UMONS
Université de Mons

Faculté des Sciences

Traitement du signal : Projet

Speaker classification

I-ISIA-030

Réalisé par Louis DASCOTTE
& Nicolas DELPLANQUE
& Nicolas SOURNAC

Faculté
des Sciences

3e Bachelier en Sciences Informatiques
Année 2021-2022

ACADÉMIE
UNIVERSITAIRE
WALLONIE-
BRUXELLES

Résumé

Ce rapport contient l'ensemble des résultats obtenus, leurs interprétations ainsi que les explications du fonctionnement de notre implémentation du projet de traitement du signal. Ce projet consiste en la réalisation de solutions visant à classifier des personnes en fonction de leur genre à partir d'enregistrement de leur voix.

Table des matières

1. Exécution du code

Le fichier main.py doit être lancé depuis la racine du projet. Il en est de même pour les fichier utils.py et ml_utils.py, il faut alors exécuter les commandes : "python src/main/utils.py" et "python src/main/ml_utils.py". Les fichiers tests doivent quant à eux être exécuté depuis le dossier où ils sont stockés.

1.1 Librairies utilisées

Afin de réaliser ce projet, nous avons été amenés à utiliser des librairies extérieures aux librairies de bases que nous proposait python :
- audiofile : permet d'extraire le signal et la fréquence d'échantillonnage des fichiers audios.
- numpy (version 1.20) : utilisée pour les différentes fonctions mathématiques.
- scipy : utilisée pour avoir le signal et les transformées de Fourier.
-pickle : utiliser pour sauvegarder et charger des objets.
-pandas : utiliser pour lire les CSV .
-torch : utiliser pour instancier le modèle de machine learning et le paramétrer.
-sklearn : utiliser pour instancier le scaler.

1.2 Structure du code source

Le projet est divisé en plusieurs dossiers. Le dossier racine contient un dossier data contenant l'ensemble des fichiers audio utilisés lors de la réalisation du projet ainsi que les données relatives au modèle de machine learning, et un dossier src contenant la source du programme (ce qui est utilisé par le programme en lui-même ainsi que les tests unitaires).

2. Caractéristiques étudiées

Pour la réalisation du projet, nous avons été amenés à étudier différentes caractéristiques des signaux de parole en vue de les utiliser afin de pouvoir classifier les différents speakers. Pour chacune de ces caractéristiques, nous avons donc implémenté un algorithme qui permet de la calculer et nous avons ensuite analysé et interprété ses résultats afin d'en déduire les règles sur lesquelles la classification se base.

2.1 Energie du signal

L'énergie du signal est calculée à partir de la fonction compute_energy() qui prend un signal en paramètre. Cette fonctionne parcourt simplement l'entièreté du signal en question et y applique la formule du calcul de l'énergie. Nous utilisons les résultats du calcule de l'énergie du signal afin de déterminer si une frame est voiced ou unvoiced.

2.2 Fréquence fondamentale

Le pitch ou la fréquence fondamentale représente la plus basse fréquence qui constitue un signal. Cette valeur doit varier entre 60 Hz pour les voix les plus graves et 550 Hz pour les voix les plus aigües.
Lors du projet, nous avons été amenés à calculer la fréquence fondamentale des speakers en utilisant deux méthodes différentes : La méthode basée sur l'autocorrelation et la méthode basée sur les cepstrums. Pour ce faire, nous avons créé les fonctions autocorrelation_pitch_estim() et cepstrum_pitch_estim().
Pour la méthode basée sur l'autocorrelation, l'algorithme doit recevoir en paramètre une liste de minimum 5 fichiers audio. Nous lisons chaque fichier, normalisons son signal, le divisons en frames de 50 ms et récupérons uniquement les frames dont l'énergie est supérieure au seuil que nous avons déterminé graphiquement. Ensuite, pour chacune des frames restantes, nous calculons sa corrélations avec un maxlags qui vaut 200. Grâce à la fonction find_peaks() fournie par la librairie numpy, nous récupérons facilement l'ensemble des pics du signal corrélé. Il ne nous reste plus qu'à calculer la distance en Hz entre les deux plus hauts pics pour obtenir la fréquence fondamentale de la frame. La fonction autocorrelation_pitch_estim() retourne la moyenne de toutes les fréquences fondamentales calculées à partir des frames en retirant les fréquences supérieures à 550 Hz considérées comme erronées. Nous considérons ce résultat comme étant la fréquence fondamentale de la personne à l'origine des fichiers audio.
Pour la méthode basée sur les cepstrums, le début de l'algorithme est similaire au précédent. Nous lisons également chaque fichiers audio fournis, les normalisons, les divisons en frames de 50 ms et récupérons uniquement les frames dont l'énergie est supérieur au seuil que nous avons déterminé graphiquement. Ensuite, pour chacune des frames restantes, nous calculons dans un premier temps son cepstrum de manière brut et dans un deuxième temps son cepstrum après lui avoir appliqué une fenêtre de hamming. Il suffit ensuite de trouver l'indice du plus haut pics présents dans ces deux cepstrums dans l'interval allant de 60Hz à 550Hz et de le convertir en Hz pour obtenir la fréquence fondamentale de la frame. La fonction cepstrum_pitch_estim() retourne la moyenne de toutes les fréquences fondamentales calculées à partir des frames. Nous considérons ce résultat comme étant la fréquence fondamentale de la personne à l'origine des fichiers audio.

Afin de déterminer des règles de différenciations sur base de la fréquence fondamentale, nous avons exécuté 15 fois les deux algorithmes en donnant des fichiers des deux speakers. Pour BDL nous avons pris la valeur maximum obtenues sur les 15 itérations et l'avons considérées comme borne supérieure du pitch et pour SLT nous avons pris la valeur minimum obtenues sur les 15 itérations et l'avons considérées comme borne inférieure du pitch. Cela nous a donné les règles suivantes :
Pour la méthode basée sur l'autocorrelation, si f0 $<$ 145Hz alors c'est BDL et si f0 $>$ 170Hz alors c'est SLT.
Pour la méthode basée sur les ceptrums, si f0 $<$ 166Hz alors c'est BDL et si f0 $>$ 217Hz, alors c'est SLT.

2.3 Formants

Les formants sont les pics observés sur l'enveloppe de la réponse en fréquence. Dans le cadre de ce projet, l'estimation des formants se base sur un algorithme LPC. Pour ce faire, nous avons créé la fonction compute_formant() qui prend en paramètre un fichier audio. L'algorithme va lire le fichier, le normaliser, et le diviser en frames de 25 ms. Pour chacune des frames, un filtre passe-haut du premier ordre lui est appliqué suivi d'une fenêtre de hamming. Nous calculons ensuite les coefficients LPC et en déterminons les racines associées. Les formants sont directement déduits de ces racines complexes. L'algorithmes retourne donc une liste de formants correspondants aux formants de chacune des frames du fichier audio.
Afin de déterminer des règles de différenciations sur base des formants 1 et 2 nous avons, étant donné que la valeur de ces formants variaient beaucoup pour un même fichier, calculé les formants de plusieurs fichiers audio pour BDL et SLT. De ces formants nous avons extrait les formants 1 et 2 en suivant ces règles : F1 $\in$ [60Hz, 1000Hz] et F2 $\in$ [600Hz, 3200Hz] et pour chacun d'eux, nous les avons affiché sous la forme d'un histogramme. Grâce aux résultats observés et surtout à la position (en rouge) de la moyenne des histogrammes, nous avons déterminé les règles suivantes :
Si F1 $<$ 400Hz alors c'est BDL et si F1 $>$ 380Hz alors c'est SLT.
Si F2 $<$ 1900Hz alors c'est BDL et si F2 $>$ 1800Hz alors c'est SLT.
Voici les observations sur lesquelles nous avons basé notre raisonnement :

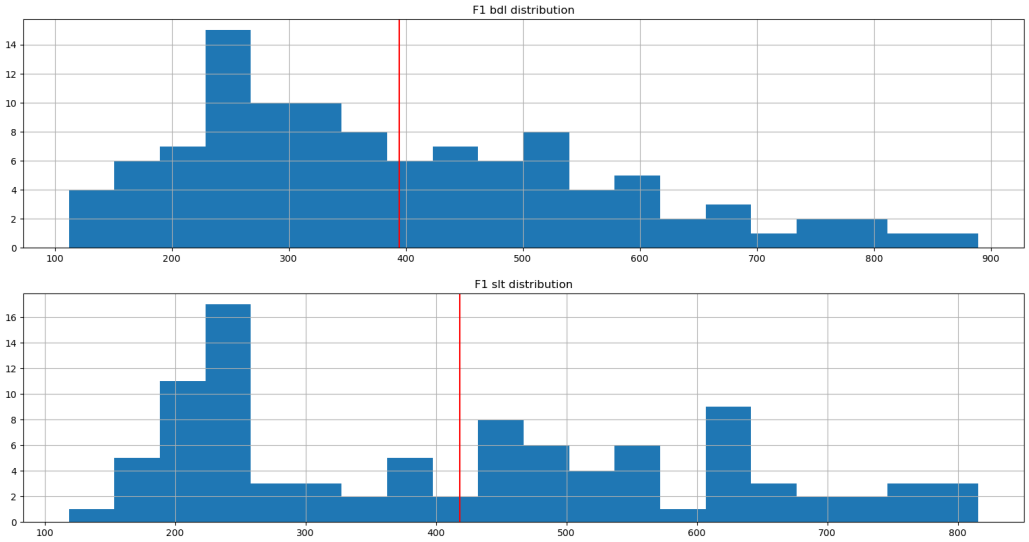

FIGURE 1 – fichier a0006.wav

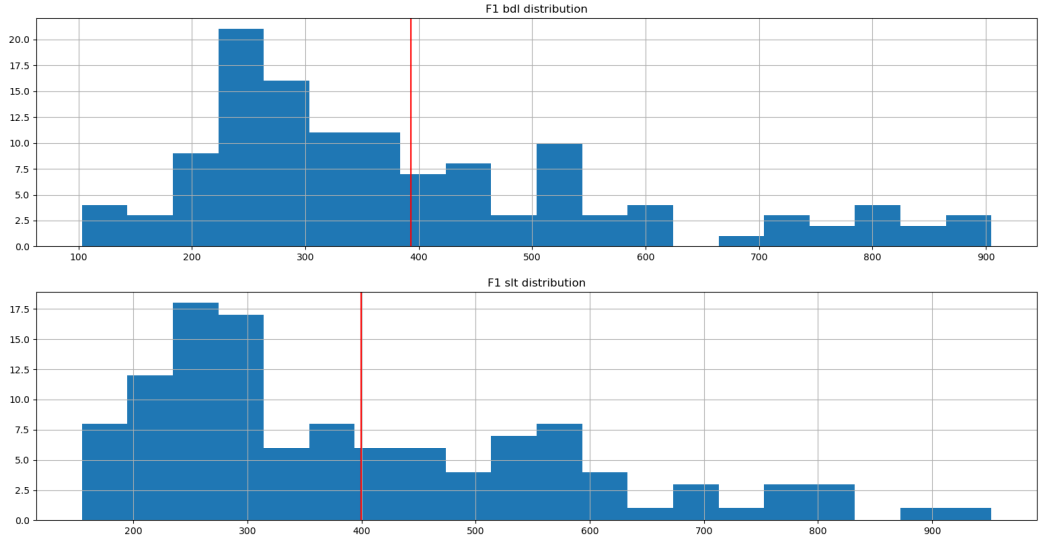

FIGURE 2 – fichier a0392.wav

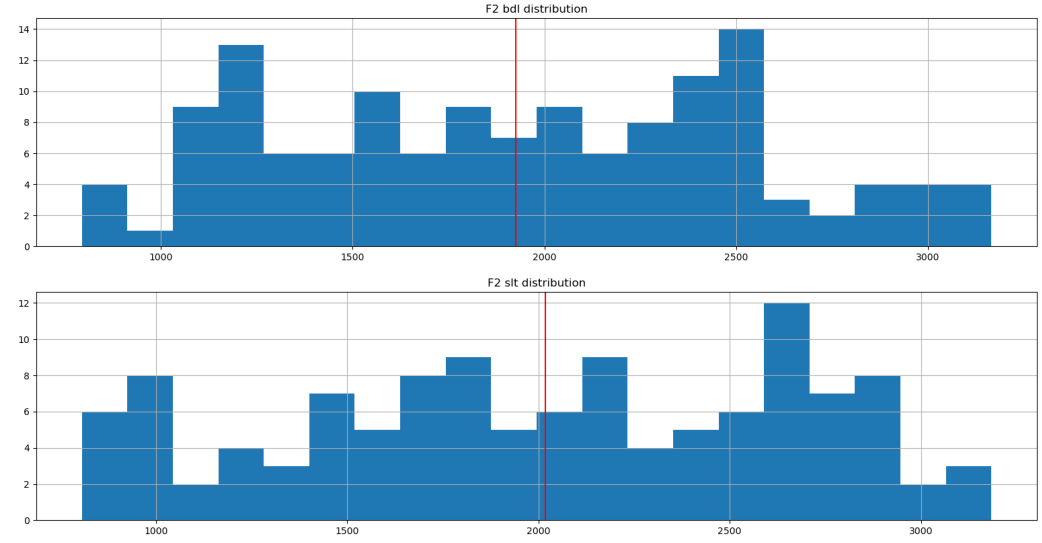

FIGURE 3 – fichier a0347.wav

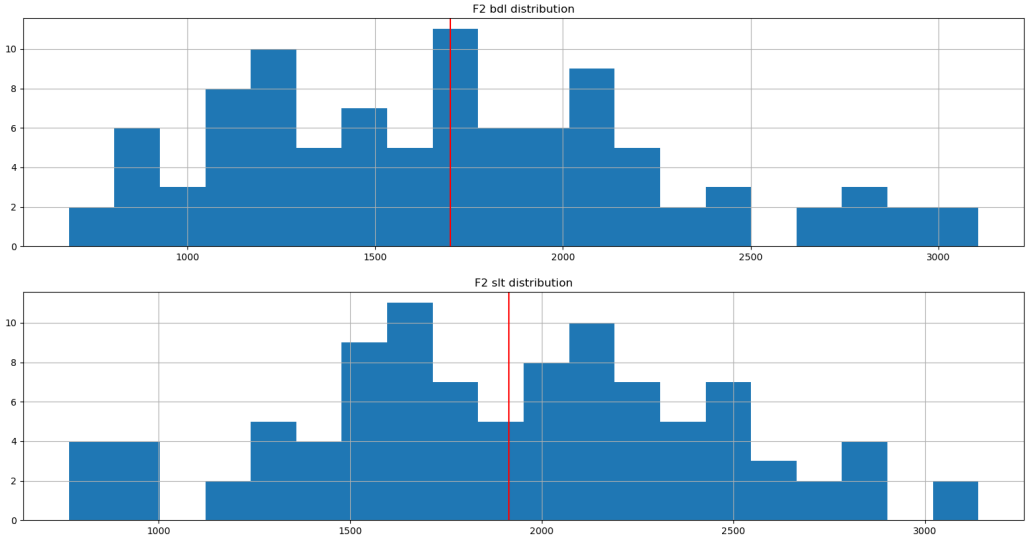

FIGURE 4 – fichier a0494.wav

2.4 MFCCs

Les MFCCs représentent les activitées du conduit vocal durant la production de paroles. Afin de les extraire de nos fichiers audio, on a créé l'algorithme compute_mfcc() prenant un fichier audio comme paramètre. Celui va d'abord extraire le signal et le normaliser. Puis, il va le faire passer par un filtre pass-haut et le diviser en frames de chacune 25ms. Sur chacune de ses frames, une fenêtre de Hamming va être appliquée. Ensuite, on va calculer le spectre de puissance du signal en utilisant l'équation donnée dans l'explication du projet. On changera l'échelle du spectre avec le "MelFilter Bank". Comme les coefficients sont très corrélés, on leur applique une transformée en cosinus discrète, ce qui nous permettra d'avoir les MFCCs, mais on ne prendra que les 13 premiers. Malheureusement, malgré nos tests, nous avons pas pu trouver une façon de les utiliser pour différencier SLT et BDL.

3. Systèmes basés sur des règles

Nous avons élaboré trois systèmes différents pour effectuer la classification homme/femme. Les trois systèmes fonctionnent de la même manière, ils doivent recevoir en paramètre un chemin menant vers un dossier qui contient au minimum 5 fichiers audio contenant des enregistrements de voix d'une même personne. Le système se charge d'appeler les méthodes pour en extraire les différentes caractéristiques et retourne le résultat, c'est à dire s'il détermine que c'est une femme ou un homme. Afin de tester la précision des systèmes, nous les avons exécuté 10 fois en leur donnant des fichiers des speakers BDL (homme), RMS (homme), SLT (femme) et CMS (femme). Nous avons récupérés son nombre de bonnes classifications et en avons calculé son rapport avec le nombre total de classifications. Plus précisément, nous avons calculé la précision métrique de chaque système sur base d'enregistrements de BDL, RMS, SLT et CMS.

3.1 Système 01

Le système numéro 1 est implémenté dans la fonction system_01() et utilise les fréquences fondamentales calculées à partir des deux méthodes décrites ci-dessus ainsi que le formant 1. Nous avons commencé par simplement implémenter les règles définies pour BDL (homme) et SLT (femme) et nous les avons donc extrapolées pour déterminer si il s'agit d'un homme ou d'une femme. Nous avons ensuite affinés les valeurs afin d'améliorer la précision du système. Voici la précision du système :

Globale	BDL	SLT	RMS	CMS
70%	90%	90%	100%	0%

Les résultats du système sont bons pour BDL, SLT et RMS mais nul pour CMS souvent en raison du F1.

3.2 Système 02

Le système numéro 2 est implémenté dans la fonction system_02() et utilise les fréquences fondamentales calculées à partir des deux méthodes décrites ci-dessus ainsi que le formant 1. Ce système a pour but d'améliorer les performances du système précédent. Pour ce faire, nous avons imaginé attribuer des poids aux différentes caractéristiques. Lorsqu'une caractéristique rentre dans une classe (homme ou femme), alors la probabilité que les fichiers soient de cette classe augmente du poids attribué à cette caractéristique. A la fin, le système regarde quelle probabilité est supérieur à 50% pour effectuer son choix. Nous avons attribué des poids de 0.4 pour les deux fréquences fondamentales et 0.2 pour le formant 1. Voici la précision du système :

Globale	BDL	SLT	RMS	CMS
100%	100%	100%	100%	100%

Le système dispose de très bonnes performances pour BDL, SLT, RMS et CMS. Cependant, son choix se fait majoritairement via le calcul de la fréquence fondamentale. Nous nous questionnons donc sur sa précision lorsqu'il sera confronté à des sujets dont la fréquence fondamentale ne se situe pas dans la moyenne définie. Ce système est cependant celui qui dispose de la meilleur précisions pour les sujets étudiés dans le cadre de ce projet.

3.3 Système 03

Le système numéro 3 est implémenté dans la fonction system_03() et utilise les fréquences fondamentales calculées à partir des deux méthodes décrites ci-dessus ainsi que le formant 1 et le formant 2. Le système fonctionne globalement comme le système 1. Il utilise néanmoins le formant 2 en plus et les valeurs utilisées dans les structure conditionnelles ont été affinés. Voici la précision du système :

Globale	BDL	SLT	RMS	CMS
92.5%	80%	90%	100%	100%

Les résultats du système sont bons pour BDL, SLT, RMS et CMS.

4. Machine learning

N'ayant jamais fait de machine learning auparavant, nous avons suivi un tutoriel trouvé en ligne que nous avons adapté au projet.
Ce tutoriel nous a montré comment utiliser un réseau neuronale implémentant la classification binaire. La classification binaire nous a parue parfaite pour notre problème étant donné que nous n'avons que 2 possibilités de genre : Homme ou Femme. Tout le code de la partie Machine Learning se trouve dans le fichier ml_utils.py
Pour utiliser ce modèle, nous devons lui donner des features en entrée. Ces features permettront au modèle de prendre une décision quant-au genre de la personne parlant dans le fichier audio. Ces features sont au nombre de 4, nous avons pris "autocorrelation pitch", "cepstrum pitch", "formant 1" et "formant 2", ces features sont calculées dans la fonction analyse_ml(path, firstAudioName) où path est le chemin d'accès au dossier contenant les fichiers audio en .wav et firstAudioName est le nom du fichier a analyser. Une fois ces features calculées pour l'audio, nous ajoutons un label à cet audio, le label dans notre cas est le sexe de la personne qui parle et équivaut à (Homme = 1 | Femme = 0). Nous répétons l'opération pour tous les fichiers des dossiers bdl_a, bdl_b, slt_a ainsi que slt_b et plaçons tout dans un fichier CSV grâce à la fonction create_TrainingCSV(). Ce fichier sera le fichier d'entrainement de notre modèle. En répétant cette manipulation avec les dossiers rms_a, rms_b, cms_a ainsi que cms_b en passant par la fonction create_TestCSV() nous créons le fichier de test de notre modèle, ce fichier permettra de tester la validité de la classification de notre modèle. Pour créer notre modèle, nous devons charger les 2 fichiers CSV créés précédement, le fichier d'entrainement (training.csv) et le fichier de test (test.csv).

Pour passer les features au modèle nous allons enlever les dernière colonnes des fichiers qui stockent le résultat attendu et les sauvegarder dans des variables (y_train et y_test) pour comparer le résultat du modèle plus tard. Nous allons ensuite instancier un "scaler" pour normaliser nos features d'entrées pour les données d'entrainement et les données de test. Nous fixons le nombre d'EPOCHS (nombre de fois que nous faisons passer le jeux de données d'entrainement au modèle) à 10 car suite à plusieurs essaies, nous avons remarqué que 10 était suffisant et raisonnable. Le BATCH_SIZE est fixé à 64, ainsi nous envoyons des "paquets" de 64 lignes du CSV de test au modèle lorsqu'il s'entraine. Le dernier BATCH envoyé est inférieur ou égal à 64. Le LEARNING_RATE est fixé à 0,001 comme dans le tutoriel. Nous créons ensuite notre "BinaryClassification" modèle en l'instanciant. Le modèle s'entraine ensuite, il effectue une boucle qui lui permet d'optimiser ses WEIGHTs (poids entre les noeuds) et ses BIAS (poids des noeuds) à chaque itération en vérifiant le taux d'erreur qu'il fait. Le taux d'erreur est calculé en comparant la prédiction du vrai label du jeu de test. Une fois le modèle entrainé, il faut savoir le sauvegarder et le charger pour pouvoir l'utiliser, nous utilisons donc la fonction save_BinaryClassificationModel(model, scaler) pour sauver notre modèle et le scaler, la fonction prend en paramètre le modèle a sauvegarder ainsi que le scaler lié à ce modèle. Pour charger le modèle nous utilisons la fonction load_BinaryClassificationModel(pathModel, pathScaler) qui retourne le modèle ainsi que le scaler lié au modèle. La fonction prend en paramètres le chemin d'accès au modèle ainsi que le chemin d'accès au scaler. Pour utiliser le modèle que nous avons entrainé, il suffit d'utiliser la fonction useModel(model, scaler, inputs), avec en paramètre le modèle a utiliser, le scaler associé et une list ([..., ..., ..., ...]) contenant les 4 features d'entrée du modèle. Le modèle prédira si c'est un homme ou une femme en sortie. Le modèle a une efficacité de 100%, cette manière de faire est plus efficace qu'en essayant plusieurs poids pour chaque feature "à la main" car les poids sont optimisé par le modèle. Celui donne directement la combinaison optimale afin d'optimiser la précision de la classification.
Voici le lien du tutoriel sur lequel notre implémentation de la classification binaire s'est basée :
`https://towardsdatascience.com/pytorch-tabular-binary-classification-a0368da5bb89`