nous ne ressentions pas le besoin d'en faire. La deuxième a été mise en place après la séance de gestion de projet avec John Kingston, afin de rediriger, réorienter le projet. Elle nous a permis de discuter et débattre de certains problèmes, comme développer une idée ou non. Une des raisons pour lesquelles nous avons limité les réunions c'est pour éviter les « réunionites ». Notre sujet étant assez large, il nous a fallu poser les limites du projet afin d'éviter tout hors-sujet. En travaillant tous dans la même pièce il était simple de capter l'attention du groupe pour mettre au point certains détails. Tout au long du projet, nous avons ainsi travaillé ensemble, dans la même salle, en organisant l'espace en pôles en fonction des tâches réparties dans les différentes équipes. Cette méthode de travail était efficace, des membres des équipes pouvaient venir aider d'autres équipes, impliquant ainsi un maximum de personnes dans le plus de tâches possibles.

## Contexte scientifique

Nous avons débuté les séances de ConfPeip notamment par des leçons introductives sur le thème « Musique et informatique ». Un premier cours magistral qui aura servi de socle commun pour l'ensemble de la promotion. La transformée de Fourier a été abordée ainsi que des connaissances générales sur les signaux, les notions de fréquence, d'amplitude, etc. Ensuite, un second cours avait pour objectif de développer la transformée de Fourier discrète, en étudiant les coefficients de Fourier. La confection de l'accordeur demandait une compréhension précise des différents composants utilisés. Les besoins et le fonctionnement de chaque composant devaient être connus, par exemple en sachant quel signal sera envoyé pour transmettre le message par l'intermédiaire d'un pin en particulier. Il est également nécessaire de prendre en compte l'analyse des puissances minimales et maximales de chaque composant. Dans le cas contraire, si la puissance fournie à un composant est trop faible, il ne fonctionnera pas, et dans le pire des cas si la puissance est trop forte, on risque de griller le composant. Par exemple, si on ne place pas une résistance avant le passage du courant dans une diode, cette dernière va griller. Notre projet est à dominante technique, les connaissances scientifiques nécessaires à sa réalisation sont minces. Une des causes de ce déséquilibre entre l'apport scientifique et technique au projet est la conception de l'accordeur. En effet, comme nous le verrons dans la partie dédiée au contexte technique, la transcription d'une part de nos programmes python dans un langage adapté fût

chronophage. Notre projet est à dominante informatique mais la partie électronique a introduit de nouvelles problématiques. C'est un point positif puisque ces travaux en électronique permettait de diversifier les tâches et offrait une dimension nouvelle à notre projet.

## Contexte technique

Notre formation étant à dominante théorique, il est naturel que les compétences techniques de chacun soient plutôt limitées, malgré quelques séances de travaux pratiques organisées dans l'année. Mais en contre-partie, nous avons une bonne expérience dans la réalisation de projet et dans les présentations orales. Lors de l'avant projet, nous avons estimé le matériel dont nous allions avoir besoin, ce dernier n'a pas évolué depuis. Les débuts de notre projet nécessitaient aucun matériel en particulier, excepté un accès à un ordinateur pour effectuer les nombreuses recherches bibliographiques. Ce détail n'a pas constitué une contrainte puisque chacun d'entre nous possédait un ordinateur portable, pouvant ainsi faire ses propres recherches en autonomie. De plus, des espaces de travail avec des postes informatiques étaient accessibles dans l'école, nous n'avions donc pas de problème de ce point de vue là. Ensuite, le second semestre commençant, nous avons eu besoin de mettre au point l'accordeur. Pour cela, nous avons eu recours à un kit « arduino starter »<sup>2</sup> contenant les composants nécessaires pour construire des circuits électroniques simples. Au sein de l'équipe, deux membres avaient des connaissances en électronique, Augustin et Antoine qui possédaient déjà de tels kits avant le projet. Cette discipline était donc une découverte pour le reste de l'équipe. En revanche, toute l'équipe avait des connaissances en algorithmie, certes avec plus ou moins d'aisance selon les membres mais nous possédions tous un socle commun. Des connaissances en algorithmie sont évidemment indispensables dans notre projet pour implémenter la transformée de Fourier (avec le FFT, Fast Fourier Transform), l'algorithme de reconnaissance d'instrument, etc. Pour l'algorithme de prédiction d'instrument, nous avons utilisé le langage Python. Les bibliothèques proposées par le langage nous ont bien aidé. On s'est ensuite servi du langage C++, pour implémenter un algorithme sur la carte Arduino qui détermine si la note jouée par un instrument est de la bonne fréquence, c'est-à-dire comment se situe cette dernière par rapport à la fréquence de référence pour la note en question. Nous avons utilisé une bibliothèque C++ pour implémenter la transformée de Fourier, mais nous avons modifié certaines fonctions, notamment celles de la transformée de Fourier pour réduire les calculs et donc la latence pour l'affichage sur l'accordeur.

---

2. <https://www.arduino.cc/en/Main/ArduinoStarterKit/?setlang=en>

## 3.2 Problématique et plan pour y répondre

Nous invitons le lecteur à relire la section 2 où nous donnons la problématique et expliquons, dans l'ensemble, comment nous avons fait pour y répondre.

## 3.3 Algorithme de prédiction de l'instrument joué

### Fonctionnement de l'algorithme

Pour l'algorithme de prédiction, nous avons utilisé le langage Python. La construction de l'algorithme peut se décomposer en quatre étapes :

1. **Préparation des données à passer en entrée de l'algorithme** : Nous partons d'une base de données composée de plusieurs centaines voire milliers de fichiers sons. Ce sont des extraits de certains instruments (guitare, clarinette, flûte, piano, etc.) de durées d'environ six secondes. Ensuite, nous extrayons des **caractéristiques** (essentiellement les MFCCs) pour chaque son. Nous organisons tout ça dans une matrice avec pour lignes les fichiers son et pour colonnes les caractéristiques. C'est cette matrice qui sera passée en entrée de l'algorithme. Précisons que dans les caractéristiques, nous incluons l'instrument joué dans le son. En effet, nous avons utilisé un algorithme d'apprentissage supervisé : à un tel algorithme, nous fournissons des « problèmes » (ici prédire l'instrument joué dans chaque son) et nous donnons les « réponses » (ici, l'instrument effectivement joué).
2. **Création de l'algorithme (ou la fonction) capable, pour un son (ou ses caractéristiques) en entrée, de fournir l'instrument joué dans ce son en sortie** : Cette étape est en réalité assez compliqué et chronophage. Nous avons décidé d'utiliser une librairie<sup>3</sup> fournissant divers algorithmes de prédiction car, étant donné l'objectif de notre projet (caractériser le timbre), nous n'avons pas intérêt à dépenser beaucoup de temps et d'effort là-dessus. Cette étape consistait donc simplement à choisir l'algorithme le plus efficace pour notre problème, puis à choisir les paramètres de ce dernier de manière judicieuse.
3. **Création de l'algorithme (bis)** : Avant de pouvoir faire des prédictions avec l'algorithme, il faut l'*entraîner* sur nos données. En effet, les algorithmes fournis par *scikit-learn* sont des algorithmes généraux qui font l'apprentissage, pas des algorithmes qui ont appris. Pour entraîner notre algorithme donc, *scikit-learn* offre une interface très simple. Le code est de la forme :

```
monAlgo.fit(mesDonnees)
```

---

3. Nous avons utilisé la librairie Python [scikit-learn](#).

Cet algorithme entraîné est notre algorithme final, prêt à faire des prédictions. Nous le stockons dans une variable :

```
algoFinal = monAlgo.fit(mesDonnees)
```

4. **Prédictions** : Allez dans votre navigateur web et tapez « musique guitare ». Choisissez une des vidéos parmi les premiers liens (faites attention à prendre une musique où il n'y a que de la guitare). Enregistrez un extrait de dix secondes en plein milieu de la vidéo. Nous donnerons le fichier résultant en entrée de notre *algoFinal*. Pour rappel, l'objectif de l'algorithme est de prédire l'instrument joué dans votre extrait. Et pour faire une prédiction, *scikit-learn* fournit de nouveau une interface très agréable :

```
prediction = algoFinal.predict(votreFichierSon)
```

En réalité, ce n'est pas tout à fait ça. Il nous faut extraire les caractéristiques de *votre-FichierSon* de la même manière qu'on a extrait les caractéristiques des sons d'« entraînement », puisque pour rappel, *algoFinal* prend les caractéristiques d'un son en entrée, et non le son en lui-même. Nous avons plutôt :

```
prediction = algoFinal.predict(caractDuSon)
```

Une fois ces quatre étapes réalisées, il est tout de même important de se faire une idée de la performance de notre algorithme, c'est-à-dire de sa capacité à faire des prédictions justes. Évaluer la performance d'un algorithme de prédiction est en fait une tâche complexe. En effet, il n'y a pas vraiment de mesure de la performance globale et satisfaisante dans tous les cas. Il y a plutôt différentes métriques qui rendent chacune compte d'une partie de ce qui constitue la performance (voir l'annexe 2 dédiée pour en savoir plus). C'est entre les mains du programmeur de choisir quels aspects de la performance lui importe le plus afin de savoir quoi mesurer et optimiser. Nous verrons plus loin en détails comment nous avons évalué notre algorithme.

Discutons maintenant des caractéristiques que nous avons extrait des sons, puisque c'est là où est l'enjeu. Pour chaque son, nous avons extrait les caractéristiques (ou critères) suivantes :

1. harmonic
2. mfcc
3. chroma
4. contrast
5. instrument

Expliquons de suite ce qu'elles sont :



1. Nous pouvons décomposer un son en deux parties : une partie « harmonique » et une partie « percussion ». Les sons harmoniques sont les sons que l'on perçoit comme ayant une certaine hauteur. Par exemple, le son d'un violon est un son harmonique. À l'inverse les sons de percussion sont généralement les sons qui résultent d'un choc entre deux objets. Ils n'ont pas vraiment de hauteur et sont très localisés dans le temps. La majorité des sons du quotidien sont des mélanges de sons harmoniques et de percussion. Par exemple, une note jouée avec un piano commence par un son de percussion (dû au marteau qui frappe la corde), puis est suivi d'un son harmonique (dû à la vibration de la corde). Nous calculons<sup>2</sup> la caractéristique « harmonic » à l'aide d'un algorithme HPSS (Harmonic-Percussive Source Separation) fourni par la librairie *Librosa* [13]. Cette variable prend la valeur 0 si le son est majoritairement un son de percussion, et 1 si c'est majoritairement un son harmonique.
2. « MFCC » est l'abréviation de « Mel-Frequency Cepstrum Coefficients ». Cette caractéristique est la plus importante de toute dans le sens où elle est la plus corrélée au timbre, celle qui a le plus grand pouvoir prédictif. Pour comprendre les MFCCs, nous allons expliquer chaque terme en commençant par « cepstrum ». L'intuition fondamentale en traitement du son (et d'en beaucoup d'autres domaines d'ailleurs) est de transformer le signal initialement exprimé en fonction du temps afin de l'exprimer en fonction d'une fréquence. Cela permet de faire ressortir le caractère périodique du signal, et cela s'avère très utile. La transformation qui permet ce changement de domaine (du domaine « temps » au domaine « fréquence ») est la transformée de Fourier (discrète pour un ordinateur). Le graphe de la transformée de Fourier d'un signal (fréquence en abscisse et intensité/amplitude en ordonnée) est appelé le **spectre de fréquence** (*spectrum* en anglais) du signal. Et bien dans l'idée, le cepstrum d'un signal est le « spectre » obtenu en réappliquant la transformée de Fourier au spectre de fréquence de ce signal (vous aurez remarqué la ressemblance entre *spectrum* et *cepstrum*<sup>4</sup>). Ce n'est pas tout à fait vrai, mais continuons. À l'origine, l'idée fût motivée par l'étude des tremblements de terre, pour lesquels nous souhaitions étudier les échos. L'hypothèse était qu'un signal peut se décomposer en une somme d'un signal « pur » et d'un écho :

$$x(t) = s(t) + \alpha s(t - \tau)$$

où  $x(t)$  est l'amplitude du signal à l'instant  $t$ ,  $s(t)$  l'amplitude du signal pur à l'instant  $t$  et  $\alpha s(t - \tau)$  l'amplitude de l'écho à l'instant  $t$  (mais initié par le signal pur à l'instant

---

4. La source du nom (et du concept) *cepstrum* semble être cet article : B.P. Bogert, M.J.R. Healy, and J.W. Tukey, « The quefrency analysis of time series for echoes : Cepstrum, pseudo-autocovariance, cross-cepstrum, and saphe cracking » in Time Series Analysis, M. Rosenblatt, Ed., 1963, ch.15, pp. 209–243.

$t - \tau$  où  $\tau$  est le retard). La *densité spectrale d'énergie* de  $x$  est donnée par :

$$DS_x(f) = |X(f)|^2 = |S(f)|^2(1 + \alpha^2 + 2\alpha \cos(2\pi f\tau))$$

où  $X$  et  $S$  sont les transformées de Fourier de  $x$  et  $s$  respectivement.  $DS_x(f)$  représente la quantité d'« énergie » du signal lorsque sa fréquence est  $f$ . Nous pouvons déjà voir qu'une partie de l'expression où intervient  $\tau$  se démarque. Pour accentuer la démarcation, nous prenons le logarithme de  $DS_x$  :

$$C_x(f) = \log(DS_x(f)) = \log(|S(f)|^2) + \log((1 + \alpha^2 + 2\alpha \cos(2\pi f\tau)))$$
<sup>5</sup>

Nous pouvons voir  $C_x$  comme un signal temporel (si ce n'est qu'ici nous avons remplacé la variable  $t$  par  $f$ ) de fréquence  $\tau$ . Or que faisons-nous lorsque nous souhaitons mettre en avant le caractère périodique d'un signal (ce qui revient ici à mettre en valeur le caractère périodique de l'écho) ? Une transformée de Fourier ! Le « cepstrum » du signal  $x$  est donc le « spectre » obtenu à partir de la transformée de Fourier de  $C_x$ . Il n'a pas été appelé spectre car nous ne sommes plus dans le domaine du temps ni dans celui des fréquences. Les auteurs du terme « cepstrum » l'ont appelé le « que-frequency domain ». Pour appliquer cette idée à l'étude du timbre, il nous faut rajouter une petite étape. Nous savons (grâce à l'expérience) que toute variation de fréquence n'est pas perçue de la même manière par l'oreille humaine. Dans un premier temps, nous sommes en moyenne sensible qu'à un certain intervalle de fréquence : de 20 Hz à 20 kHz. Le second phénomène est que nous percevons de moins en moins bien les variations de fréquence lorsque celle-ci augmente. Autrement dit, nous jugeons que le changement de hauteur d'un son lorsque nous passons de 50 Hz à 100 Hz est équivalent au changement de hauteur lorsque nous passons de 10 000 Hz à 20 000 Hz<sup>6</sup>. Le rapport entre ces fréquences est de 10, mais la différence est de 50 dans le premier cas et 10 000 dans le second. Pour rendre compte de ces phénomènes, nous transformons la fréquence en **fréquence mel** selon la formule (empirique) :

$$m = 1127 \ln \left( 1 + \frac{f}{700} \right)$$

Cette étape se fait juste après le calcul de la densité spectrale d'énergie du signal. Nous obtenons donc un cepstrum avec pour abscisse la fréquence mel et pour ordonnée la quantité d'énergie. Les MFCCs sont simplement des coefficients qui caractérisent le cepstrum exprimé dans le domaine des fréquences mel<sup>7</sup>. Notre caractéristique *mfcc* est donc une liste de nombres (de taille variable).

---

5. Nous ne précisons pas la base du logarithme ici car ce n'est pas très important. L'objectif est simplement de transformer un produit en somme, et tous les logarithmes nous permettent de le faire.

6. Les fréquences choisies ne sont là qu'à titre indicatif. Ce n'est sûrement pas vrai dans la réalité.

7. Nous ne détaillerons pas comment ces coefficients sont calculés exactement.

3. « chroma » est une liste de douze nombres, par exemple

`chroma = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]`

Chacun des douze nombres correspond à l'énergie du signal pour chaque note de la gamme tempérée. Les douze notes de la gamme tempérée sont (notées ainsi dans le milieu anglophone) C, C#, D, D#, E, E#, F, F#, G, G#, A, A#. Notons qu'une même note est associée à plusieurs fréquences, étant donné qu'on retrouve cette note dans chaque octave. Par exemple, la note A (le La) est associée aux fréquences 27.50 Hz, 55.00 Hz, 110.00 Hz, 220.00 Hz, 440.00 Hz, 880.00 Hz, etc., qui correspondent respectivement aux octaves 0, 1, 2, 3, 4, 5, etc.<sup>8</sup> Nous pouvons noter plusieurs choses. D'abord, le musicien se rendra compte que la fréquence 440.00 Hz est un La<sub>3</sub>, alors qu'elle est ici présentée comme étant un A<sub>4</sub>. En effet, comme nous l'avons déjà dit, le A correspond au La (de manière générale, les notes Do, Ré, Mi, Fa, Sol, La, Si correspondent respectivement aux notes anglophones C, D, E, F, G, A, B). Mais les anglophones numérotent aussi les octaves différemment de nous : ils numérotent les octaves 0, 1, 2, 3, 4, 5, 6, 7 là où nous les numérotions -2, -1, 1, 2, 3, 4, 5, 6. De plus, nous pouvons remarquer que les fréquences sont multipliées par deux à chaque octave. C'est en effet comme cela qu'est définie une octave dans la gamme tempérée : prenez la fréquence d'une note dans n'importe quelle octave et divisez-la ou multipliez-la par deux pour descendre ou monter d'une octave. Revenons à notre caractéristique *chroma*. L'idée est la suivante : les sons associés à une même note (malgré des fréquences différentes) ont « la même couleur ». On dit qu'une note correspond à une couleur, ou un « chroma » (d'où le nom). Précisons que les chromas sont normalisés entre 0 et 1.

4. *contrast* (voir [https://librosa.org/doc/latest/generated/librosa.feature.spectral\\_contrast.html#idl](https://librosa.org/doc/latest/generated/librosa.feature.spectral_contrast.html#idl) et le papier)
5. La caractéristique *instrument* d'un son a pour valeur l'instrument joué dans ce son. Par exemple, notre algorithme final est entraîné sur quatre instruments : la guitare, la clarinette, la flûte et le piano. Donc, par exemple,

`instrument = "guitar"`

En réalité, *instrument* prend une valeur numérique car l'algorithme oblige à ce qu'on lui fournisse des valeurs entières pour distinguer les catégories qui feront l'objet des prédictions. Pour répondre à cette exigence, nous avons simplement placé nos instruments dans une liste :

---

8. Voir <https://pages.mtu.edu/~suits/notefreqs.html>

```
INSTRUMENTS = ["guitar", "clarinet", "flute", "piano"]
```

et nous avons associé à chacun leur indice dans cette liste (0 pour "guitar", 1 pour "clarinet", etc.). Ainsi, une prédiction faite par l'algorithme est sous forme d'entier (0, 1, 2, ou 3), et il faut faire la transformation inverse pour savoir quel instrument a été prédit.






Dans cette sous-section, nous avons donné un aperçu global du fonctionnement de notre algorithme. À ce stade, vous devriez avoir compris comment les « choses s'imbriquent les unes dans les autres », mais c'est normal si vous vous demandez toujours comment *exactement* nous avons fait tout ça. Seul un coup d'oeil sur le code vous permettra de palier cette incompréhension, et c'est pourquoi nous allons tout de suite expliquer celui-ci.

## Implémentation de l'algorithme : explication des codes

Nous avons donné une vue d'ensemble du fonctionnement de l'algorithme dans la section précédente. Ici, nous allons vous présenter notre implémentation de cet algorithme. La description ne sera que moyennement granulaire puisque nous jugeons inutile d'expliquer le code ligne par ligne, étant donné que le code parle pour lui-même. C'est plutôt l'explication de notre manière d'organiser le code et les motivations derrière chaque fonction qui apportera une véritable valeur ajoutée. Sans plus attendre, allons-y !

Pour suivre cette partie, il vous faudra avoir les codes. Soit vous avez déjà téléchargé le dossier .zip les contenant, soit vous pouvez le faire en suivant [ce lien](#) (vous devrez être connecté à Gitlab avec votre compte université). Si vous ne voulez pas vous encombrer du code, vous pouvez simplement garder une fenêtre ouverte sur le Gitlab du projet et visionner le code depuis le site. Vous ne pourrez cependant pas le tester.

Commençons par expliquer la structure du dossier.

Name	Last commit	Last update
 data	amélioration fructueuse du modèle	1 week ago
 src	amélioration fructueuse du modèle	1 week ago
 .gitignore	prediction sur son enregistre par micro	2 months ago
 README.md	Initial commit	3 months ago
 dependencies.txt	record et prediction_micro	2 months ago

Nous avons un dossier *data* dans lequel nous plaçons tous les fichiers sons, que ce soit ceux qui servent à l'entraînement du modèle ou ceux qui servent à le tester (c'est dans cette dernière catégorie que nous trouvons les sons enregistrés par nous-mêmes, avec un micro). Il y a aussi un autre type de fichier (extension .pickle) que nous détaillerons juste après.

L'autre dossier est le dossier *src* (diminutif de *source*) : c'est là où nous plaçons tous les fichiers contenant les codes (des fichiers Python donc, avec pour extension *.py*).

Enfin, vous pouvez observer trois fichiers qui n'ont en fait pas un grand intérêt pour nous. Disons que ce sont des fichiers de « configuration » du projet. Expliquons l'utilité de chacun rapidement :

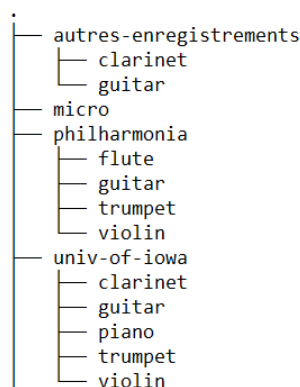
- **.gitignore** : Pour ce projet, nous avons utilisé un logiciel dit de « versionnage » appelé [Git](#) (notons d'ailleurs que Github et Gitlab sont principalement des interfaces web de Git, même s'ils offrent d'autres fonctionnalités). Un logiciel de versionnage permet de garder un historique d'un projet. Cela permet entre autres de revenir en arrière dans un projet si un bug a été malencontreusement introduit. Dans le fichier *.gitignore*, nous écrivons les chemins des fichiers et dossiers que nous ne voulons pas que Git enregistre/prenne en compte dans l'historique (nous voulons qu'il les « ignore », d'où le nom du fichier).
- **README.md** (se lit « read me ») : Ce fichier est un fichier *markdown* (extension *.md*). *Markdown* est un langage dit de « balisage ». C'est similaire à  $\text{\LaTeX}$  mais en plus simple. Ce fichier est donc une sorte de fichier texte où nous expliquons dans l'ensemble le projet, où nous donnons des instructions à ceux qui voudraient l'explorer.
- **dependencies.txt** : C'est un fichier texte dans lequel sont écrites les librairies Python utilisées dans le projet, ainsi que leurs versions. Cela permet à ceux qui veulent faire fonctionner le projet sur leur ordinateur d'installer les bonnes librairies. Mais ça nous permet avant tout d'explicitier les dépendances de notre projet.

Rentrons maintenant dans le dossier *src* pour y voir les codes. Nous ferons des allers-retours vers le dossier *data* entre temps.

 prediction.py	amélioration fructueuse du modèle	1 week ago
 prediction_micro.py	amélioration fructueuse du modèle	1 week ago
 prepa_donnees.py	amélioration fructueuse du modèle	1 week ago
 record.py	ajout de commentaires	1 month ago
 record_then_predict.py	amélioration fructueuse du modèle	1 week ago
 utils.py	amélioration fructueuse du modèle	1 week ago

Vous rappelez-vous la première étape du fonctionnement de l'algorithme ? : extraire les caractéristiques des fichiers sons d'entraînement. Notre base de données est constituée de divers fichiers sons pour divers instruments, et provenant de quatre sources (dans l'ordre d'importance) : [University of Iowa](#), [Philharmonia](#), [Youtube](#), nos propres enregistrements (de guitare et de clarinette). Vous ne les trouverez pas sur Gitlab car il était compliqué d'envoyer autant de données sur le nuage, mais sachez qu'ils sont normalement localisés dans le dossier *data/sons*

qui a la structure suivante :



Pour l'extraction des caractéristiques donc, tout se passe dans *prepa\_donnees.py*. Dans ce fichier, il y a trois fonctions principales (liées les unes aux autres) :

1. **extract\_features** : Cette fonction prend le chemin d'un fichier son d'extension .wav en entrée. Elle renvoie une liste contenant les fameuses caractéristiques du son dont nous avons parlé précédemment.
2. **do\_single\_feature\_extraction** : Cette fonction prend le chemin d'un fichier son d'extension .wav, l'instrument joué dans ce son et la liste des instruments de la base de données en entrée. Elle renvoie une matrice<sup>9</sup> avec pour seule ligne le nom du fichier son, et pour colonnes ses caractéristiques. Cette fonction réutilise *extract\_features* et fait en effet essentiellement le même travail si ce n'est qu'elle place les données dans une structure qui sera plus facile à manipuler plus tard. En fait, nous utilisons cette fonction pour extraire les caractéristiques des fichiers sons que nous avons enregistrés pour tester l'algorithme. Mais pour les fichiers sons d'entraînement, nous avons écrit une autre fonction qui traverse tous les sous-dossiers de la base de données, puis réunit toutes les caractéristiques calculées dans une même matrice qui sera donnée à l'algorithme pour l'entraîner : c'est la fonction *do\_feature\_extraction*.
3. **do\_feature\_extraction** : Cette fonction prend en entrée la liste des instruments de la base de données et le chemin d'un dossier contenant des sous-dossiers ayant pour noms ces instruments. Par exemple, nous pouvons appeler la fonction de la manière suivante :

```
do_feature_extraction("data/philharmonia/", INSTRUMENTS)
```

Nous avons utilisé cette fonction pour *data/philharmonia*, *data/univ-of-iowa* et *data/autres-enregistrements*, puis nous avons réuni les trois dataframes obtenus en un seul à l'aide

---

9. La structure exacte est un *dataframe*. C'est une structure fournie par la librairie [pandas](#) que nous pouvons considérer comme étant un tableau/une matrice.

d'une fonction [pandas](#).

L'extraction des caractéristiques est ce qu'il y a de plus coûteux en temps. Pour notre base de données d'environ quatre cents fichiers sons (cent pour chaque instrument), il nous faut attendre aux alentours de vingt minutes. Afin d'éviter de refaire tous les calculs à chaque fois que nous souhaitons utiliser les caractéristiques, nous les « sérialisons ». Autrement dit, nous encodons la matrice des caractéristiques en binaire dans un fichier. Une librairie Python nous permet de faire ça très facilement : elle s'appelle [pickle](#). Les fichiers en binaire ont donc pour extension `.pickle`, et nous les stockons dans le dossier `data/serialized`. Pour récupérer la matrice après l'avoir sérialisée, nous la « désérialisons ». Les processus de sérialisation et désérialisation sont relativement rapides (quelques secondes tout au plus) <sup>10</sup>.

Ainsi, nous pouvons réutiliser la matrice des caractéristiques à notre guise, et principalement pour la fournir à l'algorithme de prédiction afin qu'il s'entraîne. La partie « entraînement » se passe dans le fichier `prediction.py` qui ne contient qu'une seule fonction : `make_random_forest_model`. Elle prend les données d'entraînement en entrée (c'est-à-dire la matrice des caractéristiques <sup>11</sup>) et renvoie un algorithme de prédiction entraîné de type `RandomForestClassifier`. Cet algorithme est fourni par une librairie Python appelée [scikit-learn](#), et c'est un *classifieur*, c'est-à-dire un algorithme qui prédit à quelle catégorie (dans notre cas à quel instrument) appartient un objet (dans notre cas une bande son). « Random Forest » se traduit par « **forêt d'arbres de décision** » en français, et nous allons de suite expliquer son fonctionnement.

## Fonctionnement d'une forêt d'arbres de décision

Une *forêt d'arbres de décision* est un algorithme d'apprentissage qui est constitué de « sous-algorithmes » appelés *arbres de décision*. Ces arbres de décision sont entraînés sur différents morceaux de la base de données, morceaux qui se recoupent dans une certaine mesure. Dans notre cas, nous utilisons la version « classifieur » de l'algorithme, mais une version « régression » existe (où l'on prédit une valeur continue et non une classe/catégorie). Lorsque nous demandons à l'algorithme de faire une prédiction, chaque arbre de décision tente de prédire la classe (dans notre cas l'instrument) à laquelle appartient l'objet de la prédiction (dans notre cas un son). Les prédictions faites sont ensuite agrégées de manière à obtenir la prédiction de l'ensemble

---

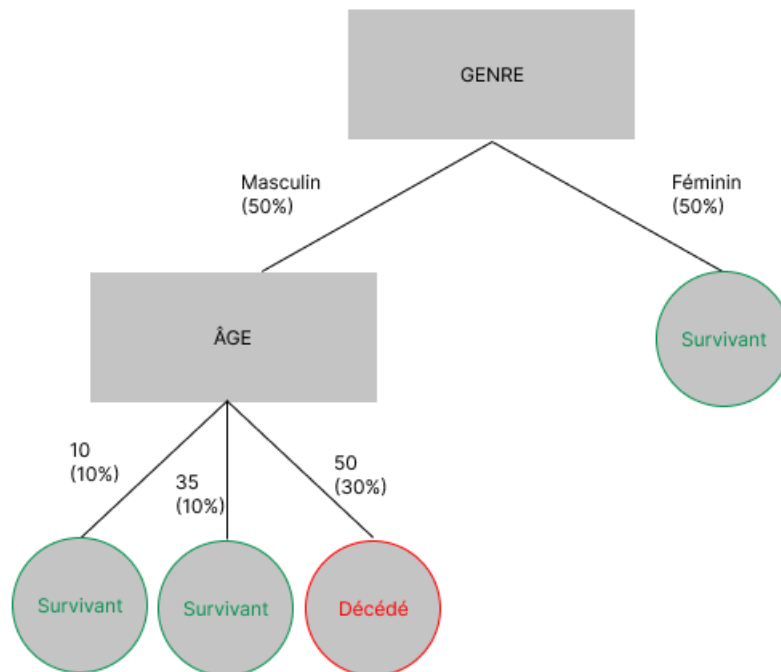
10. Nous vous avouons que nous ne savons pas comment fonctionnent les processus de sérialisation et de désérialisation. En effet, cela peut paraître magique, et nous partageons cette impression ! Nous invitons donc le lecteur curieux à mener ses recherches.

11. En réalité, elle ne prend pas directement la matrice des caractéristiques. Elle prend une matrice avec pour seule colonne la caractéristique à prédire, la colonne « instrument » ici, et une autre matrice avec toutes les autres caractéristiques.

des arbres<sup>12</sup>. Maintenant, vous vous demandez certainement comment fonctionne un arbre de décision, et c'est ce que nous allons voir.

Un arbre de décision est essentiellement un arbre (en tant que structure de données en informatique<sup>13</sup>) ayant pour noeuds des caractéristiques, et pour branches les valeurs de ces caractéristiques. Partons d'un exemple simple afin de contextualiser les choses :

	A	B	C	D
1	Nom	Genre	Âge	Survivant
2	p1	Masculin	50	Non
3	p2	Masculin	10	Oui
4	p3	Féminin	10	Oui
5	p4	Masculin	35	Non
6	p5	Féminin	10	Oui
7	p6	Féminin	35	Oui
8	p7	Féminin	50	Non
9	p8	Masculin	50	Non
10	p9	Masculin	50	Non
11	p10	Féminin	10	Oui



Source : Inspiré de « [Decision tree learning](#) » - Wikipédia

Ici, le jeu de données est un ensemble de personnes (les passagers du Titanic) qui ont certaines caractéristiques : genre et âge. L'objectif est de prédire, sachant ses caractéristiques, si une personne est décédée suite au naufrage du Titanic ou non. C'est donc un problème de classification où les classes sont « décédé.e » et « survivant.e ». L'arbre de décision ci-dessus

12. La probabilité qu'une classe soit la bonne est simplement le nombre de prédictions en faveur de cette classe parmi toutes les prédictions faites par les arbres de décision. C'est ensuite la classe qui obtient la plus haute probabilité (le plus de « votes ») qui sera la prédiction finale de la forêt d'arbres.

13. Voir [cet article](#) si vous n'êtes pas familier avec le concept.



est entraîné. Admettons que nous voulons obtenir sa prédiction pour une personne de genre masculin et d'âge cinquante ans. Et bien il suffit de partir du haut de l'arbre et de descendre les branches en fonction des caractéristiques de cette personne. Nous aboutissons à une feuille de l'arbre qui correspond à une classe : la classe prédite pour cette personne (en l'occurrence, « décédé »). L'idée de l'arbre de décision est en fait de diviser la base de données d'entraînement en sous-groupes en fonction des valeurs prises par chaque caractéristique, de sorte qu'en bas de l'arbre, nous obtenions des sous-groupes qui correspondent chacun à une seule classe. Par exemple, si l'arbre c'était arrêté à la caractéristique « genre », il n'aurait pas pu trancher entre « survivant.e » et « décédé.e » pour un homme/garçon puisqu'il y aurait eu des échantillons « décédés » et des échantillons « survivants » dans le nœud de gauche.

Jusqu'ici, nous avons vu comment l'arbre opère une fois entraîné, mais comment se construit-il ? Quelles caractéristiques choisir et dans quel ordre est-il avantageux de les placer ? Pour rappel, l'objectif est de diviser la base de données en sous-groupes pour lesquels chaque donnée appartient à une même classe, et ceux avec un minimum de caractéristiques/nœuds. Nous cherchons donc la caractéristique qui va nous apporter le plus d'**information** concernant la classe dans laquelle se situe une personne. Nous aimerions avoir une manière systématique et rigoureuse de quantifier l'apport d'information d'une caractéristique. Cela va nécessiter quelques notions de théorie de l'information, mais n'ayez pas peur, ce qu'il se cache derrière tout ça ce sont simplement des probabilités. Commençons à formaliser le problème. Nous pouvons considérer les caractéristiques comme étant des variables aléatoires finies, et les valeurs prises par celles-ci sont les valeurs présentes dans la base de données d'entraînement (le nombre de valeurs prises par une caractéristique est au maximum le nombre d'échantillons dans la base de données, or le nombre d'échantillons d'une base de données est fini). Nous introduisons une fonction  $I : P(\Omega) \rightarrow [0, +\infty[$  qui à un événement associe un réel positif qui représente la quantité d'**information** que nous nous attendons à obtenir si cet événement se réalise. L'intuition qu'il faut avoir ici est que l'information portée par un événement est directement liée à la probabilité que celui-ci est lieu. Plus un événement est probable, moins sa réalisation nous apportera d'information : « on s'y attendait déjà », en quelque sorte. À l'inverse, moins un événement est probable, plus sa réalisation sera surprenante et informatrice. Soit  $E$  un événement. Une relation entre l'information et la probabilité de  $E$  qui rend compte de la constatation précédente est :

$$I(E) = \frac{1}{\mathbb{P}(E)} \text{ }^{14}$$

Mais nous souhaitons vérifier d'autres propriétés :

— Nous aimerions que  $I$  soit décroissante en  $\mathbb{P}(E)$ .

---

14. Notons cependant que la probabilité de  $E$  doit être non nulle pour que cette expression soit définie.

- Nous aimerions que si  $\mathbb{P}(E) = 1$ , alors  $I(E) = 0$ .
- Nous aimerions que pour deux évènements indépendants A et B, l'information gagnée lors de la réalisation de ces deux évènements soit la somme de l'information gagnée pour chaque évènement séparément :  $I(A \cap B) = I(A) + I(B)$ .

Il s'avère que les seules solutions possibles pour vérifier ces propriétés sont :

$$I(E) = \log_b \left( \frac{1}{\mathbb{P}(E)} \right)$$

pour n'importe quelle base  $b > 0$  (nous vous laissons vérifier par vous-mêmes que les propriétés sont en effet vérifiées dans ce cas). La base la plus couramment choisie est  $b = 2$ , de sorte que l'information soit exprimée en *bits*. Nous définissons ensuite l'**information I d'une variable aléatoire finie**  $X : \Omega \rightarrow \{x_1, \dots, x_n\}$  comme étant la variable aléatoire qui à un évènement élémentaire  $a_i$  de l'univers associe l'information  $I(X = X(a_i))$ . Enfin, nous définissons l'**entropie H d'une variable aléatoire X** comme étant l'espérance de  $I(X)$  :

$$\begin{aligned} H(X) &= \mathbb{E}(I(X)) \\ &= \sum_{i=1}^n \mathbb{P}(I(X) = I(X = x_i)) I(X = x_i) \\ &= \sum_{i=1}^n \mathbb{P}(X = x_i) \log_2 \left( \frac{1}{\mathbb{P}(X = x_i)} \right) \quad 15 \end{aligned}$$

L'entropie de X représente ainsi l'information que nous espérons en moyenne obtenir lors d'un évènement ( $X = x_i$ ). Nous pouvons aussi dire qu'elle représente l'incertitude moyenne d'observer un évènement ( $X = x_i$ ).

Revenons à notre exemple du Titanic. Si nous considérons le genre et l'âge comme étant des variables aléatoires finies, alors nous pouvons calculer leurs entropies. Pour calculer la probabilité d'un évènement, par exemple (Genre = Masculin), il nous suffit de prendre la proportion d'individus dans la base de données qui vérifient cet évènement. Dans notre exemple,  $\mathbb{P}(\text{Genre} = \text{Masculin}) = \mathbb{P}(\text{Genre} = \text{Féminin}) = 0.5$ . L'entropie de Genre est donc <sup>16</sup> :

$$H(\text{Genre}) = \frac{1}{2} \log_2(2) + \frac{1}{2} \log_2(2) = \log_2(2) = 1 \text{ bit}$$

Pour la caractéristique Âge, nous avons :

$$H(\hat{\text{Age}}) = \frac{4}{10} \log_2 \left( \frac{10}{4} \right) + \frac{2}{10} \log_2 \left( \frac{10}{2} \right) + \frac{4}{10} \log_2 \left( \frac{10}{4} \right) \approx 1.522 \text{ bit}$$

Cependant, nous ne voulons pas que connaître l'information générale moyenne d'une caractéristique. Nous voulons connaître l'information qu'elle apporte **concernant** la classe (décédé.e

---

16. Pour pousser l'intuition derrière la notion d'entropie, nous vous invitons à regarder la [vidéo de 3Blue1Brown](#) sur le sujet, où il traite du cas du jeu *Wordle* (similaire au *Sutom* français).

ou survivant.e) d'un échantillon (une personne). Pour cela, nous introduisons une variable aléatoire  $C$  qui à une personne associe sa classe. De nouveau, la probabilité d'être dans une classe est déterminée par la proportion d'échantillons dans cette classe dans la base de données. Pour nous,  $\mathbb{P}(C = \text{décédé.e}) = 0.3$  et  $\mathbb{P}(C = \text{survivant.e}) = 0.7$ . Ainsi, nous pouvons définir le gain d'information  $GI$  d'une caractéristique  $X$  concernant  $C$  par :

$$GI(C, X) = H(C) - H(C|X)$$

où

$$\begin{aligned} H(C|X) &= \sum_i \sum_j \mathbb{P}(X = x_i, C = c_j) \log_2 \left( \frac{1}{\mathbb{P}(C = c_j | X = x_i)} \right) \\ &= \sum_i \sum_j \mathbb{P}(X = x_i, C = c_j) \log_2 \left( \frac{\mathbb{P}(X = x_i)}{\mathbb{P}(X = x_i, C = c_j)} \right) \end{aligned}$$

Ce qui est calculé ici c'est la diminution d'entropie de  $C$  (en moyenne) causée par la connaissance de la valeur de  $X$ . Autrement dit, plus  $GI(C, X)$  est grand, plus nous sommes certains de la valeur de  $C$  après avoir appris la valeur de  $X$ . A contrario, si  $C$  et  $X$  sont indépendants, alors  $H(C|X) = H(C)$ , donc  $GI(C, X) = 0$ . La construction de l'arbre de décision consiste donc à choisir la caractéristique qui maximise le gain d'information, puis de répéter le processus en restreignant les données considérées à chaque fois.

Reprenons notre exemple pour que ce soit plus clair. Calculons les gains d'information de Genre et Âge concernant  $C$  :

$$\begin{aligned} H(C) &= \frac{3}{10} \log_2 \left( \frac{10}{3} \right) + \frac{7}{10} \log_2 \left( \frac{10}{7} \right) \approx 0.811 \text{ bit} \\ H(C|\text{Genre}) &= \frac{3}{10} \log_2 \left( \frac{1}{2} \frac{10}{3} \right) + \frac{2}{10} \log_2 \left( \frac{1}{2} \frac{10}{2} \right) + 0 + \frac{1}{2} \log_2 \left( \frac{1}{2} 2 \right) \approx 0.485 \text{ bit} \\ H(C|\hat{\text{Age}}) &= 0 + \frac{4}{10} \log_2(1) + 0 + \frac{2}{10} \log_2(1) + \frac{3}{10} \log_2 \left( \frac{4}{10} \frac{10}{3} \right) + \frac{1}{10} \log_2(4) \approx 0.325 \text{ bit} \\ GI(C, \text{Genre}) &= H(C) - H(C|\text{Genre}) \approx 0.811 - 0.485 \approx 0.326 \\ GI(C, \hat{\text{Age}}) &= H(C) - H(C|\hat{\text{Age}}) \approx 0.811 - 0.325 \approx 0.486 \end{aligned}$$

Les résultats indiquent que c'est Âge qui offre le plus grand gain d'information ! Nous aurions donc dû commencer l'arbre avec elle. Dans notre exemple simplifié, nous n'avons que deux caractéristiques, donc le choix de la deuxième caractéristique est restreint à Genre. Cependant, si nous avions plus de caractéristiques, nous aurions répété le processus, c'est-à-dire calculer les gains d'information des caractéristiques restantes, mais cette fois-ci pour chaque noeud du deuxième étage (il y en aurait trois si nous avions commencé par Âge), et ce considérant non plus  $C$ , mais  $(C|\hat{\text{Age}}=10 \text{ ou } 35 \text{ ou } 50)$ . Nous restreignons la quantité de données au fur-et-à-mesure

selon leurs caractéristiques. Nous conditionnons C aux valeurs prises par les caractéristiques qui ont été choisies comme noeuds.

Nous espérons qu'à ce stade, le fonctionnement d'un arbre de décision est clair pour vous. Nous espérons que vous imaginez comment nous pouvons facilement généraliser l'exemple du Titanic à plus de caractéristiques et plus de données. Nous vous laissons d'ailleurs imaginer ce que ça peut donner avec notre base de données de fichiers sons et leurs caractéristiques.

En effet, c'est ainsi que fonctionne notre algorithme de prédiction, même si certaines subtilités ont été laissées de côté. Par exemple, nous n'avons pas parlé de comment déterminer le nombre d'arbres de décision à utiliser. Nous n'avons pas déterminé comment répartir le jeu de données pour entraîner chaque arbre de décision. De plus, un arbre de décision est particulièrement enclin au surapprentissage. En effet, si le jeu de données d'entraînement est plutôt petit et que l'on autorise l'arbre de décision à se développer sur une très grande profondeur (c'est-à-dire sur beaucoup d'étages de noeuds/conditions), alors il va essentiellement diviser le jeu de données au cas par cas : chaque feuille correspondra à un échantillon plutôt qu'à un sous-groupe d'échantillons. Ce comportement correspond plus à celui d'une mémoire inerte qu'à celui d'un algorithme qui a su généraliser. Cet effet de surapprentissage est cependant grandement mitigé par le fait que nous travaillons avec plusieurs arbres, ainsi que par un ajustement judicieux des paramètres de la forêt d'arbres (par exemple en spécifiant une profondeur maximale des arbres pas trop grande). Il faut aussi mentionner le fait qu'il existe d'autres stratégies pour choisir les conditions et l'ordre dans lequel les placer. Nous avons ici présenté la stratégie qui vise à maximiser le gain d'information à chaque condition. Cependant, il existe d'autres stratégies, dont une assez connue qui utilise ce qu'on appelle l'[indice de diversité Gini](#). Il existe des algorithmes standards pour créer des arbres de décision selon chacune des stratégies : l'algorithme [C4.5](#) pour le gain d'information et l'algorithme [CART](#) pour l'indice de diversité Gini. Enfin, des comportements indésirables peuvent survenir lorsqu'une des caractéristiques est fondamentalement continue (par exemple la taille d'un passager du Titanic) et que (sans surprise) chaque échantillon de la base de données (ou presque) a une valeur différente pour cette caractéristique.

### **Implémentation de l'algorithme : explication des codes (bis)**

À ce stade, nous avons notre algorithme final prêt à l'emploi. Mais tout comme pour la création de la matrice des caractéristiques, l'entraînement du modèle prend un certain temps. Nous sérialisons donc aussi l'algorithme entraîné. Vous pouvez retrouver le fichier binaire au chemin `data/serialized/trained-random-forest.pickle`. Nous pouvons maintenant passer à l'étape « test » de l'algorithme où nous allons faire des prédictions.

Pour commencer, il faut savoir que nous n'utilisons pas toute la base de données pour entraî-

ner le modèle. Nous la divisons en deux parties (de manière aléatoire) avec environ deux tiers des données pour l'entraînement et un tiers pour tester le modèle entraîné. Nous réalisons cela avec la fonction `train_test_split` de scikit-learn. Ainsi, nous commençons par tester l'algorithme sur la partie des données réservée au test. Nous obtenons de très bons résultats<sup>17</sup> :

Performance report:				
	precision	recall	f1-score	support
clarinet	1.00	1.00	1.00	25
flute	1.00	1.00	1.00	25
guitar	1.00	1.00	1.00	30
piano	1.00	1.00	1.00	22
accuracy			1.00	102
macro avg	1.00	1.00	1.00	102
weighted avg	1.00	1.00	1.00	102

(a) Diverses métriques de performance pour les données « test »

True Label	guitar	clarinet	flute	piano
guitar	30	0	0	0
clarinet	0	22	0	0
flute	0	0	25	0
piano	0	0	0	25
		Predicted Label		

(b) Matrice de confusion pour les données « test »

Nous pourrions argumenter que les résultats sont mêmes trop bons. Notre algorithme est entraîné sur une relativement petite base de données, ce qui fait que celui-ci *surapprend* probablement en partie. Le phénomène de surapprentissage (*overfitting* en anglais) est le fait qu'un modèle correspond trop précisément aux données, de sorte qu'il se soit focalisé sur des détails spécifiques aux données d'entraînement et non sur des régularités générales. On s'aperçoit qu'un algorithme à surappris justement lorsqu'il a de très bons résultats pour des données similaires à ses données d'entraînement, mais qu'il échoue lamentablement lorsque les données sont ne serait-ce que légèrement différentes. Dans une certaine mesure, c'est ce qu'il nous est arrivé. Notre groupe possède deux instruments : une guitare et une clarinette. C'est avec ces instruments que nous avons testé notre algorithme. Nos algorithmes précédents avaient de très bons résultats sur les données de test, mais sur nos enregistrements, ils se trompaient souvent. Ils faisaient tout le temps de bonnes prédictions pour la clarinette mais ils confondaient la guitare avec la clarinette au moins trois quarts du temps. Pour résoudre ce problème, nous avons apporté deux solutions :

1. Une première constatation que nous avons faite est que la distribution des données dans notre base était déséquilibrée. Environ 70% des sons étaient des sons de flûte, et les 30% restants se départageaient équitablement entre les trois autres instruments. Nous avons donc enlevé des sons de flûte et ré-entraîné l'algorithme sur un jeu de données équilibré.
2. Une deuxième solution a été de diversifier nos sources de données. Au départ, nous n'uti-

17. Voir l'annexe 2 dédiée aux méthodes d'évaluation de la performance d'un algorithme de classification.

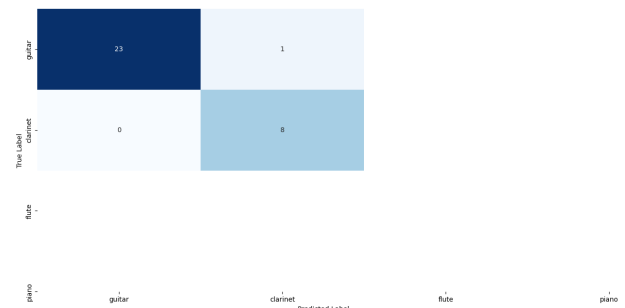
lisons que les sons de [University of Iowa](#). Nous avons ajouté trois autres sources citées précédemment : [Philharmonia](#), [Youtube](#) et nos propres enregistrements (de guitare et de clarinette).

- (Bonus) Vous pourrez voir dans l'implémentation de la fonction `make_random_forest_model` que nous utilisons une fonction [RandomizedSearchCV](#). Elle teste l'algorithme `RandomForestClassifier` avec différents paramètres, puis renvoie l'algorithme avec les paramètres qui maximisent la performance.

C'est surtout la deuxième solution qui a apporté le plus d'amélioration. Voici la performance de notre algorithme final sur les sons enregistrés par nous-mêmes (une vingtaine de sons de clarinette et une dizaine de sons de guitare) :

	precision	recall	f1-score	support
clarinet	0.89	1.00	0.94	8
guitar	1.00	0.96	0.98	24
accuracy			0.97	32
macro avg	0.94	0.98	0.96	32
weighted avg	0.97	0.97	0.97	32

(a) Diverses métriques de performance pour nos enregistrements



(b) Matrice de confusion pour nos enregistrements

Nous voyons dans la matrice de confusion que seul un son de guitare parmi vingt-quatre a été prédit par l'algorithme comme étant un son de clarinette.

Passons au fichier où nous avons le code qui concerne les prédictions sur nos enregistrements : le fichier `prediction_micro.py`. Avant cela, faisons un petit détour dans le fichier `record.py`. Lorsque nous exécutons ce fichier dans une invite de commande, il nous est demandé d'entrer quelques informations concernant l'enregistrement audio que nous nous apprêtons à faire :

```
(venv) yanns@laptop-yanns ~\Documents\yann\studies\cours\peip2\conf-peip\code\conf-peip\reconnaissance-timbre
Entrez le nom du fichier .wav dans lequel sera stocké le son que vous vous apprêtez à enregistrer: mon-fichier
Entrez la durée de l'enregistrement, en secondes (entier ou nombre à virgule): 10
Tapez sur 'Entrer' pour lancer l'enregistrement.
```

Une fois les informations données, nous pouvons taper sur la touche `Enter` pour lancer l'enregistrement. Le script Python va accéder au micro entrant du PC et récupérer les données captées par celui-ci. Nous écrivons ensuite ces données dans un fichier `.wav` (ici dans `data/sons/micro/mon-fichier.wav`).

Nous avons un autre fichier similaire, `record_then_predict.py`, qui, comme son nom l'indique, enregistre un son, puis calcule et affiche directement la prédiction de l'algorithme pour ce son.

Ce fichier nous permet simplement de gagner du temps lors de nos tests.

Revenons au fichier *prediction\_micro.py*. On y trouve trois fonctions :

- **make\_single\_prediction** : Cette fonction prend les éléments suivants en entrée :
  - le chemin d'un fichier son .wav
  - l'instrument joué dans ce fichier son
  - la liste des instruments de la base de données
  - le chemin du fichier .pickle dans lequel se trouve l'algorithme entraîné sérialisé

Elle renvoie la prédiction de l'algorithme pour le fichier son passé en argument, ainsi que les probabilités assignées par l'algorithme pour chaque instrument. En effet, nous pouvons demander à l'algorithme de nous donner, pour chaque instrument, les chances que celui-ci soit effectivement le bon instrument. Ainsi, pour chaque instrument qu'il connaît (instruments sur lesquels il a été entraîné), il fournit un nombre entre 0 et 1 représentant la probabilité que selon lui l'instrument soit le bon. Pour sa prédiction finale, il prend l'instrument qui a la plus haute probabilité d'être le bon. Notons que cette fonction réutilise la fonction *do\_single\_feature\_extraction* vue précédemment.

- **fait\_pred\_sons\_micro** : Cette fonction prend le chemin du fichier .pickle dans lequel se trouve l'algorithme ainsi que la liste des instruments de la base de données. Son job est de traverser le dossier *data/sons/micro* dans lequel se trouve tous les sons que nous avons enregistré, puis d'amasser les prédictions de l'algorithme sur tous ces sons. Elle renvoie une liste contenant les prédictions et une liste (de listes) contenant les probabilités associées. Cela nous permet ensuite de calculer diverses métriques de performance et d'ainsi avoir une vue d'ensemble sur les performances de l'algorithme pour les sons enregistrés.
- **show\_results** : Cette fonction nous permet juste d'afficher la prédiction et les probabilités pour un fichier son afin qu'il soit plus aisé de les lire.

Enfin, nous avons un dernier fichier qui s'appelle *utils.py* (*utils* est un diminutif de *utilities*). Nous y avons écrit diverses fonctions et variables qui nous ont servi à plusieurs reprises dans le projet, de sorte que nous n'avions plus qu'à les importer lorsque nous en avions besoin.

Nous terminons ici la présentation de la structure du projet et du contenu des fichiers de code. En regardant ces fichiers cependant, vous aurez peut être remarqué qu'on trouve dans chacun (à la fin) un bloc de code commençant par

```
if __name__ == "__main__":  
    # diverses lignes de code...
```

En fait, les fichiers suivent tous la même structure. En haut, nous mettons les « importations », c'est-à-dire des lignes de code où nous précisons quelles librairies (et quelles fonctions, objets dans ces librairies) nous utilisons dans le fichier. On dit qu'on les « importe ». Ensuite, nous avons

les déclarations de fonction. Si nous exécutons le script qu'avec les déclarations de fonction, il ne se passerait rien. Nous sommes simplement en train de les définir. Tout se passe justement à la fin du fichier, dans le bloc ci-dessus. C'est là où nous définissons des variables et appelons des fonctions. C'est là où les choses se font. Vous remarquerez que c'est un bloc if, mais la condition n'est sans doute pas claire. La variable `__name__` est une variable créée automatiquement par Python pour chaque fichier. Python lui assigne une valeur lorsque le fichier est exécuté en fonction du contexte dans lequel il l'est. Par exemple, lorsque l'on exécute un fichier en tant que *script* dans une invite de commande, Python donne la valeur `"__main__"` à `__name__`. Lorsque le fichier est appelé depuis un autre fichier du projet, en tant que *module* donc, la valeur que prend `__name__` est le nom du fichier. Ainsi le bloc if s'exécute que lorsque le fichier est appelé en tant que script, et non lorsqu'il l'est en tant que module : c'est ce que nous voulons !

## Conclusion et ouverture

Dans cette section dédiée à l'algorithme de prédiction de l'instrument joué, nous avons discuté de la logique globale de son fonctionnement que nous pouvons résumer ici : nous partons d'une base de données de fichiers sons, puis nous extrayons des caractéristiques pour chacun de ces sons, puis nous fournissons ces caractéristiques à un algorithme capable d'apprendre afin qu'il s'entraîne à prédire l'instrument joué dans un fichier son, puis nous testons cet algorithme une fois entraîné.

Nous avons ensuite fait une présentation de notre implémentation de l'algorithme en vous montrant la structure du projet ainsi que les utilités de chaque fichier.

Si vous souhaitez tester l'algorithme sur votre ordinateur, vous pouvez suivre le petit tutoriel proposé en annexe [1](#).

## 3.4 Programmation de l'accordeur

Le programme de l'accordeur comporte plusieurs parties que nous allons décrire. En effet, nous avons créé de multiples fonctions pour l'écran LCD, le micro, la transformée de Fourier rapide... Nous avons ensuite utilisé ces fonctions dans notre programme final pour faire fonctionner notre accordeur.

### Interaction avec le micro

La première étape pour le fonctionnement de l'accordeur est intuitive : il est nécessaire de capter un son à partir du microphone. Pour déterminer le son reçu durant un certain laps de temps, il suffit d'enregistrer un nombre de valeurs que renvoie le micro à une certaine fréquence. Dans notre cas, nous savons qu'il nous faut `SAMPLE SIZE` valeurs, qui est la taille par défaut



de tous nos tableaux, et nous relevons la valeur que renvoie le micro à une période définie. Le temps d'enregistrement est donc `SAMPLE SIZE` multiplié par cette période. Il suffit de mettre cette fonction dans une boucle pour que le micro enregistre en continu.

Nous pouvons remarquer que plus la période définie est petite, plus il y aura de valeurs pour un laps de temps plus court, donc plus ce sera précis. Nous avons dû déterminer une période optimale qui est assez précise sans pour autant consommer trop de mémoire.

## Transformée de Fourier rapide

Nous entrons maintenant dans le coeur de notre algorithme. Nous avons obtenu les données du signal sonore grâce au microphone. Ce qui nous intéresse c'est la fréquence fondamentale de ce signal. Pour l'obtenir, la première étape est de calculer le spectre de fréquence du signal à l'aide de la transformée de Fourier discrète. L'algorithme standard implémentant cette transformée est le **Fast Fourier Transform** (FFT). Nous avons utilisé la bibliothèque `arduino.FFT` pour implémenter cet algorithme. Nous avons exploité la bibliothèque en créant une fonction TFD (Transformée de Fourier Discrète) que nous allons décrire. Commençons par la création de deux tableaux ayant la taille du nombre d'échantillons :

1. `vReal`, à valeurs réelles dans lequel nous stockerons les valeurs de l'échantillon sous 8bits.
2. `vImag`, qui représente une seconde composante de ces valeurs, leur partie imaginaire.

Nous stockons dans ce tableau des valeurs nulles que nous changerons par la suite.

Ces deux tableaux seront la base de nos calculs et seront modifiés tout au long du processus du FFT. Notre algorithme utilise trois fonctions pour ses calculs :

1. **FFT.Windowing** utilise une fonction de fenêtrage de type Hamming. Notre signal a une quantité finie de valeurs, ce qui entraîne en ses bords une discontinuité. La fonction de fenêtrage de type Hamming permet d'adoucir la transition du signal sur les bords (tendant ainsi vers zéro au début et à la fin), empêchant l'apparition d'harmoniques faussant nos calculs. Le signal se déformera de par cette fonction de fenêtrage, mais la Transformée de Fourier restera valable dans une approximation satisfaisante.
2. **FFT.Compute** effectue les calculs de la transformée de Fourier sur le signal modifié par la fonction de fenêtrage. Il modifie les deux tableaux créés précédemment (`Vreal` et `Vimag` qui ont été modifiés par la fonction précédente).
3. **FFT.ComplexToMagnitude** utilise les deux tableaux pour calculer le module de chaque nombre complexe (dont la partie réelle est à un certain indice `i` dans `Vreal`, et la partie imaginaire à l'indice `i` dans `Vimag`). `Vreal` est réécrit avec les modules résultants.

4. **FFT.MajorPeak** utilise le tableau *Vreal* et détermine la fréquence où l'on trouve le plus grand module. Cette fréquence est la fréquence fondamentale du signal, c'est-à-dire sa note. La transformée de Fourier est ainsi terminée.

## Analyse des résultats

Avant de commencer l'algorithme, nous créons un tableau *bonnesNotes*, où nous rentrons toutes les fréquences adéquates pour l'instrument que nous souhaitons accorder. Par exemple, le tableau *bonnesNotes* destiné à la guitare comprend six fréquences correspondant aux fréquences des six cordes de l'instrument. Nous récupérons ensuite la fréquence trouvée précédemment via la transformée de Fourier, puis nous rentrons cette valeur dans la fonction *ajust*. Cette fonction reçoit la fréquence émise par l'instrument, cette fréquence étant potentiellement fausse, et il renvoie la fréquence de la bonne note associée, son indice dans le tableau *bonnesNotes* ainsi que l'information servant à accorder l'instrument (c'est-à-dire s'il faut jouer plus aigu ou plus grave). Plusieurs fonctions sont nécessaires pour parvenir à ce résultat. Nous allons vous les détailler dans cette partie. Tout d'abord, nous rentrons la fréquence de la mauvaise note dans la fonction *trouveBonneNote*. Dans cette fonction, nous créons un tableau vide appelé *ecarts* et ayant la même taille que *bonnesNotes*. Nous créons ensuite une boucle qui met dans chaque case de *ecarts* la différence en valeur absolue entre la mauvaise note et chaque notes du tableau *bonnesNotes*. Nous renvoyons ensuite le tableau *ecarts*. Nous passons le tableau *ecarts* en argument de la fonction *indiceEtNote*. Tout d'abord, nous créons une variable *minimum* dans laquelle nous mettons par défaut la première valeur du tableau *ecarts*. Nous mettons également dans la variable *indice* le chiffre 0. Ces manipulations servent juste à initialiser ces deux variables, mais leurs valeurs évolueront au cours de la fonction. Ensuite, nous comparons, dans une boucle, la variable *minimum* avec chaque valeur de *ecarts*. Si *minimum* est inférieur à *ecarts*, alors nous ne changeons rien et nous gardons la variable *minimum* intacte. Si, au contraire, la valeur de *ecarts* est inférieure à celle de *minimum*, nous remplaçons *minimum* par cette valeur. Si cette condition est validée, nous mettons également dans la variable *indice* la position de la note dans le tableau. Cette fonction nous permet donc d'obtenir la valeur minimale du tableau *ecarts* ainsi que son indice dans le tableau. Pour résumer, nous rentrons la fréquence de la note jouée dans une fonction qui utilise elle-même une deuxième fonction, et nous obtenons en sortie la bonne note la plus proche ainsi que son indice. Ensuite, nous utilisons ces données pour exploiter la fonction *ajust* qui nous donne l'information nécessaire pour accorder notre instrument. Nous récupérons la mauvaise note *mn* et la bonne note associée *bn*, et nous les comparons. Nous créons une condition avec trois possibilités :

- *mn* est supérieure à *bn* : La note est trop aiguë et il est donc nécessaire de diminuer la

fréquence. Nous rentrons donc dans une nouvelle variable nommée *ajustement* la valeur -1.

- *mn* est inférieure à *bn* : La note est trop grave. Il faut donc augmenter la fréquence. Nous rentrons donc la valeur 1 dans *ajustement*.
- *mn* est égale à *bn* : L'instrument est bien accordé. Nous rentrons donc la valeur 0 dans *ajustement*.

Nous sommes maintenant à la fin de la fonction *ajust* qui renvoie donc les trois données qui nous intéressent, c'est-à-dire les valeurs *bn*, *indice* et *ajustement*.

## Interaction avec l'écran LCD

Pour finaliser la construction de notre accordeur, il a fallu trouver un moyen pratique pour transmettre le résultat à l'utilisateur. Une fois le son reçu par le microphone et les données analysées, nous avons utilisé un dernier composant, l'écran LCD. Pour cela, nous avons utilisé la bibliothèque LiquidCrystal, facilitant grandement son utilisation.

Afin de préparer l'utilisation de l'écran LCD, nous introduisons une variable de type LiquidCrystal comme ceci : LiquidCrystal lcd(7,8,9,10,11,12). Nous avons ici créé la variable lcd, représentant l'écran branché aux différents pins du microprocesseur indiqués. La variable lcd nous permettra de régler l'écran tout au long du programme, grâce à différentes fonctions.

C'est dans la fonction écran que tout se déroule ;

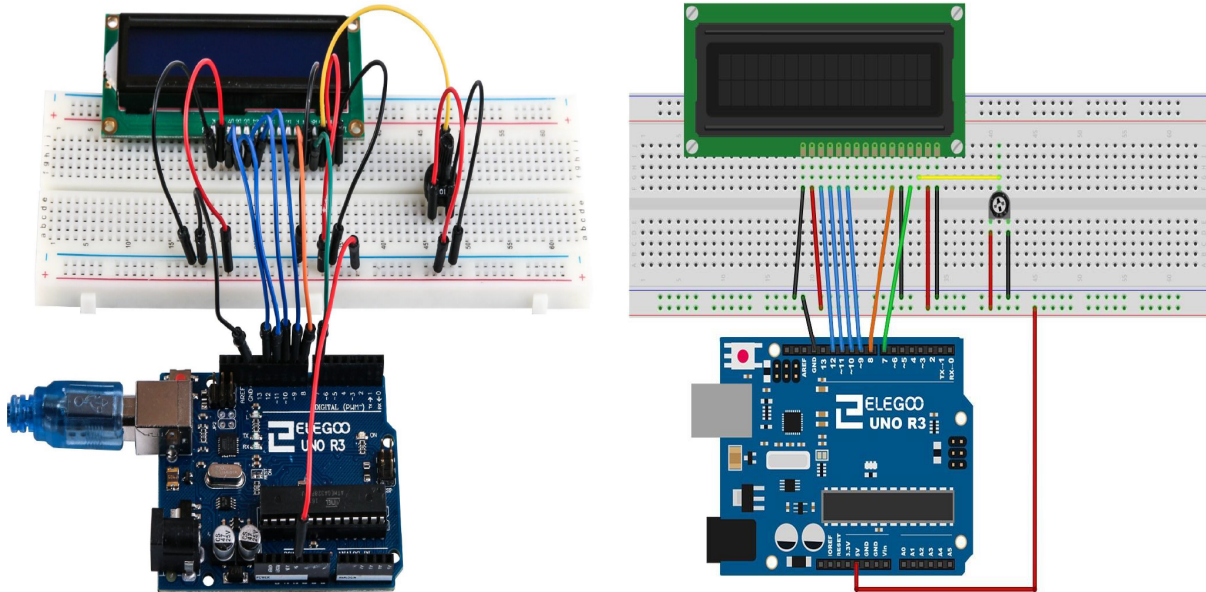
1. lcd.clear() réinitialise les écritures de l'écran et supprime ainsi tous les affichages.
2. Par la suite, l'écran affiche l'instruction nécessaire en fonction de la valeur de la variable "ajustement". L'écran affichera ainsi "Jouez + grave", "Jouez + aigu", ou "Note Bonne!".
3. lcd.setCursor(0,1) place "le curseur" sur la ligne 0 (la première), à la colonne 1 (la deuxième).
4. lcd.print(notes[indice]) affiche la note se rapprochant le plus de la note jouée (captée par le microphone). L'utilisateur saura ainsi quelle note il joue, et comment accorder son instrument pour que celle-ci soit juste.

Comme toutes les autres fonctions, celle-ci sera répétée indéfiniment dans le programme tant que l'accordeur fonctionnera. Elle permettra d'actualiser en continu les données affichées sur l'écran LCD.

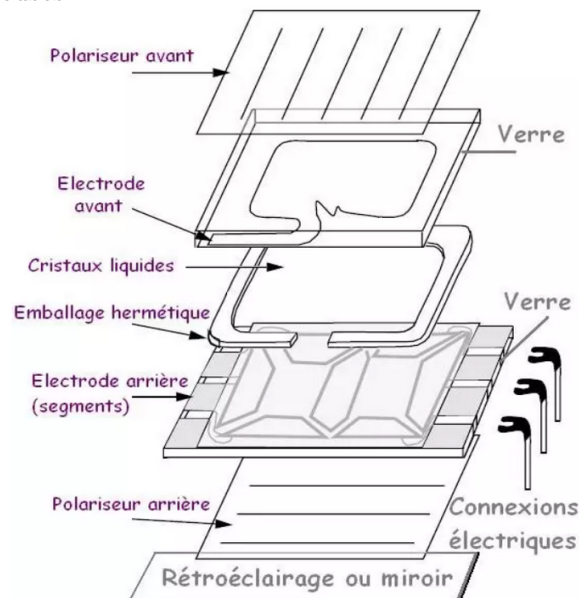
### 3.5 Les composants de l'accordeur

#### L'écran LCD

Comme vous l'avez compris, nous avons besoin pour notre accordeur d'un écran LCD. Nous tâcherons dans cette partie de décrire les différents besoins du composant pour fonctionner.



L'écran LCD (Liquid Crystal Display) se compose d'un rétro-éclairage (ensemble de néons) et d'une couche de pixels. Les néons sont la source lumineuse de l'écran, et envoient constamment une lumière blanche. L'information électrique est reçue par les pixels, permettant la transmission ou non des ondes lumineuses.



En effet, un champ électrique modifie l'orientation des cristaux liquides placés entre les deux polariseurs (voir schéma). Ces cristaux possèdent une propriété très intéressante, puisqu'ils

peuvent changer la polarisation des ondes lumineuses selon leur orientation. Vous l'aurez compris, la modification du champ électrique modifie la polarisation de la lumière, et donc l'intensité lumineuse traversant le polariseur avant. On remarque aussi que sans ces cristaux liquides, la lumière ne passerait pas du tout puisque le polariseur arrière est opposé au polariseur avant.

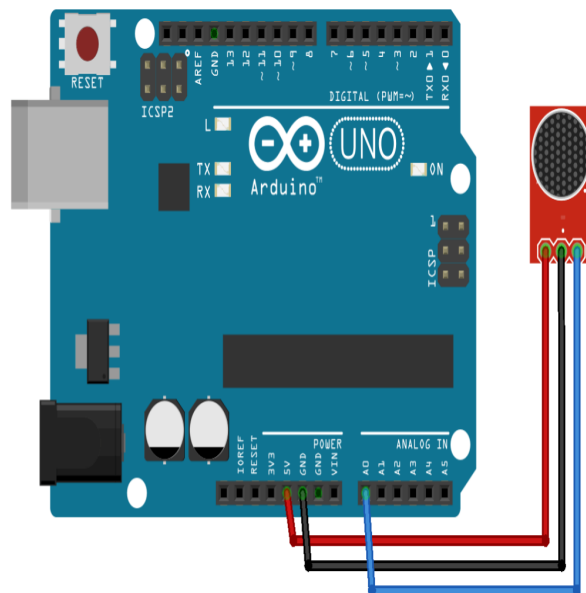
Bref, notre écran aura besoin de signaux électriques très précis pour permettre un affichage adapté. Notre écran renverra de la lumière bleue en l'absence d'un signal électrique, et de la lumière blanche s'il y a présence d'un signal électrique. Il est aussi à noter qu'on branche un potentiomètre en série avec l'afficheur pour venir ajuster le contraste de notre écran, et de ce fait, rendre l'affichage le plus lisible possible.

Heureusement, le montage électronique de l'écran LCD simplifie fortement son utilisation et ses réglages : seulement 12 branchements sont nécessaires (comparé au nombre de pixels, ceci est très peu!) : alimentation, connexion à la masse, contrôle de l'intensité du rétro-éclairage, réglage du contraste, 4 pins pour la transmission des données, contrôle du registre de mémoire, et un dernier pin pour lire ou écrire l'information.

Il en est de même au niveau de la programmation. La bibliothèque LiquidCrystal a grandement simplifié la mise en place des instructions données au microprocesseur Arduino.

## Le microphone

Comme vous le savez, notre accordeur ne pouvait fonctionner sans micro. D'une manière analogue à l'écran LCD, nous tâcherons d'explicitier son principe de fonctionnement, ainsi que ses besoins en branchements et signaux électriques.



Le microphone est composé d'une plaque mobile reliée à une bobine ainsi que d'une plaque fixe, un aimant. Les vibrations de l'air mettent en mouvement la bobine, entraînant une induction

électromagnétique. En réalité, on peut voir cela comme un circuit fermé en mouvement dans un champ magnétique, créant ainsi un courant. L'avantage de ce mécanisme est qu'on peut quantifier le son joué : la variation de l'amplitude modifie le courant induit et donc l'information transmise au microprocesseur. C'est sur ce principe de fonctionnement que se base notre microphone.

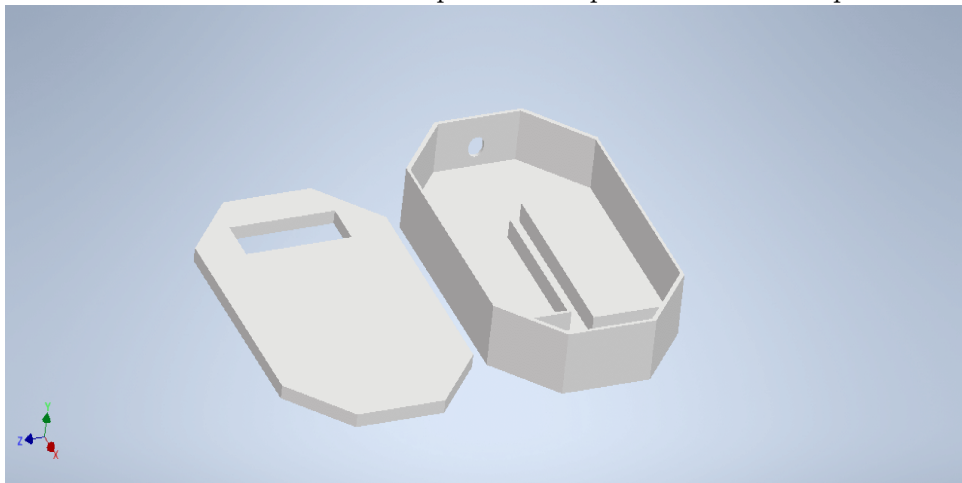
Les branchements du microphone sont très simples : une tension d'alimentation (5V), une borne reliée à la masse, ainsi qu'un pin permettant la transmission du signal au microprocesseur. Un potentiomètre intégré nous a permis de régler sa sensibilité.

Pour récupérer le signal capté, les instructions transmises au microprocesseur Arduino étaient assez simples, et n'ont pas nécessité de bibliothèque.

## Le boîtier

Évidemment un accordeur doit être un outil mobile, que l'on peut transporter où l'on veut. Donc il a fallu créer un boîtier unique qui permet de protéger les composants des transports. De plus cela permet d'avoir un objet plus agréable et simple à utiliser.

Une fois que nous avons fini toute la partie électronique, nous avons pu le modéliser avec le logiciel "AutoDesk Inventor". Pour ensuite pouvoir l'imprimer avec une imprimante 3D.



## 3.6 L'assemblage

Une fois que tout était terminé, que toutes les fonctions sont écrites et fonctionnent séparément, il a fallu toutes les assembler pour que notre accordeur fonctionne parfaitement. Intuitivement et grâce à l'expérience de certains membres de l'équipe, toutes nos fonctions étaient facilement reliables et tout fut plutôt simple à harmoniser. Il a aussi fallu que nous connectons tous les composants à la même carte arduino, ce qui était aussi trivial au vu des circuits vraiment simples.

Mais, une partie de l'assemblage fut plus compliquée à mettre en place. La modélisation du

boîtier n'a pas été assez réfléchi, il y a un souci d'ergonomie dans le boîtier. Par soucis de simplicité, nous avons modélisé le boîtier en deux parties comme expliqué juste au-dessus, mais nous n'avons prévu aucune attache permettant de lier ces deux parties. Nous avons donc dû utiliser de la colle et de la pâte de fixation.

De plus, nous n'avons pas réfléchi à l'alimentation, nous n'avons prévu aucune sortie sur le boîtier pour alimenter la carte Arduino. La seule solution possible est donc d'utiliser une pile, mais le processus pour remplacer cette pile lorsqu'elle est vide est assez fastidieux, car il faut séparer les deux parties que nous avons fixé précédemment.

Mis à part ce détail d'ergonomie, l'accordeur est fonctionnel et prêt à l'usage, ce qui était notre objectif principal.



## 4 Conclusion

Dans ce rapport, nous avons établi la problématique de notre projet ainsi que notre plan pour y répondre. Nous sommes ensuite rentrés dans les détails concernant les deux créations du projet : l'accordeur et l'algorithme de prédiction de l'instrument joué. Ces deux créations représentent une belle réussite de notre expérience durant le long de ces cinq mois. Mais bien sûr, nous pourrions aller beaucoup plus loin avec plus de temps, nous pourrions améliorer notre accordeur, en lui ajoutant plus d'instruments à cordes, autres que notre guitare et notre ukulélé. Également, nous pourrions perfectionner notre algorithme de prédiction en ajoutant plus de données d'entraînement, plus d'instruments, mais aussi en utilisant un réseau de neurones permettant une sophistication de cet algorithme. Dans la prochaine section, nous discuterons des risques et besoins de notre projet, de comment nous nous sommes organisés durant le projet.

# Bibliographie

- [1] 3BLUE1BROWN. *Solving Wordle using information theory*. 6 fév. 2022. URL : <https://www.youtube.com/watch?v=v68zYyaEmEA> (visité le 13/05/2022).
- [2] ACOUSTICIAN. *Comfortably Numb Solo - Pink Floyd - Acoustic Guitar Cover*. 8 oct. 2016. URL : <https://www.youtube.com/watch?v=MRqjOuhZsoY> (visité le 26/04/2022).
- [3] Jason BROWNLEE. *Evaluate the Performance of Machine Learning Algorithms in Python using Resampling*. Machine Learning Mastery. 22 mai 2016. URL : <https://machinelearningmastery.com/evaluate-performance-machine-learning-algorithms-python-using-resampling/> (visité le 14/05/2022).
- [4] *Decision tree learning*. In : *Wikipedia*. Page Version ID : 1087493830. 12 mai 2022. URL : [https://en.wikipedia.org/w/index.php?title=Decision\\_tree\\_learning&oldid=1087493830](https://en.wikipedia.org/w/index.php?title=Decision_tree_learning&oldid=1087493830) (visité le 13/05/2022).
- [5] *Energy (signal processing)*. In : *Wikipedia*. Page Version ID : 975039763. 26 août 2020. URL : [https://en.wikipedia.org/w/index.php?title=Energy\\_\(signal\\_processing\)&oldid=975039763](https://en.wikipedia.org/w/index.php?title=Energy_(signal_processing)&oldid=975039763) (visité le 15/05/2022).
- [6] *Entropie conditionnelle*. In : *Wikipédia*. Page Version ID : 170593916. 8 mai 2020. URL : [https://fr.wikipedia.org/w/index.php?title=Entropie\\_conditionnelle&oldid=170593916](https://fr.wikipedia.org/w/index.php?title=Entropie_conditionnelle&oldid=170593916) (visité le 14/05/2022).
- [7] *Entropy (information theory)*. In : *Wikipedia*. Page Version ID : 1084479990. 24 avr. 2022. URL : [https://en.wikipedia.org/w/index.php?title=Entropy\\_\(information\\_theory\)&oldid=1084479990](https://en.wikipedia.org/w/index.php?title=Entropy_(information_theory)&oldid=1084479990) (visité le 13/05/2022).
- [8] Homero ESMERALDO. *Answer to "Alternative to Shannon's entropy when probability equal to zero"*. Cross Validated. 25 oct. 2019. URL : <https://stats.stackexchange.com/a/433096> (visité le 14/05/2022).
- [9] *F1-score & F-beta score, compromis entre Precision et Recall en classification*. Kobia. 17 nov. 2021. URL : <https://kobia.fr/classification-metrics-f1-score/> (visité le 14/05/2022).



- [10] *Frequencies of Musical Notes*,  $A_4 = 440$  Hz. URL : <https://pages.mtu.edu/~suits/notefreqs.html> (visit  le 11/05/2022).
- [11] Bhuvaneswari GOPALAN. *What is Entropy and Information Gain ? How are they used to construct decision trees ?* Numpy Ninja. 10 d c. 2020. URL : <https://www.numpyninja.com/post/what-is-entropy-and-information-gain-how-are-they-used-to-construct-decision-trees> (visit  le 13/05/2022).
- [12] *Information gain in decision trees*. In : *Wikipedia*. Page Version ID : 1083057479. 16 avr. 2022. URL : [https://en.wikipedia.org/w/index.php?title=Information\\_gain\\_in\\_decision\\_trees&oldid=1083057479](https://en.wikipedia.org/w/index.php?title=Information_gain_in_decision_trees&oldid=1083057479) (visit  le 13/05/2022).
- [13] *librosa — librosa 0.9.1 documentation*. URL : <https://librosa.org/doc/latest/index.html> (visit  le 15/05/2022).
- [14] *Mel scale*. In : *Wikipedia*. Page Version ID : 1085283171. 29 avr. 2022. URL : [https://en.wikipedia.org/w/index.php?title=Mel\\_scale&oldid=1085283171](https://en.wikipedia.org/w/index.php?title=Mel_scale&oldid=1085283171) (visit  le 15/05/2022).
- [15] *MFCCs : Engineering features from sound - Life at Pex*. Pex. 7 ao t 2020. URL : <https://pex.com/blog/machine-learning-mfccs-engineering-features-from-sound/> (visit  le 12/04/2022).
- [16] Aditya MISHRA. *Metrics to Evaluate your Machine Learning Algorithm*. Medium. 28 mai 2020. URL : <https://towardsdatascience.com/metrics-to-evaluate-your-machine-learning-algorithm-f10ba6e38234> (visit  le 14/05/2022).
- [17] Dr Meinard MULLER. « Harmonic Percussive Source Separation ». In : (), p. 13.
- [18] Pratheeksha NAIR. *The dummy's guide to MFCC*. prathena. 27 juill. 2018. URL : <https://medium.com/prathena/the-dummys-guide-to-mfcc-aceab2450fd> (visit  le 12/04/2022).
- [19] *Note de musique*. In : *Wikip dia*. Page Version ID : 187406478. 24 oct. 2021. URL : [https://fr.wikipedia.org/w/index.php?title=Note\\_de\\_musique&oldid=187406478](https://fr.wikipedia.org/w/index.php?title=Note_de_musique&oldid=187406478) (visit  le 28/11/2021).
- [20] *Octave (musique)*. In : *Wikip dia*. Page Version ID : 191747275. 9 mars 2022. URL : [https://fr.wikipedia.org/w/index.php?title=Octave\\_\(musique\)&oldid=191747275](https://fr.wikipedia.org/w/index.php?title=Octave_(musique)&oldid=191747275) (visit  le 11/05/2022).
- [21] *Pitch, loudness and timbre. From Physclips*. URL : <https://www.animations.physics.unsw.edu.au/jw/sound-pitch-loudness-timbre.htm#sub4> (visit  le 22/01/2022).

- [22] *Principe de fonctionnement d'un écran LCD*. Astuces Pratiques. Section : Informatique, Electronique, high-tech, auto-moto, bois, maison et bricolage, diy, jardinage, santé, cuisine, finance et droit. 3 déc. 2012. URL : <https://www.astuces-pratiques.fr/high-tech/principe-de-fonctionnement-d-un-ecran-lcd> (visité le 15/05/2022).
- [23] Pierre SCHAEFFER. « 1/ HAUTEUR 2/ INTENSITE 3/ TIMBRE a/ nature de la source sonore b/ qualité du son ». In : (), p. 26.
- [24] *Scikit-Learn - Model Evaluation & Scoring Metrics*. URL : <https://coderzcolumn.com/tutorials/machine-learning/model-evaluation-scoring-metrics-scikit-learn-sklearn> (visité le 21/04/2022).
- [25] *scikit-learn : machine learning in Python — scikit-learn 1.1.0 documentation*. URL : <https://scikit-learn.org/stable/index.html> (visité le 15/05/2022).
- [26] *son : timbre et harmoniques. Cours pour débutants*. URL : [https://www.spirit-science.fr/doc\\_musique/Timbre.html](https://www.spirit-science.fr/doc_musique/Timbre.html) (visité le 28/11/2021).
- [27] *Sound samples*. Philharmonia. URL : <https://philharmonia.co.uk/resources/sound-samples/> (visité le 01/02/2022).
- [28] *Spectral density*. In : *Wikipedia*. Page Version ID : 1070903811. 9 fév. 2022. URL : [https://en.wikipedia.org/w/index.php?title=Spectral\\_density&oldid=1070903811](https://en.wikipedia.org/w/index.php?title=Spectral_density&oldid=1070903811) (visité le 15/05/2022).
- [29] XUKYO. *Utilisation d'un Microphone avec Arduino • AranaCorp*. AranaCorp. 2 juin 2020. URL : <https://www.aranacorp.com/fr/utilisation-dun-microphone-avec-arduino/> (visité le 15/05/2022).

## 5 Aspects gestion de projet

En tant que futurs ingénieurs, nous serons confrontés à beaucoup d'imprévus et de problèmes, auxquels nous devons être dans la capacité de les résoudre. Durant toute la durée de ce projet atypique mais aussi représentatif de la complexité de la collaboration nécessaire en équipe, notre organisation du travail n'a de cesse d'évoluer en fonction de tous les imprévus rencontrés. Rappelons que notre thème est « La musique et l'information numérique », sujet qui avait posé notre premier problème, soit de trouver l'objet d'étude de notre projet. Après des recherches sur ce sujet, et prenant en compte nos diverses envies et volontés à l'égard de certains potentiels projets, nous avons décidé d'étudier la caractérisation du timbre d'un instrument. Notre objet d'étude étant bien défini, nous avons mis en place un planning, comprenant les différentes tâches à réaliser sur toute la durée du projet, ainsi que les potentiels risques à survenir. Également, étant donné que nous avons formé notre groupe par affinité (et donc que nous connaissons les goûts de chacun), nous avons divisé notre groupe en deux silos assignés à un domaine spécifique de notre objet d'étude :

- **L'équipe « programmation »** : Notre projet étant essentiellement concentré sur la transformation d'un son en signal informatique, ainsi que l'écriture informatique (principalement python, mais aussi en c++) des programmes nécessaires à la réalisation de notre projet, il nous semblait indispensable d'avoir un nombre important de personnes. Nous comptons parmi les membres de cette équipe Yann, Dorian, Glen, Maxime et Antoine.
- **L'équipe « information et électronique »** : Notre étude requérait des aspects physiques liés au son et à sa transformation. Également, pour développer notre sujet, mais aussi l'étendre à un plus grand nombre de personnes lors de la conférence, nous avons formé une équipe de trois membres, s'assurant la réalisation des différentes manipulations électroniques nécessaires. Ce groupe est constitué d'Augustin, Arthur et Marin.

Nos objectifs finaux étaient de faire des programmes informatiques détectant les notes jouées par un instrument et de distinguer les timbres de deux instruments différents.

Nous avons pour cela réalisé un plan détaillé du déroulement de notre projet, mais nos attentes et prévisions se révélèrent très différentes de la réalité. En effet, entre incompréhension du sujet, baisse de motivation, informations nécessaires difficiles à trouver et manque de matériel, nous avons rencontré de nombreux problèmes au cours de ce projet, et ainsi dû ré-adapter notre stratégie de travail.

## 5.1 Déroulement des séances

Pour la bonne avancé de notre projet, nous nous sommes focalisés sur une approche agile : la stratégie du Time boxing. Plus précisément, une approche par incrémentations, où nous nous retrouvions tous les mardis après-midi (sur les temps de travaux disposés sur l'emploi du temps), travaillant sur des séances de temps variables, commençant parfois à 11h et pouvant se terminer à 18h30. Ces débuts de séances représentaient une mise en commun des recherches effectuées par les membres du groupe lors de la semaine passée. Nous sélectionnions alors les informations primordiales, apportées par chacun, à l'avancé de notre projet. Après cette réunion, nous définissions nos nouveaux objectifs pour la séance et nous allions directement travailler avec notre équipe assignée à un domaine spécifique, établi durant la phase de l'avant-projet (ces équipes qui varièrent au cours du projet). Ainsi, nous pouvions remarquer au sein d'une même salle plusieurs silos, constitués de 1 à 3 personnes.

Le travail en équipe était primordial durant ces séances. Nous pouvions toujours avoir un oeil attentif quant au travail en cours, si un membre de l'équipe ne voyait pas une erreur ou était distrait, il y avait toujours son partenaire qui pouvait veiller au bon déroulement de la tâche. Egalement, l'esprit d'équipe permettait de créer une atmosphère de convivialité et de confiance, nécessaire pour les étudiants afin de sentir à l'aise, permettant de travailler pleinement. Pour certaines tâches néanmoins, il se pouvait qu'il n'y avait qu'une personne, afin de limiter le temps de production du projet, en se répartissant plus de travail entre les différents membres de l'équipe InstrumentAll.

Lors de la fin de nos séances en groupe, nous faisons un rapide point sur l'avancé de chacun, nous mettions les diverses informations acquises dans notre projet. La dernière action que nous réalisions avant de terminer la journée, c'était d'assigner chaque membre à une tâche qu'il devait avancer durant la semaine, jusqu'à la prochaine séance.

Cette méthode de travail en présentiel se révéla parfaite lors de la fin de ce projet, car nous n'avons jamais modifier cette organisation et elle nous a permis d'atteindre nos objectif en toute sécurité, sans tension de groupe ou de désordre dans notre travail. Mais malgré ce bon système de travail en groupe, nous avons quand même rencontré différents problèmes.

## 5.2 Les problèmes rencontrés et résolus

Bien que nous avons réussi à choisir un objet d'étude avec l'accord de tout le groupe, notre premier, et même principal problème lors de ce projet était la compréhension de celui-ci. La musique et l'informatique sont des domaines très vastes, mais surtout très différents, où pouvions-nous voir le rapport entre les deux ? Nous disposions tous de quelques notions sur le thème du son (onde, fréquence, hauteur, timbre...), grâce à nos connaissances acquises au programme scolaire

dédié au lycée. Mais il s'avérait que ces connaissances ne nous permettaient pas d'avancer sur ce projet : il fallait trouver une corrélation entre l'étude du son et l'information numérique.

C'est en cela que rentre la notion de transformée de Fourier, dont certains dans le groupe ne comprenait pas après le premier cours magistral sur notre thème d'étude. Il faut savoir que la moitié de notre équipe n'a pas d'expérience avec l'informatique, qui est apparue seulement lors du premier semestre de notre parcours d'ingénieurs. Ainsi, le fait d'entreprendre un programme informatique très complexe avec cette transformée se révéla un très grand défi, autant pour le groupe qui ne la comprenait que pour l'autre partie de l'équipe, qui devait ainsi expliquer son utilité dans le projet. Pour remédier à ce problème, lors d'une de nos séances en équipe le mardi après-midi, certains élèves du groupe ont mis de côté l'avancé du projet afin de réaliser un cours sur la transformée de Fourier et son utilité, permettant aux élèves dans l'incompréhension d'avoir les bases nécessaires pour appréhender notre programme informatique. Par la suite, nous avons eu une séance de travaux dirigés sur cette transformée, permettant de consolider les acquis des étudiants en difficulté.

Malgré ce problème résolu concernant cette loi mathématique et l'avancée de la retranscription d'un son pour le numériser, il y avait toujours une séparation dans notre groupe de travail : celui qui comprenait la direction dans laquelle on allait, jouant avec l'informatique, et celui qui se sentait de moins en moins investi dans le projet.

Ne pouvant n'y contourner, n'y ignorer ce problème, nous avons dû l'affronter, sinon une partie du groupe ne comprendrait pas tout le reste du projet, perdant ainsi l'objectif de cette discipline. Ainsi, après la séance avec John Kingston durant la semaine 8 (22 février 2022), nous avons posé notre 2ème jalon.

Durant cette réunion, nous avons fait un bilan de l'avancé de notre projet. Constatant que la priorité à cette étape du projet était consacrée à de la programmation et de l'électronique, nous avons reconstitué les silos, ainsi que les objectifs de ceux-ci. Nous organisons notre équipe InstrumentAll sous la forme ci-dessous :

- Le silo « programme transformée de Fourier » : finir le programme exploitant ce concept mathématique, constitué de Maxime et d'Augustin ;
- L'équipe « algorithme bonne note » : commencer l'écriture du programme permettant d'indiquer s'il faut jouer plus grave ou plus aigu afin d'être accordé, comprenant Arthur et Marin ;
- Yann qui s'occupe de terminer le programme de reconnaissance de timbre qu'il avait débuté.
- Antoine était assigné à réaliser le programme qui renvoyait la note jouée.
- Glen qui s'occupe de l'aspect électronique afin de pouvoir relier l'information de nos

programmes de la carte Arduino à l'écran d'affichage ;

— Dorian assurant le branchement électronique du micro à notre dispositif.

Le problème concernant cette disparité diminua sur les semaines suivantes, mais nous la ressentions toujours. Ainsi, lorsque nous avions des activités hors du domaine informatique, nous les assimilions aux personnes en difficulté afin que ceux-ci puissent contribuer au projet, sans se sentir délaissé. Ce procédé était une excellente solution qui se prolongea jusqu'à la "fin théorique" de notre projet.

Le 12 avril 2022, marqua l'accomplissement de tous nos objectifs : on avait réussi à fabriquer un accordeur à l'aide de divers programmes informatiques. Mais en réalisant qu'il nous restait encore un peu plus d'un mois avant la conférence Peip, nous avons posé ce même jour notre 3ème jalon. Soit nous décidions d'en rester là et de commencer le rapport explicatif de notre projet. Soit nous étendions notre sujet en travaillant sur de nouvelles idées. C'est ainsi que nous choisissons cette deuxième option et décidâmes de faire un nouveau programme, permettant de reconnaître un instrument.

Afin d'élaborer ce nouveau objectif, nous avons fait un bilan de nos ressources, aussi bien matérielles que personnelles. Personne dans notre équipe n'était musicien, posant le problème suivant : nous ne pouvions pas jouer d'un instrument afin que l'on puisse initialiser notre programme de reconnaissance. Nous ne disposions seulement que d'une guitare et d'une clarinette. Pour faire face à ce nouveau problème, nous avons pensé et appliqué deux solutions en parallèle. Durant les dernières semaines de ce projet, nous avons eu l'aide de deux personnes, deux guitaristes de Polytech. En effet, Mme Pérennou et Pierre Teixeira nous ont accordé de leur temps afin de jouer de la guitare pendant que nous enregistrions ces accords pour notre programme. En parallèle, beaucoup de membres du groupe s'entraînaient à apprendre à jouer de la guitare, ainsi que de la clarinette. Trois à quatre semaines après ce 3ème jalon, nous avions assez de compétences pour jouer de ces instruments, et nous pûmes terminer notre dernier programme. Cette partie de travail pu vraiment représenter notre thème d'étude car bien que depuis le début du projet, nous ne faisions que des sciences, nous avons également tous touché au domaine artistique. Nous avons ainsi pu comprendre toute la complexité de la musique, sous différents angles.

Également, l'une des questions que nous nous sommes posés concernait le rendu final de notre projet lors de la conférence Peip. Qu'allait-on présenter de visuelle, mis à part notre programme informatique, au public afin que notre sujet le passionne ?

Lorsque ce problème est apparu dans les esprits de notre groupe de travail, durant la séance de la semaine 4 (25 janvier 2022), nous étions en pleine programmation afin de transformer un son en code informatique. Nous posions donc notre 1er jalon afin de réfléchir en équipe. Nous

voulions un objet physique répondant à notre problématique, cherchant ce qui caractérise le timbre d'un instrument. Il nous fallut peu de temps afin de trouver notre solution. En effet, la séance même, la réponse nous paraissait évidente : fabriquer un accordeur. Ce petit objet, représentant un élément connu du monde de la musique par tout le monde, permettrait de donner envie au public d'en découvrir plus à son sujet, sur son fonctionnement. Nous réorganisons nos recherches et le travail de chacun, pour étendre notre sujet, et créer deux programmes : un reconnaissant une "bonne note" joué par un instrument, et un autre permettant de dire s'il faut baisser ou augmenter la fréquence de cette note. Cela représentait un travail puisant encore dans le domaine informatique, amenant comme nous l'avons dit précédemment une augmentation d'un sentiment de séparation au sein de notre équipe InstrumentAll.

### 5.3 Comparaison prévisions/réalité

Bien que nous ayons essayé d'anticiper tous les types de problèmes et aléas qui pourraient survenir durant la phase de projet, nous remarquons que cette phase d'avant projet était essentiel car sans celle-ci, nous ne nous serions pas préparés à affronter ces problèmes, et n'aurions pas eu le temps d'en faire face à d'autres que nous ne pensions pas.

Le premier risque auquel nous avons eu peur d'être confronté était un reconfinement dû au Covid-19. Nous avons déjà eu deux confinements depuis 2020, et sachant que le travail en distanciel était très difficile, notamment en équipe, nous savions que notre projet serait gravement impacté. Heureusement, nous avons échappé à tous ces problèmes et toutes nos réunions pouvaient se passer sur le site de Polytech. En prenant du recul, il aurait été quasiment impossible de réaliser notre projet car celui-ci nécessitait un partage de connaissances (tels que pour notre programme de reconnaissance d'instrument, où nous n'aurions pas eu de musiciens) et un partage de matériels (notre accordeur nécessitait les composants d'Augustin, le micro de Dorian, et les divers programmes informatiques des autres membres).

Nous avons également pris en compte le risque de priorité aux devoirs surveillés du lundi. En effet, même si ce problème ne risquait pas de nous impacter, peut-être que nous aurions privilégié un travail pour les futurs devoirs des autres matières, en délaissant la conférence Peip. Etant donné que nous avons prévu d'organiser nos réunions le mardi et que nous sommes organisé dans nos révisions, nous pouvions avancer sereinement notre projet. Il en était de même pour le risque lié à la disposition d'outils numériques. Nous avons pu faire des recherches efficaces concernant divers éléments informatiques et nous disposions d'un professeur en cas de besoin. Egalement, nous disposions tous d'ordinateurs portables afin de faire nos codes informatiques et s'il fallait utiliser un logiciel plus avancé, nous utiliserions l'un des ordinateurs connectés au réseau de Polytech, ce que nous avons fait lors de la réalisation en 3 dimensions de notre

accordeur. Malgré tous ces risques mineurs que nous avons évité, il y avait un risque qui nous inquiétait particulièrement : “l’effet 42”.

“L’effet 42” représente la baisse de motivation de certains membres de l’équipe. Comme nous l’avons dit, notre groupe de travail est aussi un groupe d’amis. Ainsi, nous savions que certaines personnes n’appréciaient pas l’informatique, et que ceux-ci seraient sûrement les plus désintéressés par l’avancé du projet. Pour l’anticiper, nous avons prévu de donner les tâches ne demandant pas de connaissances en informatique à ces membres de l’équipe, ce que nous avons fait en réorganisant nos silos tout au long de ces 5 mois d’expériences.

Tous les types de problèmes que nous rencontrions étaient anticipés jusqu’à la “fin initiale” de notre projet. Cette étape représentait un nouvel aléa où nous devions faire un choix. Après avoir décidé de continuer notre projet, nous nous sommes rendu compte d’un événement inattendu : il fallait apprendre à jouer d’un instrument. Nous sommes des scientifiques et le domaine artistique nous est inconnu, mais étant donné que nous possédons la principale composante qui fait de nous ce que nous sommes, la curiosité, nous nous sommes tous attardés à étudier et tester des accords sur la guitare et quelques notes sur l’accordeur, et réussi à jouer le minimum nécessaire pour notre algorithme au bout de quelques semaines.

Nous nous rendons compte que notre projet, et l’organisation de celui-ci différerait quelque peu de notre idée prévisionnelle, comme nous pouvons le voir sur la figure. Cette expérience en équipe nous fait réaliser la complexité d’avoir une bonne organisation et à l’avenir, s’il fallait ajouter une chose à notre organisation, ce serait de tenir un journal de bord, avec tous les jours de travail regroupant les tâches effectuées.



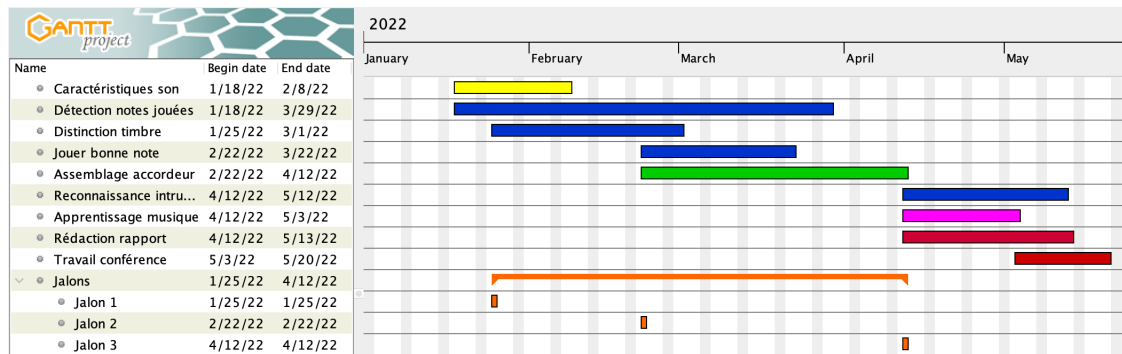


FIGURE 3 – Diagramme Gantt de ce qui a effectivement été fait durant le projet

Légende :

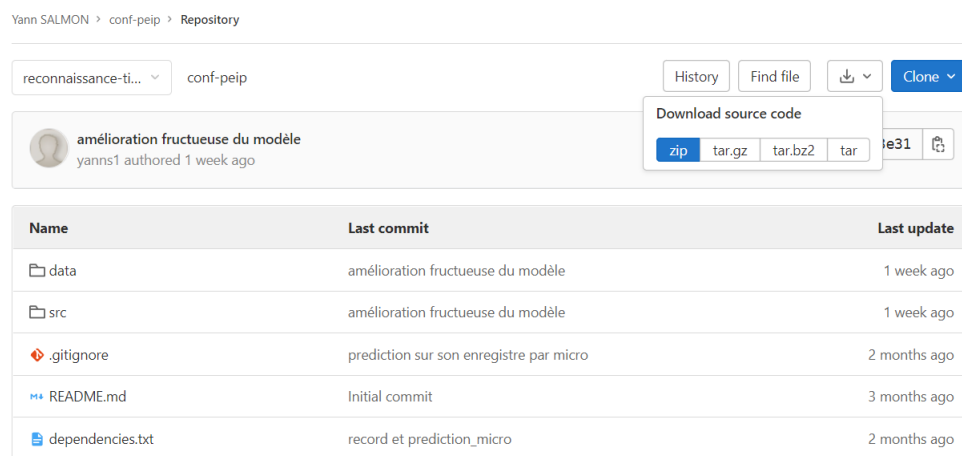
- En **bleu** : Tâches de l'équipe « programmation »
- En **jaune** : Tâches de l'équipe « information »
- En **vert** : Tâches de l'équipe « électronique »
- En **violet** : Tâches de l'équipe « artistique »
- En **rouge** : Rendu d'un livrable (rapport ou conférence)
- En **orange** : Jalons

# Annexes

# 1 Prise en main : comment VOUS pouvez tester l'algorithme

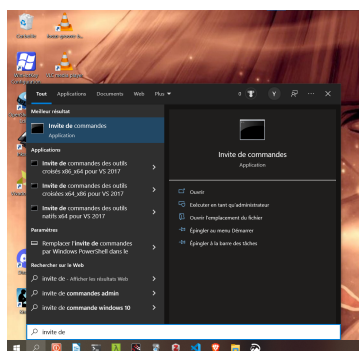
La base de données n'est pas disponible sur Gitlab, donc vous ne pourrez pas re-calculer la matrice des caractéristiques et ré-entraîner l'algorithme sur ces données. Cependant, vous avez accès aux fichiers .pickle dans lesquels sont sérialisés la matrice et l'algorithme final entraîné. Ils sont donc prêts à l'emploi ! Nous allons vous expliquer de A à Z comment enregistrer un son et obtenir la prédiction de l'algorithme pour celui-ci.

Tout d'abord, assurez-vous que vous avez téléchargé les codes. Si ce n'est pas déjà fait, cliquez sur [ce lien](#) (vous devrez être connecté à Gitlab avec votre compte université), puis cliquez sur l'icône téléchargement, puis sûr .zip en bleu.

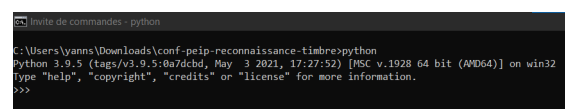


Cela va ouvrir votre explorateur de fichier et vous invitez à télécharger le dossier du projet. Placez le dossier où cela vous arrange, mais gardez en tête là où vous l'avez mis. Vous pouvez ensuite en extraire le contenu.

Deuxième étape : assurez-vous d'avoir Python installé sur votre machine. Pour vérifier si c'est le cas, ouvrez une invite de commandes et tapez "python". Vous devriez obtenir ce qu'il y a sur l'image de droite.



(a) Ouverture de l'invite de commandes par défaut de Windows

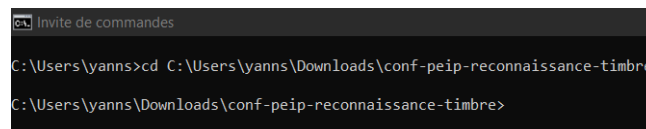


(b) Démarrage de Python

Si la commande "python" n'est pas reconnu, c'est que vous n'avez pas Python. Dans ce cas, suivez [ce lien](#) pour le télécharger. Il est aussi possible que vous ayez Python, mais que la version installée sur votre machine soit trop ancienne. Vous pouvez voir la version de votre Python au début de la première ligne dans l'invite de commandes. La mienne est 3.9.5 comme vous pouvez le voir sur l'image de droite. Si votre version commence par un 1 ou un 2, réinstallez Python en suivant le lien ci-dessus (l'ancien Python sera automatiquement désinstallé/mis-à-jour).

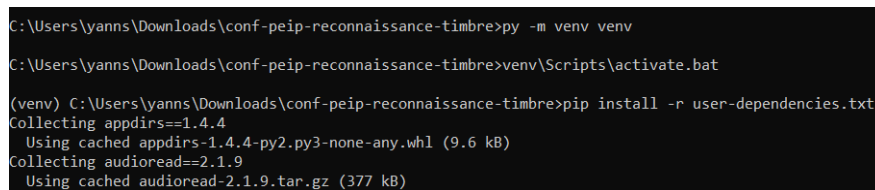
La suite des explications seront spécifiques au système d'exploitation Windows (Windows 10 précisément) car les développeurs du projet ont fonctionné avec celui-ci et sont peu familiers avec d'autres systèmes tels que macOS ou Linux. Cependant, les instructions devraient être facilement adaptables par l'utilisateur qui fonctionne avec un autre système d'exploitation.

L'objectif maintenant est d'exécuter le fichier *record\_then\_predict*, tout simplement. Ouvrez de nouveau l'invite de commandes (ou tapez "Ctrl+Z" si vous êtes au stade de l'image (b)), puis déplacez-vous jusqu'au dossier *C:\...\conf-peip-reconnaissance-timbre*. Par exemple :



```
Invite de commandes
C:\Users\yanns>cd C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre
C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre>
```

Maintenant, tapez les commandes comme sûr l'image ci-dessous :



```
C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre>py -m venv venv
C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre>venv\Scripts\activate.bat
(venv) C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre>pip install -r user-dependencies.txt
Collecting appdirs==1.4.4
  Using cached appdirs-1.4.4-py2.py3-none-any.whl (9.6 kB)
Collecting audioread==2.1.9
  Using cached audioread-2.1.9.tar.gz (377 kB)
```

La première ligne sert à créer ce qu'on appelle un *environnement virtuel* (*virtual environment* en anglais, d'où l'abréviation "venv") en Python. C'est un dossier dans lequel sera stocké toutes les librairies nécessaires au projet, librairies que l'on installe justement à la troisième ligne. Cela permet de confiner les librairies (et leurs versions) au projet. On dit que l'on installe les librairies *localement*. Par défaut, les librairies s'installent globalement, c'est-à-dire aux alentours de l'endroit où Python est installé sur votre ordinateur. Le problème étant que si vous avez deux projets qui demandent une même librairie mais de versions différentes, vous êtes coincés. Imaginez de plus les problèmes de compatibilité de version qui attendent un groupe de développeurs qui travaillent sur un même projet, mais où chacun utilise ses librairies globales et leurs versions spécifiques... C'est pourquoi les environnements virtuels sont si utiles. La deuxième ligne permet juste d'activer l'environnement virtuel. Notez aussi que vous aurez beaucoup plus de lignes du type de celles que l'on voit en bas de l'image ("Collecting ..."). Ce sont les librairies qui s'installent. Une fois cela terminé, tapez "pipwin install pyaudio".

```
(venv) C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre>pipwin install pyaudio
Package 'pyaudio' found in cache
Downloading package . . .
https://download.lfd.uci.edu/pythonlibs/x6hvwk7i/PyAudio-0.2.11-cp310-cp310-win_amd64.whl
PyAudio-0.2.11-cp310-cp310-win_amd64.whl
[*] 111 kB / 111 kB @ 37 kB/s [#####] [100%, 0s left]
Processing c:\users\yanns\pipwin\pyaudio-0.2.11-cp310-cp310-win_amd64.whl
Installing collected packages: PyAudio
Successfully installed PyAudio-0.2.11
WARNING: You are using pip version 21.2.4; however, version 22.1 is available.
You should consider upgrading via the 'C:\Users\yanns\Downloads\conf-peip-reconnaissance-timbre\venv\Scripts\python.exe -m pip install --upgrade pip' command.
```

Il serait un peu long d'expliquer ce qui se passe ici. Comprenez juste que "pyaudio" est une librairie réticente à une installation standard, et que la commande ci-dessus est une solution temporaire.

Nous sommes bons ! Nous pouvons enfin lancer le fichier qui nous intéresse :

```
Entrez le nom du fichier .wav dans lequel sera stocké le son que vous vous apprêtez à enregistrer: le-nom-que-vous-voulez
Entrez la durée de l'enregistrement, en secondes (entier ou nombre à virgule): 10
Tapez sur 'Enter' pour lancer l'enregistrement.
```

Entrez les informations demandées et tapez "Enter" pour passer à la ligne suivante. Une fois que vous êtes à la troisième ligne, attendez avant de lancer l'enregistrement. Sortez votre guitare, clarinette, flûte ou piano, et préparez-vous à jouer. Si nous n'en avez pas, sortez votre téléphone et cherchez une musique où l'on entend (uniquement) l'un de ces instruments. Dernière petite chose, branchez un micro de (relativement bonne) qualité à votre ordinateur, car en général, les micros par défaut d'ordinateurs portables sont de mauvaise qualité. Une fois prêt, tapez sur "Enter" et joué ! Vous devriez observer quelque chose comme ça.

```
Entrez le nom du fichier .wav dans lequel sera stocké le son que vous vous apprêtez à enregistrer: le-nom-que-vous-voulez
Entrez la durée de l'enregistrement, en secondes (entier ou nombre à virgule): 10
Tapez sur 'Enter' pour lancer l'enregistrement.
recording...
finished recording

Résultat pour le-nom-que-vous-voulez.wav
Résultat de la prédiction: guitar
Pourcentages pour chaque instrument:
- guitar: 0.55
- piano: 0.1
- flute: 0.3
- clarinet: 0.05
```

## 2 Évaluer la performance d'un algorithme de prédiction

Dans cet annexe, nous allons énumérer quelques méthodes (du plus simple au plus sophistiqué) permettant d'évaluer la performance d'un algorithme de classification <sup>18</sup>.

### Première méthode : Diviser la base de données en deux avec une partie « entraînement » et une partie « test »

La première stratégie consiste à diviser la base de données en deux, avec une partie des données dédiés à l'entraînement et une autre dédiée au test de l'algorithme. En effet, il est intéressant de tester un algorithme sur les données sur lesquelles il s'est entraîné, étant donné que nous cherchons à évaluer sa capacité à généraliser. Si nous faisons ça, un algorithme qui retiendrait simplement les résultats aperçus durant l'entraînement aurait 100% de bonnes prédictions. Il aurait cependant aucune bonne prédiction pour de nouvelles données. En général, nous dédions deux tiers des données à l'entraînement (il en faut tout de même un certain nombre pour obtenir des résultats intéressants) et un tiers au test.

### La métrique de base : la précision

Une fois l'algorithme entraîné et les prédictions sur les données de test récupérées, nous aimerions quantifier la performance sur les tests d'une certaine manière. L'idée naturelle est de regarder la proportion de bonnes prédictions parmi toutes les prédictions faites. Nous définissons ainsi la **précision générale** de l'algorithme :

$$\text{précision générale} = \frac{\text{nombre de bonnes prédictions}}{\text{nombre total de prédictions}}$$

Un premier défaut de la précision est qu'elle est susceptible de grandement varier en fonction de la manière avec laquelle vous avez divisé votre base de données. Sur certains groupes de données « test », l'algorithme sera particulièrement bon. Sur d'autres potentiellement beaucoup moins.

Sinon, cette quantité est en général invariante, mais pour certains problèmes et bases de données, elle ne l'est pas autant qu'elle peut paraître.

Par exemple, imaginez que les échantillons de votre base de données sont réparties en deux classes, et l'algorithme doit choisir parmi celles-ci lors d'une prédiction. Imaginez ensuite que 90% des échantillons de votre base de données (et même 90% des échantillons en général) sont de la première classe, et seulement 10% de la seconde. Alors votre algorithme pourrait obtenir une précision de 0.9 (ce qui est très bien) juste en choisissant la première classe à chaque prédiction.

---

18. Nous ne traiterons pas le cas de l'algorithme de régression, c'est-à-dire l'algorithme qui a pour but de prédire une valeur continue.

Votre algorithme a donc une grande précision alors qu'il n'est pas très intelligent. La précision est ici rendue quasi-obsolète de part la nature de la base de données.

Mais le problème énoncé peut-être largement amplifié de part la nature du problème en question. Imaginez que dans la base de données précédentes, les échantillons soient les patients d'un hôpital. Imaginez de plus que 90% d'entre eux souffrent d'un rhume, et 10% d'une maladie grave fatale dans la majorité du temps. Si votre algorithme se trompe en prédisant qu'un patient à la maladie grave alors qu'elle a un rhume, ce n'est pas très grave. Au pire, le patient recevra un traitement inadapté (on suppose ici qu'il est bien pire de souffrir de la maladie grave que de souffrir d'un rhume et de recevoir, à tort, le traitement pour la maladie grave). Cependant, si l'algorithme prédit qu'une personne à un rhume alors qu'elle souffre de la maladie grave (et que celle-ci passe inaperçue), alors cette personne ne sera pas soignée et aura de grande chance de mourir. Nous voyons ici que toutes les prédictions/erreurs n'ont pas la même importance. Or la précision ne rend pas compte de ça. Pour elle, toutes les prédictions ont la même valeur.

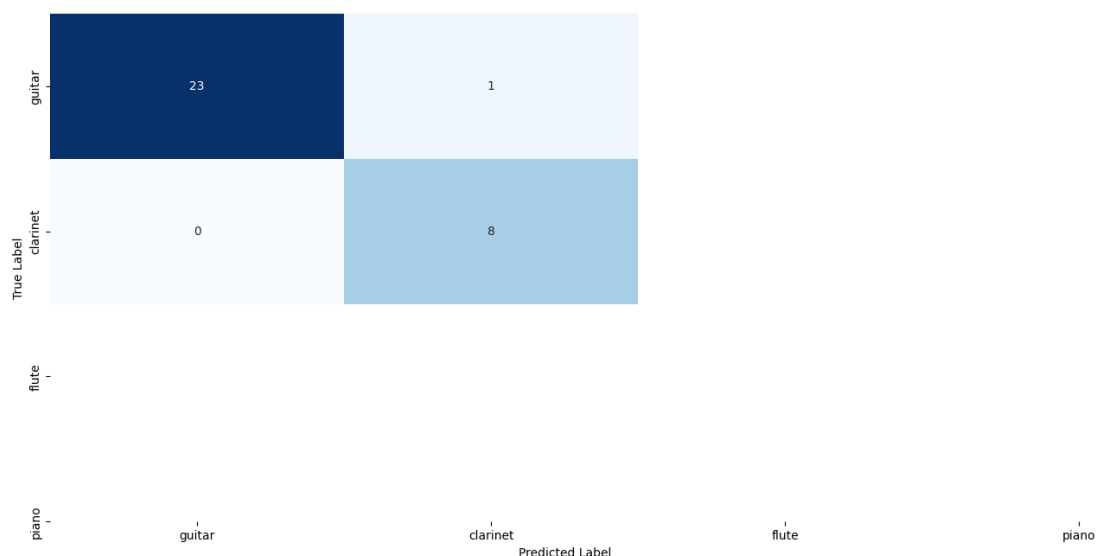
## **Deuxième méthode : la validation croisée**

Avant d'apporter des solutions aux deux problèmes énoncés précédemment, nous allons présenter une seconde méthode qui se base sur la division de la base de données, mais qui pallie à la grande variation potentielle de la précision. L'idée est la suivante : Vous découpez toute votre base de données en  $k$  parties, puis vous entraînez votre algorithme sur  $k-1$  parties, puis testez celui-ci sur la partie restante. Vous répétez le processus  $k$  fois en changeant la partie « test » et en récoltant la précision de l'algorithme à chaque fois. Vous pouvez ensuite calculer la moyenne de la précision, son écart-type, etc., obtenant ainsi des résultats plus robustes. En général, on choisit  $k = 5, 10, 25$ .

Cette méthode est quasiment une norme en matière d'évaluation de performance.

## **La matrice de confusion**

Pour résoudre les deux problèmes de tout à l'heure, nous utilisons la matrice de confusion. La matrice de confusion est un tableau à double entrée qui a pour lignes les classes des échantillons et pour colonnes les classes prédites par l'algorithme pour ces échantillons. Les nombres de prédictions faites dans chaque cas y sont répertoriés. En voici un exemple (matrice de confusion que nous avons obtenu pour les tests de notre algorithme) :



Nous voyons ici que 23 sons de guitare ont effectivement été prédits comme étant des sons de guitare, mais que 1 son de guitare a été prédit comme étant un son de clarinette. Sinon, les 8 sons de clarinette ont été correctement prédits.

Cette matrice contient tout ce qui nous intéresse. Nous pourrions ensuite « zoomer » sur différentes parties de la performance et définir quelques autres quantités utiles.

### Précision revisitée, rappel et F1-score

Ce que nous pouvons maintenant faire, c'est regarder la performance de l'algorithme pour chaque classe. Nous pouvons reprendre l'idée de la « précision générale » vue plus haut, mais en se restreignant à une classe. Nous appellerons cette quantité le « rappel pour la classe C » :

$$\text{rappel}(C) = \frac{\text{nombre de bonnes prédictions pour } C}{\text{nombre d'échantillons effectivement de classe } C}$$

Ce que met en évidence le rappel de C c'est la quantité de prédictions qui n'ont pas été en faveur de C mais qui auraient dû l'être. Maintenant, nous pouvons naturellement nous demander quelle quantité de prédictions ont été en faveur de C mais n'auraient pas dû l'être. Nous définissons une autre quantité qui s'appelle communément « précision pour une classe C » (alors que ça n'a pas grand chose à voir avec la « précision générale » malheureusement) :

$$\text{précision}(C) = \frac{\text{nombre de bonnes prédictions pour } C}{\text{nombre de prédictions pour } C \text{ (bonnes ou mauvaises)}}$$

Que ce soit pour le rappel ou la précision (qui varient de 0 à 1), l'objectif est de s'approcher de 1. Ces deux quantités caractérisent chacune deux types d'erreurs différents, et un type d'erreur peut



être plus ou moins important selon le problème. Par exemple, dans le cas avec les patients d'un hôpital, une erreur de type « omission » (c'est-à-dire que l'algorithme n'a pas prédit C lorsqu'il le fallait) est bien plus grave qu'une erreur de type « action » (c'est-à-dire que l'algorithme a prédit C lorsqu'il ne le fallait pas). En effet, mieux vaut être diagnostiqué gravement malade alors qu'on ne l'est pas plutôt que de ne pas être diagnostiqué gravement malade quand on l'est.

Une dernière quantité existe dans le but de combiner les deux précédentes. Elle s'appelle le « F<sub>1</sub>-score de la classe C ». C'est en fait la [moyenne harmonique](#) du rappel et de la précision de C, c'est-à-dire l'inverse de la moyenne arithmétique des inverses des termes :

$$F_1\text{-score}(C) = \frac{2}{\frac{1}{\text{précision}(C)} + \frac{1}{\text{rappel}(C)}} = 2 \frac{\text{précision}(C)\text{rappel}(C)}{\text{précision}(C) + \text{rappel}(C)}$$

Le F<sub>1</sub>-score donne cependant autant d'importance à la précision qu'au rappel. Pour donner un avantage à l'un plutôt qu'à l'autre, nous utilisons une généralisation du F<sub>1</sub>-score qui s'appelle le F<sub>β</sub>-score.

$$F_\beta\text{-score}(C) = (1 + \beta^2) \frac{\text{précision}(C)\text{rappel}(C)}{\beta^2\text{précision}(C) + \text{rappel}(C)}$$

β est en fait une sorte de facteur de pondération. C'est un réel, et :

- si |β| < 1, on donne plus d'importance à la précision
- si |β| = 1, on donne égale importance à la précision et au rappel (c'est le F<sub>1</sub>-score)
- si |β| > 1, on donne plus d'importance au rappel

## Conclusion

Dans cet annexe, nous avons présenté quelques méthodes d'évaluation de la performance d'un algorithme de classification. Il existe certainement d'autres manières d'évaluer un algorithme, mais l'essentiel est là. Ce qu'il faut garder en tête c'est qu'une bonne connaissance de votre base de données et de la nature de votre problème vous permettra de choisir judicieusement la stratégie d'évaluation de votre algorithme de prédiction.