Description du projet : Reconnaissance vocale des émotions

Introduction :

L'objectif de ce projet est de reconnaître les émotions dans la voix. Nous souhaitons avoir un modèle qui à partir d'un enregistrement audio pourra identifier une émotion parmi un ensemble d'émotion. Et cela devra se faire indépendamment de la personne qui parle, qu'il s'agisse d'une femme ou d'un homme et peu importe la langue parlée.

Il s'agit donc de classifier des fichiers audios dans une catégorie d'émotions.

Pour cela nous avons besoin de jeux de données composés d'enregistrements audio qui mentionnent l'émotion dans la voix de l'orateur de cet audio.

Il existe des jeux de données qui correspondent à ces critères que nous présenterons plus loin.

Extraction des caractéristiques des enregistrements audio

Nous disposons de 12697 fichiers audios dans notre base de données mais selon les émotions que l'on choisit d'utiliser avant l'exécution du code, le nombre de fichier utiliser réellement peut être moins élevé.

Taux d'échantillonnage

Le taux d'échantillonnage étant le nombre de mesures par seconde du son, plus il est élevé, plus les opérations sur un fichiers audios seront coûteuses en ressources. La bibliothèque *Librosa* que nous utilisons permet de rééchantillonner.

Pour nos fichiers audios qui sont tous aux formats .WAV nous utilisons un taux d'échantillonnage de 16 000Hz.

Stéréophonique ou monophonique

De plus on va préférer utiliser des fichiers audios avec un seul canal audio (monophonique) car tous les sons proviennent de la même source (contrairement au stéréophonique où ils sont placés plus ou moins à gauche ou à droite). En monophonique il n'y a pas de relief dans l'intensité sonore donc il est plus facile de distinguer les différents types de son.

Librosa permet, si besoin, de convertir l'audio en mono en calculant la moyenne des échantillons sur les canaux.

D'après ce que nous venons de dire nous pouvons charger nos fichiers audios en utilisant :

```
X, sample_rate = librosa.load(file_name, sr=16000, mono=True)
```

« librosa.load » renvoie un tableau numpy qui représente les valeurs chronologiques du son (X) ainsi que le taux d'échantillonnage (sample_rate).

Pour mieux comprendre ces données numériques nous les avons retranscrits en spectre audio :

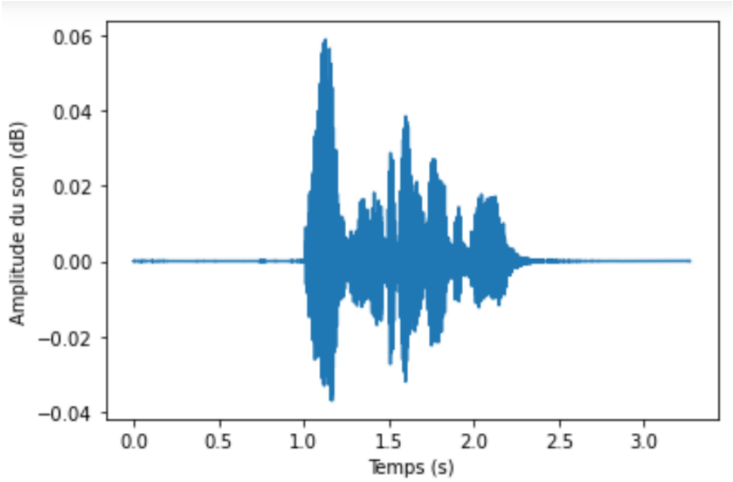

Fig 1. Spectre audio d'un fichier audio.

A partir de ces données avec la bibliothèque Librosa on peut extraire des caractéristiques intéressantes d'un enregistrement audio, ce qui permet de ne pas passer directement par le format .WAV qui contient des informations inutiles :

- **MFCC** :

Les MFCC sont des coefficients qui essayent de modéliser la façon dont le système auditif humain perçoit les sons, plutôt que de les décrire sur une fréquence directement en Hertz.

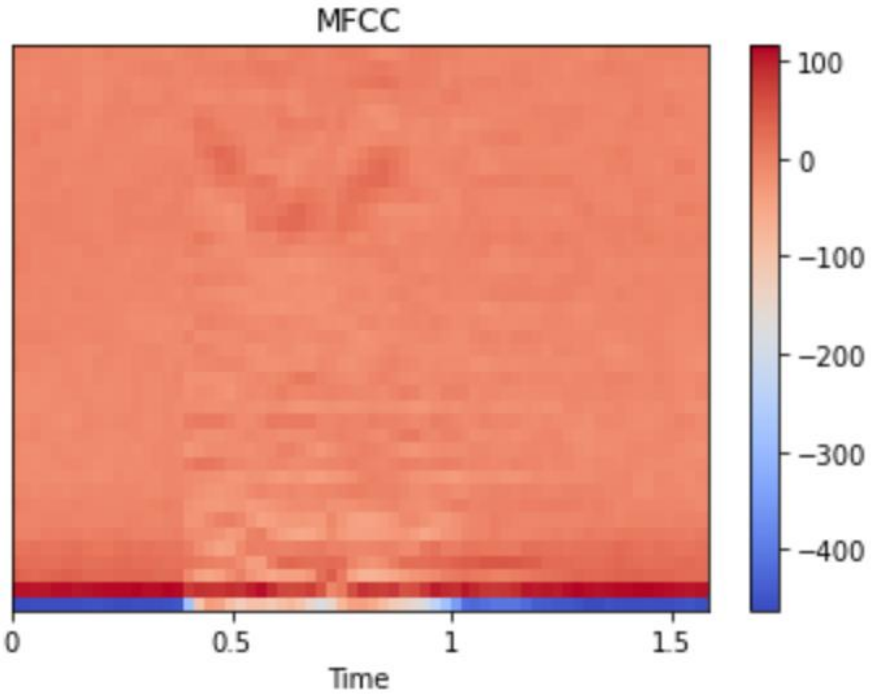

Fig 2. Visualisions de la série MFCC d'un fichier audio.

- **Chromagramme** :

Un chromagramme permet d'identifier les 12 classes de hauteurs différentes dans un son (notamment utilisé pour étudier la musique).

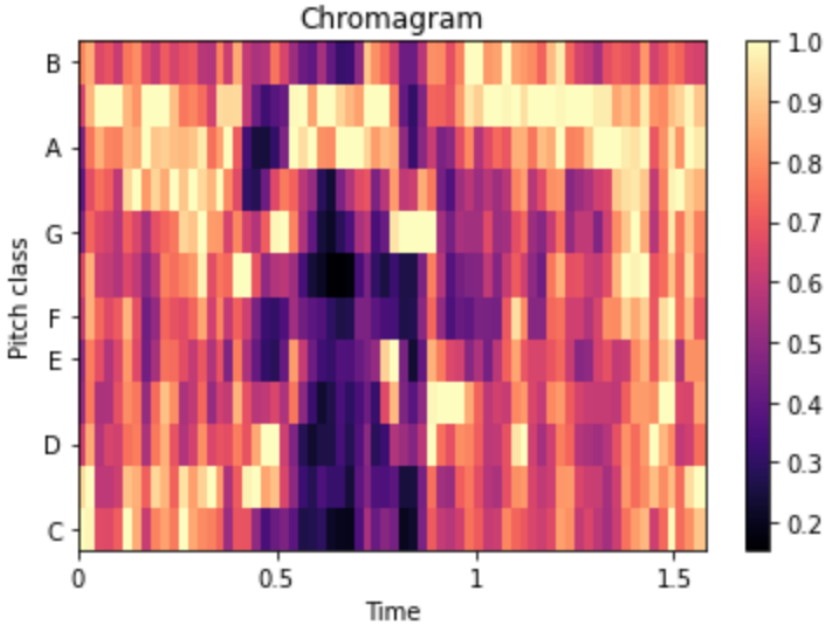

Fig 3. Chromagramme d'un fichier audio.

- **Le spectrogramme Mel**

Un spectre de Mel utilise (comme pour les MFCC) l'échelle de Mel, qui est une échelle se rapprochant de la manière dont les humains perçoivent les fréquences. Le spectrogramme permet de représenter la répartition de ces fréquences converties avec l'échelle de Mel.

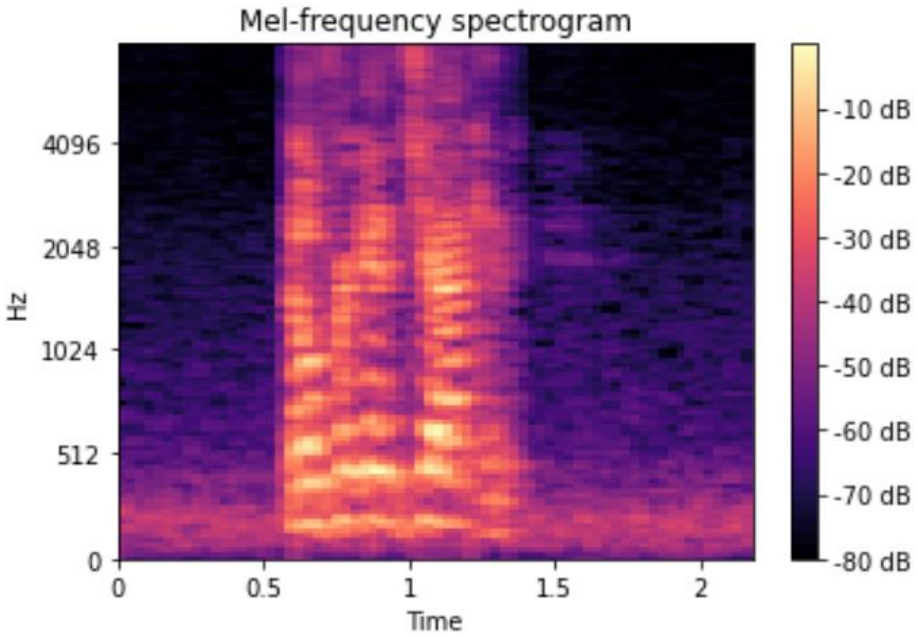

Fig 4. Spectrogramme Mel d'un fichier audio

- Autres caractéristiques que nous pourrions utiliser :
    - ***Contrast*** *:*

      Le contraste spectral traite la force des pics spectraux, des creux et de leur différence séparément dans chaque sous-bande, et représente les caractéristiques spectrales relatives.

    - ***Tonnetz*** *:*

      Le tonnetz est une représentation picturale des notes dans le plan qui révèle les affinités et les structures entre les notes et sur des morceaux de musique concrets.

Néanmoins l'extraction de ces caractéristiques demande beaucoup plus de temps et n'apporte pas énormément d'informations supplémentaires utiles à l'analyse des émotions.

Remarque : Nous avons aussi calculer la transformation de Fourier à court terme (stft). C'est une fonction qui reçoit un signal et le décompose en fréquences. Elle est nécessaire pour calculer le chromatogramme et le contraste spectral d'un fichier audio

Lorsque toutes les caractéristiques sont extraites on les stocke une à une dans une matrice à une dimension et 180 colonnes (40 pour MFCC, 12 valeurs pour le chromatogramme et 120, valeurs pour le spectogramme Mel)

Voici un extrait des données numérique extraite d'un fichier audio qui vont être utilisé par les modèles

	MFCC	Chromatogramme	Spectogramme Mel
0	-321.359558	0.630143	0.002429
1	94.798775	0.671622	0.004857
2	8.361473	0.649829	0.014863
3	23.435230	0.658581	0.050164
4	11.838125	0.611804	0.060343
5	-7.089390	0.594643	0.423530
6	-8.561271	0.606432	0.844245
7	-4.930308	0.673417	4.333143
8	-4.971758	0.690895	6.069478
9	-13.737219	0.786976	1.871774
10	-3.392093	0.851121	0.197791
11	-8.436470	0.684243	0.210842

Fig 5. Extrait du tableau des données numérique extraites d'un un fichier

Modèles utilisés pour la classification

Nous avons essayé plusieurs modèles pour notre classification :

*Utilisation d'un **arbre de décision** :*

```
from sklearn.tree import DecisionTreeClassifier
Arbre_decision = DecisionTreeClassifier(random_state=0, max_depth=20)
print('[*Entrainement du modèle*]')

# Entraînement
clf = Arbre_decision.fit(X_train, y_train)
# Prédiction
ypredit = clf.predict(X_test)
# Calcul de la précision
treeAccuracy = accuracy_score(y_true=y_test, y_pred=ypredit)

print("Précision pour l'arbre de décision: {:.2f}%".format(treeAccuracy*100))
```

```
[*Entrainement du modèle*]
Précision pour l'arbre de décision: 80.87%
```

L'arbres de décision, peut prédire l'émotion d'un enregistrement en utilisant une structure arborescente qui ajustée lors de l'entraînement du modèle.

Pour prédire, il part de la racine de l'arbre et compare les caractéristiques de la racine avec les caractéristiques de l'enregistrement en entrée et passe à un des 2 nœuds fils en fonction du résultat de la comparaison.

Ces comparaisons le font parcourir des branches à l'élément entré dans l'arbre, jusqu'à ce qu'il arrive à un nœud terminal qui le classera dans une des classes d'émotions.

Caractéristiques :

- Nécessite peu de préparation des données
- A tendance à sur-ajuster c'est-à-dire qu'il prend trop en compte les caractéristiques précises des données utilisés pour l'entraîner et donc cela rend moins fiables les prédictions sur des données non similaires

*Utilisation de **SVM** :*

```
from sklearn import svm
clf = svm.SVC(gamma=0.001)
print('[*Entrainement du modèle*]')

#Entraînement
clf.fit(X_train,y_train)
# Prédiction
ypredit = clf.predict(X_test)
# Calcul de la précision
SVMAccuracy = accuracy_score(y_true=y_test, y_pred=ypredit)

print("Précision pour SVM: {:.2f}%".format(SVMAccuracy*100))
```

```
[*Entrainement du modèle*]
Précision pour SVM: 87.40%
```

Un SVM peut être appelé et considéré comme un « séparateur à vaste marge ». Dans notre cas on peut considérer nos enregistrements audios, c'est-à-dire nos ensembles de caractéristiques comme des points ou des vecteurs dans une espace à plusieurs dimensions. Le SVM construit un plan appelé « frontière de décision » permettant de classer les points selon s'ils sont d'un côté ou de l'autre de ce plan. Ces « séparateurs » sont fait de manière à maximiser la marge entre l'enregistrement (le point ou le vecteur) le plus proche du plan et le plan en lui-même.

Le fait d'utiliser plusieurs dimensions permet de trouver un plan qui sépare réellement les différents vecteurs. Et ce peu importe la disposition des échantillons audios si on doit les positionner dans un espace en fonction des caractéristiques de chacun.

Caractéristiques :

- Efficaces dans les espaces de grandes dimensions (s'il y a beaucoup de caractéristiques)
- Il y un certain temps de formation quand il y a beaucoup de données
- Il ne fonctionne pas très bien quand les classe se chevauchent (c'est le cas pour les échantillons audios on peut imaginer que par exemple la colère et la surprise on certaines caractéristiques communes)

*Utilisation de **MLPClassifier** ou **Multi-layer Perceptron :***

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(max_iter=300)
print('[*Entrainement du modèle*]')

# Entraînement
clf.fit(X_train, y_train)
# Prédiction
ypredit = clf.predict(X_test)
# Calcul de la précision
MLPaccuracy = accuracy_score(y_true=y_test, y_pred=ypredit)

print("Précision pour MLPClassifier: {:.2f}%".format(MLPaccuracy*100))
```

```
[*Entrainement du modèle*]
Précision pour MLPClassifier: 87.71%
```

Ce troisième modèle que nous avons utilisé est un réseau de neurone.

Un réseau de neurones possède une couche d'entrée (les neurones d'entrées) et une couche de sortie (les neurones de sortie) et entre des couches de neurones "cachées".

La couche de sortie donne une prédiction sur l'entrée du réseau de neurone.

Un MLP est un réseau de neurone particulier qui, dans notre cas, paramètre un réseau en lui passant dans les caractéristiques de fichiers audios et en comparant la sortie obtenu et la sortie que l'on souhaite (dans notre cas une émotion). Le MLP calcule alors l'erreur locale de chacun des neurones et la corrige et ajustant des paramètres pour que la sortie soit celle qui était attendue. C'est un algorithme de « rétropropagation » qui est chargé de réaliser cette tâche qui a pour but de rendre le réseau le plus performant possible. Par exemple, pour qu'il n'arrive pas à la mauvaise sortie et donc se trompe de prédiction quand on lui passera les caractéristiques d'un audio dont on ne connait pas l'émotion au préalable.

Caractéristiques :

- Utilisable pour beaucoup de problèmes
- Réseau de neurones difficiles à former, l'entraînement demande du temps (mais prédictions rapides)
- Beaucoup de paramètres à définir (nous aurions pu par exemple chercher à optimiser les paramètres pour notre utilisation grâce à un GridSeachCV disponible avec Sklearn)
- Il peut y avoir du surajustement ou au contraire trop de généralisation selon les données utilisées pour l'entraînement

Ce modèle est souvent plus performant que les deux autres modèles essayés précédemment.

Résultats en fonction de l'origine des données et du nombre d'émotions

Jeux de données utilisés :

Crowd Sourced Emotional Multimodal Actors Dataset (CREMA-D)

- 7442 fichiers
- 91 acteurs (48 hommes and 43 femmes entre 20 et 74 ans)
- 12 phrases
- 6 émotions (colère, dégout, peur, joie, neutre et triste)
- Langue : Anglais

RAVDESS Emotional speech dataset

- 1440 fichiers
- 24 acteurs
- 2 phrases
- 8 émotions (neutre, calme, joie, triste, colère, peur, surprise et dégout)
- Langue : Anglais

EmoDB Dataset

- 535 fichiers
- 10 acteurs (5 hommes et 5 femmes)
- 7 émotions (colère, ennui, peur, joie, triste, dégout et neutre)
- Langue : Allemand

Toronto emotional speech set (TESS)

- 2800 fichiers
- 2 actrices (de 26 et 64 ans)
- 200 phrases (qui commencent par « Say the word … »)
- 7 émotions (colère, dégout, peur, joie, triste, neutre et surprise (agréable))
- Langue : Anglais

Surrey Audio-Visual Expressed Emotion (SAVEE)

- 480 files
- 4 acteurs (hommes de 27 à 31 ans)
- 15 phrases par émotions (3 communes, 12 propre à l'émotion)
- Langue : Anglais

<u>Résultats des données par nos modèles sur ces données :</u>

En utilisant les jeux de données « TESS » et « SAVEE » uniquement pour les émotions « neutre », « joie » et « triste » :

```
[Pour l'entraînement] : 1080 fichiers
[Pour le test] : 360 fichiers

Précision pour l'arbre de décision : 93.89%

Précision pour SVM : 97.50%

Précision pour MLPClassifier : 98.06%
```

Si on ajoute tous les autres jeux de données :

```
[Pour l'entraînement] : 4320 fichiers
[Pour le test] : 1441 fichiers

Précision pour l'arbre de décision : 67.04%

Précision pour SVM : 73.49%

Précision pour MLPClassifier : 74.46%
```

Si on ajoute une émotion (colère) :

```
[Pour l'entraînement] : 5858 fichiers
[Pour le test] : 1953 fichiers
Précision pour l'arbre de décision : 61.65%
Précision pour SVM : 69.18%
Précision pour MLPClassifier : 73.07%
```

Au niveau des modèles utilisés, on constate bien que le MLPClassifier et généralement le plus performant.

SVM est moins performant car la séparation entre les classes n'est pas forcément claire comme il s'agit d'émotions et certaines peuvent même avoir des points communs.

Enfin, l'arbre de décision est encore moins performant. Cela est attendu car il est sujet au surajustement et donc ses prédictions ont plus de chances d'être fausses que pour les autres modèles.

Lors de nos tests nous avons remarqués que la précision atteinte par nos modèles variait lorsque nous changions :

- Le **nombre d'émotions** que nous souhaitons classer puisqu'il y a donc plus de chances de se tromper d'émotions pour les modèles.

- La **combinaison des jeux de données**. Chaque jeu de données a des acteurs et des phrases propres (parfois la langue et l'accent varient aussi). Ainsi lorsqu'on les combine, même si l'on entraîne mieux notre modèle, on peut perdre en précision puisque les audios sont moins « habituels » pour le modèle.

Néanmoins certains jeux de données se combinent bien puisqu'en utilisant les jeux de données « TESS » et « SAVEE » uniquement pour les émotions « neutre », « joie » et « triste » on obtient une très bonne précision. En effet, dans environ 98% des cas le modèle « MLPClassifier » ne s'est pas trompé.