Deep Learning Project

Mathis DOUSSE and Inès FAIVRE

Université Lumière Lyon 2

octobre 2023 - décembre 2023

Abstract

L'objectif de cette étude consiste à réaliser un outil de classification fondé sur un algorithme utilisant les réseaux de neurones. La première étape consiste à choisir un jeu de données qui répond à un certain nombre de caractéristiques. Puis, la seconde étape porte sur le développement d'une architecture de classification, en utilisant un réseau de neurones artificiels.

Keywords : Batch normalization, deep learning, dropout, gradient clipping, heuristiques, hyper-paramètres, réseau de neurones artificiels.

1 Introduction

Dans cette étude, nous allons tenter de résoudre un problème de classification en utilisant un réseau de neurones artificiels. Pour ce faire, nous devions dans un premier temps choisir un jeu de données sur lequel porter notre étude. Nous avons opté pour le jeu de données Spotify Songs qui contient plus de trente mille chansons représentées par différentes variables. Ensuite, nous devions pré-traiter nos données afin de les rendre expoitables pour l'étude et établir une architecture de réseau de neurones artificiels. Enfin, à l'aide de différentes modifications sur le classifieur et de méthodes d'optimisation, nous porterons des expérimentations afin d'améliorer notre classification et présenterons les résultats obtenus.

C'est pourquoi, dans un premier temps, nous présenterons une description détaillée des données que nous avons sélectionnées et du problème de classification qui en résulte. Puis, nous mettrons en lumière les expériences que nous avons implémenté pour résoudre notre étude en présentant notre classifieur et les différentes méthodes d'optimisation utilisées. Durant votre lecture, vous trouverez des résultats sous la forme de tableaux de graphes ou de textes tous provenant du code accessible au lien suivant : https://github.com/FeckNeck/deep-learning-project

2 Description du jeu de données

Notre jeu de données Spotify Songs a été emprunté sur Kaggle. Comme dit précédemment, il contient pus de trente mille chansons et vingt-trois variables explicatives définies comme suit :

1. track_id **character** ID unique de la chanson
2. track_name **character** Nom de la chanson
3. track_artist **character** Artiste
4. track_popularity **double** Popularité (entre 0 et 100)
5. track_album_id **character** ID unique de l'album
6. track_album_name **character** Nom de l'album
7. track_album_release_date **character** Date album
8. playlist_name **character** Nom de la playlist

9. playlist_id **character** ID playlist
10. playlist_genre **character** Variable à prédire : Genre de la playlist (6 modalités)
11. playlist_subgenre **character** Sous-genre de la playlist
12. danceability **double** La "danceability" décrit à que point un morceau se prête à la danse (0,0 est la moins dansable et une valeur de 1,0 est la plus dansable)
13. energy **double** L'énergie est une mesure allant de 0,0 à 1,0 et représente une mesure perceptuelle de l'intensité, le dynamisme du morceau
14. key **double** La tonalité globale estimée de la piste. Les nombres entiers correspondent aux hauteurs en utilisant la notation standard de la classe de hauteur.
15. loudness **double** Volume sonore global d'une piste en décibels (dB)
16. mode **double** Le mode indique la modalité (majeure 1 ou mineure 0) d'une piste, c'est-à-dire le type d'échelle à partir duquel son contenu mélodique est dérivé
17. speechiness **double** L'attribut speechiness détecte la présence de paroles dans une piste, plus il y en a, plus on se rapproche 1.
18. acousticness **double** Mesure de confiance de 0,0 à 1,0 indiquant si la piste est acoustique
19. instrumentalness **double** Indique si une piste ne contient pas de voix, moins il y en a, plus on se rapproche de 1.
20. liveness **double** Détecte la présence d'un public dans l'enregistrement, même principe que *instrumentalness*
21. valence **double** Mesure de 0,0 à 1,0 décrivant la positivité musicale transmise par une piste
22. tempo **double** Le tempo global estimé d'une piste en battements par minute (BPM)
23. duration_ms **double** Durée de la chanson en millisecondes.

Le jeu de données que nous avons choisi devait répondre à un certain nombre de caractéristiques telles que le volume de données qui devait être assez important (quelques milliers, dans notre cas, largement suffisant), la tâche de classification qui ne doit pas être trivial. Dans notre cas, nous souhaitons prédire le genre de la playlist dans laquelle se trouve la chanson à l'aide des variables qui nous sont fournies. C'est pourquoi, nous allons donc faire un tri dans ces variables afin de respecter cette caractéristique, notamment en retirant la variable playlist_subgenre, trop corrélée à notre variable à prédire, playlist_name qui donne également trop d'informations sur le genre de la playlist, par exemple. Enfin, le traitement de ces données devait constituer un défi qui n'a pa déjà été résolu par de tiers personnes. Nous avons alors cherché sur Kaggle de possibles expérimentations sur ce jeu de données mais aucune ne concerne un problème de classification.

Pour revenir sur la variable que nous voulons prédire (playlist_genre), celle-ci correspond au genre de la playlist dans laquelle appartient la musique, autrement dit le genre de la musique en elle-même. Cette variable contient 6 modalités :

$$\textbf{pop}, \textbf{rap}, \textbf{rock}, \textbf{latin}, \textbf{r\&b} \textit{ et } \textbf{edm} \tag{1}$$

représentant les 6 genres des 30 000 musiques que nous avons (r&b étant Rhythm and blues and edm Electronic Danse Music). Nous avons étudié la fréquence des ces différents genres au sein du jeu de données et ceux-ci sont assez bien distribués. Nous n'aurons donc pas besoin de recourir à des méthodes d'under- ou d'over-sampling pour rééquilibrer le dataset.

3 Expériences

3.1 Méthodologie

Dans cette section, nous allons aborder les pré-traitements effectués sur les données, le type d'architecture de réseau de neurones artificiels que nous allons utiliser pour commencer à étudier nos résultats et l'effet de cette architecure sur nos données. Dans l'objectif ensuite d'y appliquer des modifications ou des méthodes d'optimisation.

3.1.1 Pré-traitements

Avant toute chose, comme abordé précédemment, il a fallu trier les différents variables de notre jeu de données. En effet, bon nombre d'entre elles (par exemple, *track_name*, *track_artist*, etc) sont des variables catégorielles avec autant de modalités que de lignes (voir 1).

	track_id	track_name	track_artist	track_album_id	track_album_name	track_album_release_date	playlist_name	playlist_id	playlist_genre	playlist_subgenre
count	32833	32828	32828	32833	32828	32833	32833	32833	32833	32833
unique	28356	23449	10692	22545	19743	4530	449	471	6	24
top	7BKLCZ1jbUBVqRi2FVlTVw	Poison	Martin Garrix	5L1xcowSxwzFUSJzvyMp48	Greatest Hits	2020-01-10	Indie Poptimism	4IkkvMpVH4ISioqQjeAL0q	edm	progressive electro house
freq	10	22	161	42	139	270	308	247	6043	1809

Figure 1: Description des variables catégorielles avec leur nombre de modalités (ligne 'unique').

Nous nous sommes donc séparés de ces variables-ci ainsi que les variables *playlist_name* et *playlist_subgenre* trop liées avec notre variable à prédire. Ensuite, les données que nous utilisons ont des unités très différentes, on a des décibels pour la variable *loudness* ou encore les battements par minutes (BPM) pour la variable *duration_ms*. Il a alors fallu normaliser nos données afin de ne pas biaiser nos résultats.

De plus, malgré que nos données catégorielles soient très variées, nous avons souhaité garder la variable *track_album_release_date* qui représente la date de sortie de l'album. Pour ce faire, nous avons conservé uniquement l'année de sortie de l'album, puis nous avons créé des intervalles afin de regrouper les années par décennie et nous avons utilisé une technique de OneHotEncoding pour rendre la variable exploitable par la suite. Sur ce graphique, (voir 2), qui représente la distribution des sous-genres musicaux par intervalle d'années, on peut constater que certains styles musicaux ne sont présents que pour certaines décennies (notamment le classic rock ou le neo soul sur les années 50/60). Cette variable pourra donc s'avérer utile dans la classification de nos "nouveaux" individus (chansons).

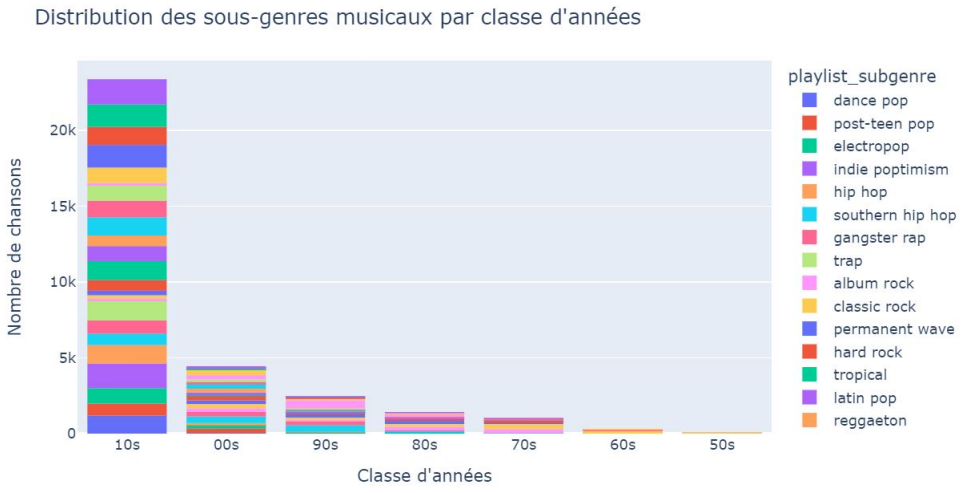

Figure 2: Distribution des sous-genres musicaux par intervalle d'années.

Enfin, une étape très importante de notre pré-traitement consiste en la vérification des corrélations entre les différentes variables prédictives. En effet, il est préférable de retirer cette corrélation car elle apporterait un biais dans autre modèle dû à une information redondante et similaire. (voir 3) On

constate que toutes nos variables sont peu corrélées entre elles, seulement un coefficient de corrélation moyennement élevé entre les variables *energy* et *loudness*. Cependant, nous n'allons pas pour autant les supprimer car elles ne sont pas fortement corrélées et nous avons déjà peu de variables.

	track_popularity	danceability	energy	key	loudness	mode	speechiness	acousticness	instrumentalness	liveness	valence	tempo	duration_
rack_popularity	1.000000	0.064754	-0.108984	-0.000405	0.057717	0.010553	0.007067	0.085042	-0.150003	-0.054593	0.033278	-0.005538	-0.1436
danceability	0.064754	1.000000	-0.086074	0.011771	0.025351	-0.058711	0.181808	-0.024515	-0.008658	-0.123899	0.330538	-0.184132	-0.0969
energy	-0.108984	-0.086074	1.000000	0.009972	0.676662	-0.004778	-0.032184	-0.539732	0.033282	0.161317	0.151050	0.150072	0.0125
key	-0.000405	0.011771	0.009972	1.000000	0.000920	-0.173981	0.022462	0.004378	0.006022	0.002834	0.019933	-0.013316	0.0151
loudness	0.057717	0.025351	0.676662	0.000920	1.000000	-0.019242	0.010313	-0.361646	-0.147823	0.077589	0.053411	0.093761	-0.1150
mode	0.010553	-0.058711	-0.004778	-0.173981	-0.019242	1.000000	-0.063446	0.009399	-0.006760	-0.005485	0.002567	0.014339	0.0155
speechiness	0.007067	0.181808	-0.032184	0.022462	0.010313	-0.063446	1.000000	0.026168	-0.103385	0.055337	0.064756	0.044649	-0.0894
acousticness	0.085042	-0.024515	-0.539732	0.004378	-0.361646	0.009399	0.026168	1.000000	-0.006881	-0.077247	-0.016833	-0.112782	-0.0815
strumentalness	-0.150003	-0.008658	0.033282	0.006022	-0.147823	-0.006760	-0.103385	-0.006881	1.000000	-0.005505	-0.175406	0.023303	0.0632
liveness	-0.054593	-0.123899	0.161317	0.002834	0.077589	-0.005485	0.055337	-0.077247	-0.005505	1.000000	-0.020432	0.020887	0.0061
valence	0.033278	0.330538	0.151050	0.019933	0.053411	0.002567	0.064756	-0.016833	-0.175406	-0.020432	1.000000	-0.025639	-0.0322
tempo	-0.005538	-0.184132	0.150072	-0.013316	0.093761	0.014339	0.044649	-0.112782	0.023303	0.020887	-0.025639	1.000000	-0.0013
duration_ms	-0.143634	-0.096922	0.012560	0.015141	-0.115039	0.015576	-0.089432	-0.081553	0.063256	0.006197	-0.032292	-0.001347	1.0000
Year_50s	-0.008384	-0.019455	-0.010254	0.007731	-0.004573	0.008586	-0.006861	0.031094	-0.005504	0.006549	0.013245	0.019473	-0.0178
Year_60s	0.021281	-0.080793	-0.037504	-0.014989	-0.087897	0.041403	-0.044908	0.044130	-0.001194	-0.002146	0.043260	0.006187	0.0034
Year_70s	0.034031	-0.137549	-0.046675	-0.014507	-0.162558	0.064254	-0.089656	0.041835	-0.026870	0.000929	0.074954	0.017046	0.1017
Year_80s	0.008044	-0.061569	0.001988	-0.006601	-0.190135	0.048475	-0.095810	-0.028004	-0.035021	-0.013136	0.100027	0.001672	0.1533
Year_90s	-0.033613	0.032860	-0.049250	0.021419	-0.142686	0.008625	0.041130	-0.038671	-0.062465	0.028670	0.107989	-0.058729	0.1743
Year_00s	-0.137528	-0.027787	0.046594	0.002242	0.024024	0.011678	0.014757	-0.027656	-0.060671	0.020789	0.151016	-0.015575	0.1341
Year_10s	0.103009	0.097297	0.017259	-0.002972	0.226146	-0.067647	0.050279	0.030285	0.107710	-0.026482	-0.257307	0.036495	-0.3076

Figure 3: Coefficient de corrélation entre les variables de notre modèle.

Pour commencer notre étude, puisqu'aucun travail de classification n'a été fait avec ce jeu de données, nous allons tester d'autres modèles de machine learning afin de pouvoir comparer nos résultats. Par exemple, (voir 4), nous avons choisi d'emprunter la **régression logistique** qui est une technique de machine learning utilisée pour prédire une variable cible en fonction de variables indépendantes, reposant sur une fonction logistique (ou sigmoïde) et a pour objectif de prédire la probabilité d'appartenance à une classe spécifique. Ainsi que l'**arbre de décision**, un modèle d'apprentissage automatique qui utilise une structure arborescente pour représenter des décisions et des actions basées sur des conditions, le **KNN** (K-Nearest Neighbors), un algorithme d'apprentissage supervisé basé sur le principe que des points de données similaires ont tendance à se regrouper dans l'espace des caractéristiques. Et enfin, nous avons souhaité tester la méthode des **forêts aléatoires** qui combine un ensemble d'arbres de décision indépendants afin de voir si nous pouvions d'ores-et-déjà améliorer nos résultats.

```
Accuracy score avec classification par régression logistique sur le jeu de données de test: 0.49
Accuracy score avec classification par arbre de décision sur le jeu de données de test: 0.45
Accuracy score avec classification par KNN sur le jeu de données de test: 0.50
Accuracy score avec classification par Random Forest sur le jeu de données de test: 0.57
```

Figure 4: Accuracy sur nos données en utilisant d'autres modèles de classification.

On constate qu'avec différentes méthodes de machine learning, les variables qui servent à expliquer notre variable cible ne permettent pas de le faire entièrement. Ainsi, nous obtenons des résultats assez mauvais oscillant entre 45 et 57% d'accuracy. Cependant, il est courant que les données que nous avons en entreprise par exemple ne permettent pas de répondre totalement à la problématique posée. C'est pourquoi, notre objectif sera alors d'améliorer les résultats que nous avons obtenu avec ces méthodes en utilisant cette fois-ci une architecture de réseau de neurones.

3.1.2 Architecture de base

En partant de cette base, nous allons pouvoir implémenter notre premier réseau de neurones artificiels. Dans un premier temps, nous avons souhaité faire simple afin de commencer avec un modèle avec peu de paramètres et de ne pas s'affoler sur des algorithmes trop complexes. En effet, il est possible qu'un modèle simple suffise pour répondre à notre problème de classificiation. Pour cela, nous avons implémenté sur 10 epochs, un perceptron à 1 couche cachée avec comme entrée autant de features que nous avons sélectionné, une couche cachée avec une fonction d'activation **Sigmoid** (qui donne une valeur entre 0 et 1), l'utilisation de la **cross entropy** comme fonction de coût (plus intéressante car elle cherche plus finement les erreurs les plus importantes) et la fonction **softmax** en sortie car on a

affaire à un problème de multi-classes (et non pas de multi-labels) :

Algorithm 1: Perceptron multi-couches

Input : E ensemble de données : $E = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$;
w les paramètres du réseau de neurones initialisés
h la fonction d'activation, η le pas d'apprentissage

Output: classification 6 classes avec softmax

1: **repeat**
2: **for** tout $e_i = (x_i, y_i)$ dans E **do**
3: $s = \sum_{i=0}^{n} w_i \cdot x_i$
4: $\widehat{y} = h(s(x))$
5: $\delta_k = \widehat{y_k}(1 - \widehat{y_k})(y_k - \widehat{y_k})$
6: $\delta_j = \widehat{y_j}(1 - \widehat{y_j}) \sum_{i=1}^{k}$
7: $\Delta w_{ij} = \eta \delta_k x_i$
8: **end for**
9: **until** un certain critère de convergence / epochs

De plus, nous avons souhaité d'ores-et-déjà implémenté un système de batch pour calculer notre gradient. On a donc créé des batchs de taille 5000 car nous avons tout de même plus de 30 000 musiques. On a tester plusieurs valeurs de batch mais si l'on réduit leur taille, nos modèles sont très coûteux en temps d'exécution (pour des batchs de taille 2500 musiques, un MLP 1 couche cachée avec 10 epochs prend plus de 6 minutes, et n'améliore pas pour autant les résultats).

Avec cette architecture, nous obtenons des résultats très similaires (54% d'accuracy) à ceux trouvés avec les différentes méthodes de machine learning (knn, etc.). C'est pourquoi, nous allons essayer d'améliorer nos résultats, en modifiant notre architecture et en s'attelant tout d'abord au nombre de couches et aux fonctions d'activitation. Nous allons notamment tester la fonction **ReLU** (qui donne le maximum entre x et 0), l'optimiseur **Adam** et non plus seulement **SGD** (Stochastic Gradient Descent) et la fonction d'activiation **tanh** (permet d'appliquer une normalisation aux valeurs d'entrée et sort un résultat entre -1 et 1). En modifiant ces paramètres, et en faisant plusieurs tests, nous obtenons les résultats suivants en évaluant l'accuracy et le temps d'exécution :

Architecture	score % (worst/best)	temps en min (worst/best)
MLP 1 couche cachée_Sigmoid_SGD	54 ± 56	4.40 ± 3.18
MLP 1 couche cachée_ReLU_Adam	47 ± 49	5.34 ± 6.06
MLP 1 couche cachée_Tanh_Adam	48 ± 50	5.24 ± 4.36
MLP 5 couches cachées_Sigmoid_SGD	15 ± 18	7.19 ± 6.14
MLP 2 couches cachées_Sigmoid_Adam	46 ± 50	5.22 ± 4.48

Table 1: Résultats des différentes architectures sur nos données de test (score) et temps en apprentissage

3.2 Ajout d'heuristiques

Pour améliorer les résultats que nous avons obtenu avec des architectures de réseau de neurones artficiels très simples, nous allons opter pour des méthodes d'optimisation. Pour ce faire, nous avons sélectionné notre meilleur modèle obtenu précédemment (MLP 1 couche cachée avec Sigmoid et SGD) pour tester ces heuristiques. Tout d'abord, nous allons tenter d'implémenter le gradient clipping.

Le gradient clipping est une technique utilisée dans l'apprentissage automatique pour limiter la valeur du gradient lors de la rétropropagation du gradient dans un réseau de neurones. L'objectif du gradient clipping est de prévenir les explosions de gradient, qui se produisent lorsque les valeurs du gradient deviennent très grandes et peuvent entraîner des problèmes de convergence du modèle. Nous avons souhaité également tester la batch normalization. Celle-ci, aussi appelée la normalisation par lots, est une technique utilisée dans l'apprentissage automatique pour normaliser les activations d'un réseau de neurones artificiels. Elle vise à accélérer l'apprentissage et à améliorer la stabilité et la performance du modèle. La batch normalization est appliquée à chaque couche du réseau de neurones,

elle agit comme une régularisation en ajoutant du bruit aux activations, ce qui peut aider à prévenir le surapprentissage. Enfin, il était possible de mettre en place le dropout au sein de nos réseaux. Le dropout est une forme de régularisation basée sur l'estimation d'un ensemble de réseaux calculés à partir du réseau initial. C'est à l'aide de ces différentes techniques d'optimisation que nous avons souhaité améliorer notre modèle. Voyons alors les résultats.

4 Résultats

Heuristiques	score % (worst/best)	temps en min (worst/best)
Gradient clipping	53.3 ± 54.6	3.10 ± 2.29
Batch normalization	52.2 ± 49.9	4.23 ± 4.11
Dropout (4 couches cachées)	$51.40 \pm 0.51.7$	0.32 ± 0.15

Table 2: Résultats des différentes heuristiques sur nos données de test (score) et temps en apprentissage

On peut voir que malgré l'utilisation d'heuristiques pour tenter d'améliorer et de contrôler notre modèle, nous n'obtenons pas forcément de meilleurs résultats. Néanmoins, l'atout a été de diminuer grandement le temps d'exécution qui d'en certains cas peut s'avérer être très important. De plus, on voit (ref 5) qu'avec l'utilisation de la batch normalization, notre loss globale a grandement diminué au cours des epochs, atteignant des valeurs plus faibles qu'avec d'autres modèles expérimentés.

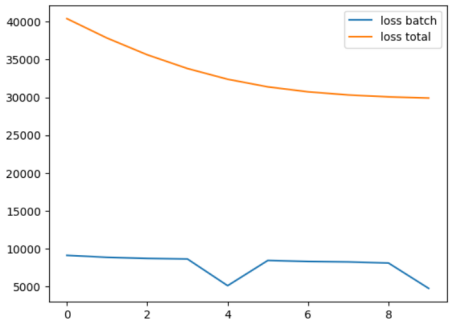

Figure 5: Evolution de la loss totale et des batchs selon les epochs pour l'heuristique de batch normalization.

Nous pouvons faire plus ou moins le même constat avec l'accuracy du modèle utilisant la batch normalization. Celle-ci augmente grandement au fil des epochs, ce qu'on peut voir sur le graphique ci-dessous (6), bien qu'elle soit toujours assez faible.

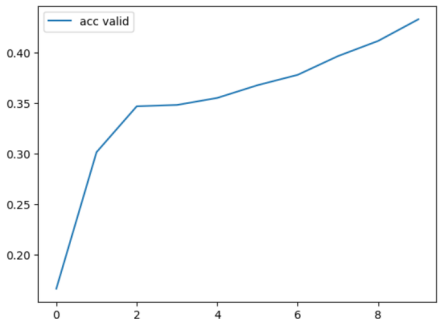

Figure 6: Evolution de l'accuracy de validation au cours des epochs.

Le détail des expérimentations et des différents tests se trouvent sur https://github.com/FeckNeck/deep-learning-project dans le notebook intitulé 'index.ipynb'.

5 Conclusion

Dans ce projet, nous avons travaillé sur la classification de genres musicaux à partir de différentes caractéristiques. Nous avons utilisé un MLP (Multi-Layer Perceptron) avec différentes architectures pour effectuer cette tâche. Nous avons commencé par un MLP à une seule couche cachée, puis nous avons exploré des modèles avec deux, quatre et cinq couches cachées, plusieurs fonctions d'activiation et types d'optimisation. Nous avons également utilisé des techniques telles que le gradient clipping, la batch normalization ou le dropout pour améliorer les performances du modèle.

Nous avons entraîné et évalué nos modèles sur un ensemble de données de musique comprenant six genres différents. Les résultats obtenus ont montré que l'ajout de couches cachées n'ont pas amélioré notre modèle mais que l'utilisation de différentes fonctions d'activation ont conduit à une légère amélioration des performances du modèle. Cependant, nous n'avons pas obtenu des résultats significativement meilleurs avec des modèles plus complexes mais nous avons tout de même pu améliorer les temps de calcul.

En conclusion, ce projet nous a permis de mettre en pratique les concepts de classification et d'apprentissage automatique sur un problème de classification de genres musicaux. Bien que nous n'ayons pas atteint des performances exceptionnelles, ce projet nous a donné une bonne compréhension des défis et des techniques associés à l'utilisation de réseaux de neurones simples. Il offre également des opportunités d'amélioration et d'exploration ultérieures, telles que l'utilisation de modèles plus avancés ou l'exploration de diverses caractéristiques que nous n'avons pas exploité. À noter que notre choix de variable à prédire était pûrement personnelle et que le jeu de données n'est pas prévu pour cet effet au départ.