Deep Learning

Anisoara ABABII
Yasmina MOUSSA

Université Paris 1 Panthéon-Sorbonne
Mars 2023
Master 2 MOSEF

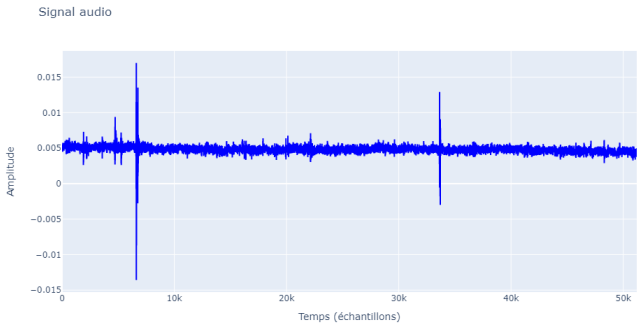
Sommaire

1 Introduction

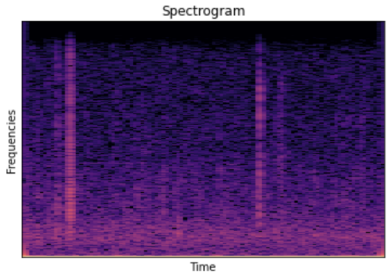
Le projet est basé sur un challenge de l'ENS nommé "Biosonar - Odontoceti Click Detection" par l'Université de Toulon. Les données sont des enregistrements d'audios de sons de dauphins et des sons d'autres animaux marins. Le but du projet est de répondre à la question suivante : l'audio est-il un biosonar (son de dauphin) ou un son perturbateur (autres animaux marins)? Pour résoudre cette problématique, nous allons prendre en input les audios dans différents modèles de Machine Learning et Deep Learning.

1.1 Présentation et traitement des données

Les données sur lesquelles nous avons travaillé sont des données audios, il faudra donc les importés avec des package spéciaux comme soundfile afin de pouvoir les visualisés. Voici un exemple de visualisation de son :



On a ensuite appliqué une transformation STFT Fourrier à court terme pour traiter nos signaux et donc obtenir les fréquences de l'audio dans une dimension temporelle.
Le signal peut aussi être représenter sous forme d'un spectrogramme:



Le signal représenté ici est un signal positif, c'est à dire un biosonar. Les signaux

positifs seront représentés par 1 et 0 sinon dans nos données : c'est un problème de classification binaire.

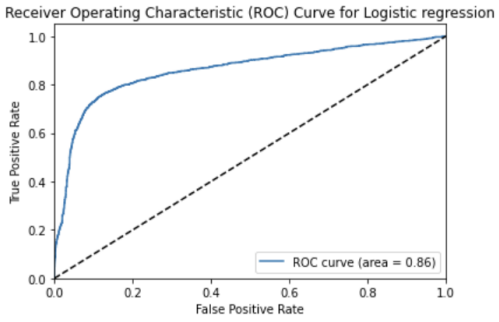
La prochaine étape est de créer les features que nous allons implémentés dans notre modèle. Pour cela, nous avons extrait des caractéristiques des audios
Ce code effectue l'extraction de caractéristiques audio à partir d'un fichier audio au format WAV. On applique un filtre de Butterworth pour obtenir que les signaux entre 5000 et 100000 Hz. Ensuite, on applique un filtre IIR qui permet d'éviter les problèmes d'instabilités qui peuvent provenir d'autres filtrage. Puis on calcul l'amplitude moyenne du signal, son centre de gravité spectral, la largeur spectrale et la platitude spectrale. On calcul ensuite la moyenne, l'écart-type, la valeur minimale pour chacune de ces quatre caractéristiques pour enfin obtenir 16 inputs pour chaque audio.

2 Entrainement et résultats des modèles

Nous allons tester deux types de modèles : un modèle de Machine Learning (Régression Logistique) et un modèle de Deep Learning (architecture de neurones). On divise notre base de données en train et test. Pour cela, nous avons décidé d'appliquer un stratified K-fold afin de trouver un effectif équilibré de 1 et de 0 (la variable target) entre les deux bases et non de faire un split test/train random.

2.1 Régression Logistique

La régression logistique est un bon choix de modèle dans notre cas car on travaille sur un problème de classification binaire. La métrique utilisée pour évaluer notre modèle est l'AUC. On obtient de très bon résultat avec un AUC égal à 0,86 qui est proche d'un et donc représente un modèle performant. L'air en dessous de la courbe est grand, comme on peut le voir sur le graphique ci-dessous :



2.2 Réseaux de neurones

Première couche:

L'architecture du modèle est la suivante : on met en input nos features dans une première couche de "Convolution1D". La couche de convolution permet d'appliquer des filtres à nos inputs extrayant les variables les plus importantes. On combine cette couche avec une fonction d'activation ReLu qui resors un output de zéro si l'output de base est inférieur à zéro.

Deuxième couche:

La seconde étape consiste à créer une couche de GlobalMaxPooling qui nous permet de réduire la dimension de nos données d'entrées. Le GlobalMaxPooling prend la valeur maximale de chaque vecteur d'input, résultant en un vecteur de plus petite taille. De cette façon, le modèle se concentre uniquement sur les variables les plus importantes pour la classification.

Troisème, quatrième, cinquième couche:

La troisième couche est aussi une couche avec une fonction d'activation ReLu.

La quatrième est une couche dropout qui nous permet d'appliquer une régularisation qui supprime certains neurones de la couche précédente afin d'éviter le surapprentissage ("overfitting). Nous avons ainsi décidé de le fixer à 0,25.

La dernière couche prend en entrée les données de sortie des couches précédentes et leur applique la fonction d'activation "Sigmoid". La fonction "Sigmoid" est souvent utilisée dans un problème de classification binaire comme le nôtre.
Après avoir construit notre modèle, nous l'avons compilé. La fonction de coût choisi pour notre modèle est la "Binary crossentropy" étant donné qu'il s'agit de la plus adéquate pour un problème de classification binaire. Le but est de minimiser notre fonction de coût pour que l'erreur entre notre Y estimé et Y observé soit minime.

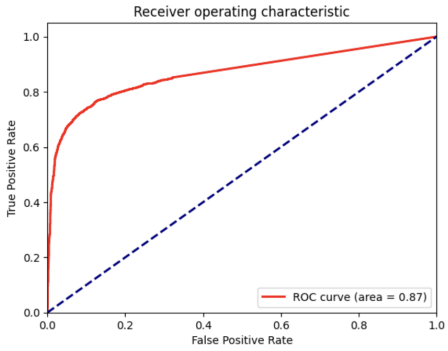
Il s'agit d'appliquer une méthode de descente de gradient pour déterminer les paramètres optimaux (le poids et le biais). Au début, on initialise des poids et biais aléatoires. Le modèle tourne (forward propagation) et on obtient une probabilité grâce à la fonction d'activation (dans notre cas Sigmoid). L'erreur est ensuite calculée avec la fonction de coût choisi. Le but est de rétropropager ("backpropager") la fonction de coût afin d'ajuster les poids pour qu'ils minimisent la fonction de coût.
La descente de gradient à lieu à cette étape, on veut trouver le minimum global de la fonction de coût. Cette dernière est convexe, quand on multiplie par la dérivée, on s'approche du minimum global.

Nous commençons donc au point où la fonction de coût est élevée jusqu'à obtenir les poids optimaux minimisant la fonction de coût. Le taux d'apprentissage ("learning rate") est aussi un paramètre à préciser, Il définit la vitesse de la descente de gradient. Si on choisit un taux d'apprentissage trop faible, le modèle convergera plus lentement vers le minimum mais ce modèle expliquera bien les données. Si on choisit un taux d'apprentissage trop élevé, la vitesse de convergence augmentera mais il y aura un risque de ne pas atteindre le minimum optimal. Le taux d'apprentissage choisi pour notre modèle est de 0,001. Avec les nouveaux poids obtenus, on réapplique ainsi la forward propagation.

L'optimiseur choisi pour notre modèle est l'optimiseur "Adam" (Adaptive Moment Estimation), qui est une méthode de descente de gradient stochastique. Cette méthode est efficace car elle adapte le taux d'apprentissage en fonction de nos paramètres.

On obtient finalement un modèle avec une accuracy égal à 0.7433. Pour ce challenge, on va plutôt utiliser l'AUC pour évaluer notre modèle mais aussi pour la comparé à l'AUC de la régression logistique.

On obtient une AUC de 0.87 avec notre réseau de neurones ce qui est supérieur au résultat obtenu avec la régression logistique. On obtient la courbe suivante :



3 Conclusion

En conclusion, nous avons commencé par traiter et représenter nos données sonores afin d'extraire différentes features du son. Ensuite, nous avons tester deux modèles différents : la régression logistique et une architecture de réseaux de neurones. La métrique utilisée est l'AUC : elle est plus élevé pour le modèle de Deep que pour la régression logistique. Il faudrait donc garder le modèle le plus performant ici notre réseau de neurones. Ce modèle est plus avantageux que la régression logistique car on travaille avec des données très volumineuse,

puisqu'on a plus d'information on peut se permettre de construire un modèle plus complexe qu'une simple régression logistique, ce qui nous permet d'obtenir de meilleurs résultats en terme d'accuracy.