# Classification du genre de musiques Spotify

Edouard Chappon

December 1, 2024

**Abstract**

Cet article présente une étude de la classification des genres de playlists musicales en combinant des approches d'apprentissage automatique classiques (Support Vector Classifier, Random Forest) avec des techniques d'apprentissage profond (Multi-Layer Perceptron et Transformers). La méthodologie inclut une exploration et un prétraitement des données, ainsi qu'une optimisation des hyperparamètres via validation croisée. Les performances des différents modèles sont évaluées, offrant une comparaison complète des approches utilisées.

## 1 Introduction

À l'ère du numérique, les plateformes de streaming musical ont révolutionné la manière dont les utilisateurs accèdent et créent leurs playlists. Avec un volume croissant de titres disponibles, la classification efficace des genres de playlists devient cruciale pour améliorer l'expérience utilisateur, offrir des recommandations personnalisées et optimiser les stratégies de marketing ciblé. La capacité à catégoriser précisément les genres de playlists permet non seulement d'affiner les suggestions musicales, mais aussi de mieux comprendre les préférences des auditeurs et les tendances émergentes dans l'industrie musicale.
Cette étude se concentre sur la classification des genres de musiques en exploitant un ensemble de données contenant à la fois des caractéristiques numériques (telles que le tempo, la danceability) et des variables catégorielles (comme la track name, la playlist name). En combinant l'encodage *Sentence Transformers*, avec la normalisation des données numériques, cette recherche vise à maximiser la pertinence et la représentativité des caractéristiques utilisées pour la classification.
Tous les scripts ont leur version classique qui n'inclut que les données numériques et leur version `_full` qui inclut également les données textuelles.

La méthodologie adoptée dans cette étude est structurée en plusieurs phases clés.

1. **Exploration et Prétraitement des Données** : À l'aide du script `explore_data.py`, une analyse des données a été réalisée pour comprendre la distribution et les relations entre les différentes features. Le script `preprocess_data.py` s'occupe de la normalisation des données numériques.

2. **Entraînement des Modèles et Optimisation des Hyperparamètres** : 3 modèles distincts ont été entraînés sur les features numériques et un modèle sur les features numériques et textuelles :

   Données numériques et textuelles : Multi-Layer Perceptron (MLP)

   Données numériques seulement : Random Forest, Support Vector Classifier (SVC)

   Chaque modèle a fait l'objet d'une optimisation des hyperparamètres via validation croisée pour garantir des performances robustes et généralisables.

3. **Évaluation et Prédiction** : Un jeu de test a été fourni avec les données d'entraînement. Les scripts predict_mlp.py et predict_mlp_full.py créent les fichiers predictions/CHAPPON_prediction_base.csv et predictions/CHAPPON_prediction_full.csv tels que demandés dans l'énoncé du projet.

Notre étude est organisée comme suit : **Section 2 : Jeu de Données** – On analyse le jeu de données et on justifie les choix faits par la suite.
**Section 3 : Méthodologie** – Détail des architectures de modèles testées, des variantes explorées, ainsi que des protocoles d'apprentissage et d'évaluation mis en œuvre.
**Section 4 : Résultats et Discussion** – Détails et discussion des résultats obtenus et de la qualité des apprentissages.
**Section 5 : Conclusion et Perspectives** – Résumé des principaux constats, discussion sur la validité de la tache accomplie par rapport aux données et propositions pour des recherches futures.

# 2 Jeu de Données

21428 données de train, 5357 données de test. La répartition des 5 classes du label du jeu d'entraînement est présentée dans **Table 1.**

Table 1: Distribution des Genres dans les Ensembles d'Entraînement

| Genre | Total_Train | Subset_Train |
|---|---|---|
| Rap | 4594 | 239 |
| Pop | 4406 | 167 |
| R&B | 4345 | 206 |
| Latin | 4122 | 218 |
| Rock | 3961 | 170 |

Table 2: Nombre d'Éléments Uniques par Feature Catégorielle

| Feature | Unique Values |
|---|---|
| track_name | 977 |
| track_album_name | 965 |
| track_album_release_date | 673 |
| playlist_name | 324 |
| playlist_id | 339 |
| playlist_subgenre | 20 |

La répartition est considérée comme équilibrée. On sélectionne les 1000 premières données afin de réduire le temps d'entraînement des modèles. Nous avons analysé le nombre de classes uniques pour les six caractéristiques catégorielles textuelles. Les résultats sont présentés dans la **Table 2.**
Il y a 13 features numériques (certaines sont binaires comme le mode). La corrélation est calculée les caractéristiques numériques et le label sur la **Figure 1.**

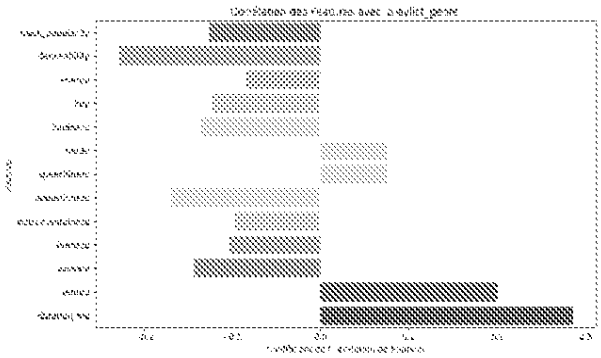

Figure 1: Corrélation des Features avec le label

## 2.1 Traitement des données textuelles

Ces données ne sont utilisées que pour l'entraînement d'un MLP.
Aucune des caractéristiques catégorielles n'est directement exploitable sous forme d'encodage classique (comme l'encodage *one-hot*), car les catégories associées sont quasiment uniques. De plus, lors du test, nous risquons de rencontrer des catégories qui n'étaient pas présentes dans le jeu d'entraînement, rendant nos modèles inapplicables si nous utilisons des techniques d'encodage traditionnelles.

Pour exploiter ces données textuelles, nous avons décidé d'utiliser des techniques d'encodage `Sentence Transformers`, pour transformer les textes en vecteurs numériques de dimension fixe (384). Cela permet de capturer la sémantique des textes et de généraliser aux nouvelles catégories lors du test. Nous avons également exclu 'playlist_subgenre' car utiliser le sous-genre pour prédire le genre ne fait pas de sens.
Pour chaque échantillon, les textes des différentes caractéristiques ont été concaténés pour former une seule chaîne de caractères.
Puis, le modèle *all-MiniLM-L6-v2* de *Sentence Transformer* a été utilisé pour encoder les textes combinés en vecteurs de dimension 384. Ce modèle est préentraîné sur des données larges et est capable de générer des embeddings de phrases qui capturent efficacement les significations sémantiques.
Les données ont donc 397 features en ajoutant les données numériques.

# 3 Méthodologie

Pour tous les modèles, nous avons fait une validation croisée avec 4 folds. Les labels sont encodés avec la classe LabelEncoder de sklearn. Le modèle avec le meilleur score moyen durant la validation croisée est réentraîné sur le jeu d'entraînement complet puis enregistré pour être exploité par la suite sur de nouvelles données.

Pour les modèles de machine learning (SVC et RF), les modèles de sklearn prédisent des valeurs entre 1 et 5 pour chacune des classes.
Pour le modèle MLP, nous utilisons l'optimiseur ADAM et la fonction de perte cross entropy de sklearn. La sortie des modèles renvoie des vecteurs de taille 5 par échantillon. Chaque dimension représente une classe. La Cross Entropy Loss combine `LogSoftmax` et `Negative Log-Likelihood Cross Entropy` :

$$\text{Log-Loss} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{C} y_{i,j}\log(\hat{y}_{i,j}) \tag{1}$$

où $N = 1000$ est le nombre d'éléments dans le jeu d'entraînement, $C = 5$ est le nombre de classes, $y_{i,j}$ indique si l'échantillon $i$ appartient à la classe $j$, et $\hat{y}_{i,j}$ est la probabilité prédite pour $i$ dans la classe $j$.
Ainsi, nous prédisons la classe d'un échantillon en prenant celle correspondant à la valeur la plus élevée dans le vecteur que l'on obtient après avoir appliqué le MLP aux features.

# 4 Résultats et Discussion

## 4.1 Organisation des résultats sauvegardés et description de la structure des résultats discutés

Tous les résultats obtenus pour chacun des modèles sont enregistrés dans `data/results/<nom_du_modèle>`. Le modèle ayant le meilleur score moyen sur les 4 folds et réentraîné sur le jeu de train est le fichier .pkl. Les hyperparamètres et les accuracies moyennes de train et de validation sont enregistrés dans `best_hyperparametres.json`

Les métriques d'entraînement se trouvent dans le fichier `train_metrics.json` où l'on trouve, pour SVC et RF, l'accuracy d'entraînement et de validation pour toutes les combinaisons d'hyperparamètres et les 4 folds. Pour MLP et MLP full, on y trouve les valeurs de training et validation loss ainsi que les valeurs d'accuracy de train et validation à chaque époque pour chaque combinaison d'hyperparamètres pour les 4 folds.

Pour les modèles SVC et RF, les accuracies de test et train moyennes pour chaque combinaison d'hyperparamètres sont tracées et enregistrées dans le fichier `metrics_evolution_<nom_du_modèle>.png` comme en figure 2. Il faut se référer au fichier `train_metrics.json` pour obtenir la configuration associée à chaque numéro de configuration.

Pour les modèles MLP et MLP full, on a également `metrics_evolution_<nom_du_modèle>.png`. On a aussi tracé toutes les courbes de training et test loss pour chacun des 4 folds pour chaque combinaison d'hyperparamètres dans le fichier `loss_plot.png`. On trouve sur chaque colonne de l'image les losses de 2 modèles, ce qui fait 8 graphiques par colonne comme dans la figure 8. Le nom de la configuration est écrit en titre du premier graphique de la configuration.

Pour tous les modèles et toutes les valeurs uniques d'hyperparamètres, nous avons calculé les performances moyennes de train et de test afin de déterminer l'impact d'un hyperparamètre sur les performances. Ces tables sont enregistrées dans `hyperparamètres_analysis.csv`, ce qui permet de gérer des tables comme la Table 3.

## 4.2 SVC

Meilleur modèle : C: 1, kernel: rbf
Accuracy moyenne : 49,1%

Table 3: Analyse des Hyperparamètres

| **Hyperparamètre** | **Accuracy Moyenne Train** | **Accuracy Moyenne Test** |
|---|---|---|
| kernel=linear | 0.5038 | 0.4749 |
| C=1.0 | 0.6171 | 0.4693 |
| C=2.0 | 0.6713 | 0.4663 |
| C=0.8 | 0.5989 | 0.4640 |
| C=5.0 | 0.7362 | 0.4627 |
| kernel=rbf | 0.8438 | 0.4501 |
| C=0.4 | 0.5476 | 0.4500 |
| C=10.0 | 0.7739 | 0.4500 |
| C=20.0 | 0.8013 | 0.4437 |
| C=30.0 | 0.8154 | 0.4400 |
| C=40.0 | 0.8212 | 0.4377 |
| C=50.0 | 0.8259 | 0.4367 |
| C=60.0 | 0.8280 | 0.4360 |
| C=70.0 | 0.8300 | 0.4347 |
| C=80.0 | 0.8319 | 0.4337 |
| C=90.0 | 0.8328 | 0.4327 |
| C=100.0 | 0.8336 | 0.4317 |
| C=0.2 | 0.4996 | 0.4270 |
| kernel=poly | 0.8243 | 0.3990 |
| C=0.1 | 0.4427 | 0.3867 |

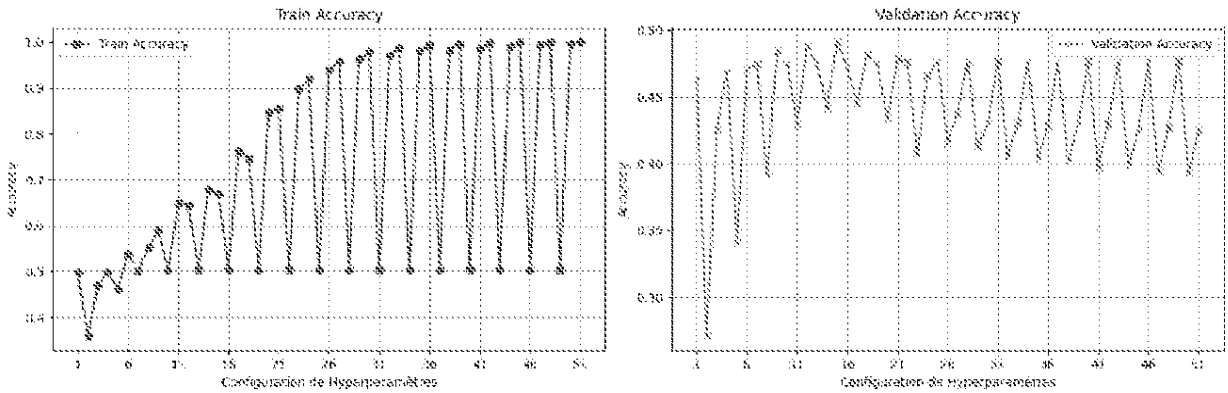


Figure 2: Accuracy de train et de validation en fonction de la configuration des hyperparamètres

Les configurations ayant un numéro congru respectivement à 1, 2 et 0 modulo 3 correspondent à un kernel respectivement linéaire, polynomial et rbf. Les pics vers le bas autour de 0,5 sur le graphique de train de la figure 2 correspondent aux modèles linéaires, alors que les pics vers le bas dans les scores de validation correspondent au kernel polynomial. On en déduit un overfitting des modèles polynomiaux. Les kernels linéaires et rbf ont des performances de validation proches, mais rbf atteint des scores quasiment parfaits sur les folds d'entraînement en raison de la flexibilité des frontières de décision qu'il permet, tandis que le kernel linéaire ne permet que des séparations linéaires dans l'espace des features.

## 4.3 Random-Forest

Meilleur modèle : max_depth: 20, min_samples_leaf: 4, min_samples_split: 5, n_estimators: 50
Accuracy moyenne : 49,9%
On voit dans la table 4 que tous les modèles ont des scores très proches.

Table 4: Analyse des Hyperparamètres

| Hyperparamètre | Accuracy Moyenne Train | Accuracy Moyenne Test |
|---|---|---|
| min_samples_leaf=8 | 0.7322 | 0.4901 |
| n_estimators=100 | 0.7269 | 0.4867 |
| max_depth=20 | 0.7697 | 0.4864 |
| n_estimators=150 | 0.7291 | 0.4863 |
| min_samples_leaf=4 | 0.8042 | 0.4856 |
| max_depth=10 | 0.7639 | 0.4854 |
| min_samples_split=15 | 0.7136 | 0.4845 |
| min_samples_split=5 | 0.7335 | 0.4845 |
| min_samples_split=10 | 0.7282 | 0.4838 |
| max_depth=5 | 0.6417 | 0.4809 |
| n_estimators=50 | 0.7192 | 0.4799 |
| min_samples_leaf=16 | 0.6389 | 0.4771 |

On a la figure 3 avec les scores de toutes les configurations.
On voit sur l'accuracy de train, 3 parties distinctes. Le premier tiers des configurations, qui obtiennent des scores entre 0,60 et 0,65 correspondent à une profondeur d'arbre maximale de 5. Les deux autres tiers qui ont des formes semblables correspondent à des profondeurs de 10 et 20. Les deux pics correspondent à un minimum de samples par feuille de 4, les premiers plateaux correspondent à une valeur de 8 et les seconds plateaux à une valeur de 16. Ces résultats sotn en accord avec la colonne de train de la table 4.

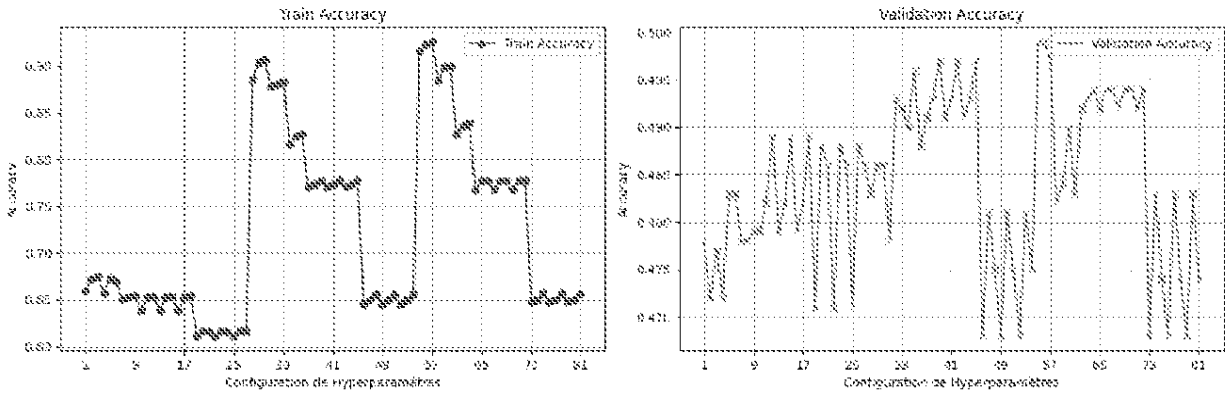


Figure 3: Accuracy de train et de validation en fonction de la configuration des hyperparamètres

## 4.4 MLP

Meilleur modèle : hidden_layers: [ 32 ] , batch_size: 256, epochs: 150, learning_rate: 0.005
Accuracy moyenne : 48,3%

La figure 4 représente les losses d'entraînement et de validation pour les 4 folds du meilleur modèle. Les losses de tous les modèles entraînés sont disponibles en Annexe (figure 8).
Les modèles avec une seule couche cachée atteignent de meilleurs scores que ceux avec 2 couches.

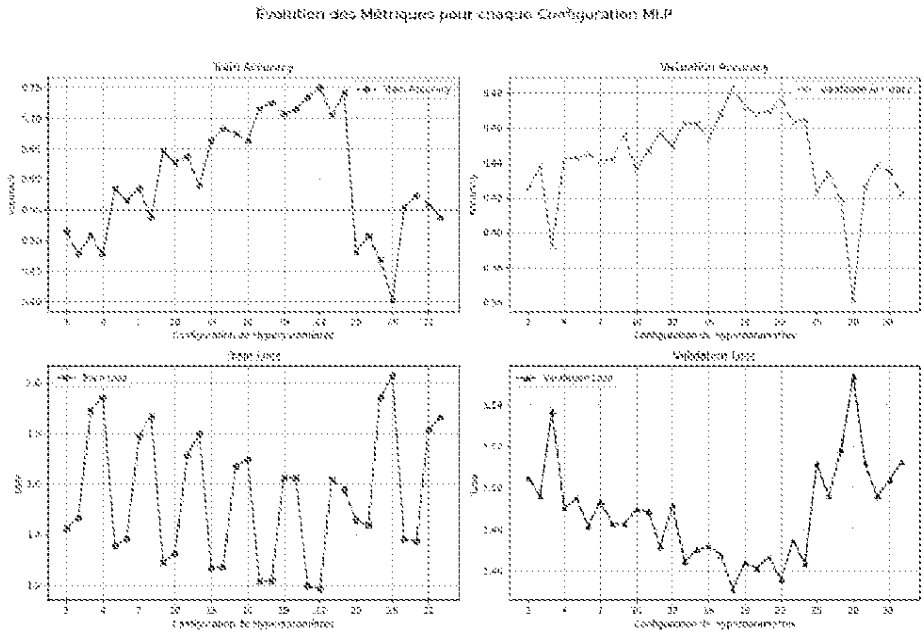


Figure 4: Accuracy de train et de validation en fonction de la configuration des hyperparamètres

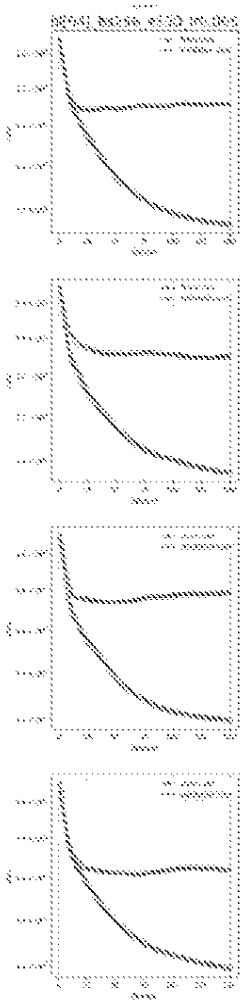

Figure 5: Loss du meilleur modèle MLP

Table 5: Analyse des Hyperparamètres pour les Paramètres d'Entraînement

| **Hyperparamètre** | **Accuracy Moyenne Train** | **Accuracy Moyenne Test** |
|---|---|---|
| hidden_layers=(32) | 0.715 | 0.473 |
| hidden_layers=(64) | 0.733 | 0.468 |
| hidden_layers=(16) | 0.671 | 0.457 |
| hidden_layers=(8) | 0.625 | 0.449 |
| batch_size=256 | 0.613 | 0.448 |
| learning_rate=0.05 | 0.609 | 0.443 |
| learning_rate=0.005 | 0.598 | 0.445 |
| hidden_layers=(4) | 0.569 | 0.443 |
| batch_size=512 | 0.594 | 0.440 |
| hidden_layers=(4, 4) | 0.556 | 0.431 |
| hidden_layers=(2) | 0.494 | 0.424 |
| hidden_layers=(2, 2) | 0.464 | 0.409 |

Le nombre de paramètres augmente de manière croissante par rapport au numéro de la configuration entre la configuration 1 et 24. On voit que l'accuracy de validation est globalement croissante par rapport au nombre de paramètres.
Le patern périodique dans les valeurs de la loss lors de l'entraînement est dû à la taille des batchs. Les configurations de numéro congru respectivement à 1 et 2 modulo 4, qui ont des valeurs plus basses, ont un batch size de 256 et celles congrues à 0 et 3 modulo 4 ont un batch size de 512.

## 4.5 MLP full

Meilleur modèle : hidden_layers: [ 8 ] batch_size: 256, epochs: 150, learning_rate: 0.005
Accuracy moyenne : 94,7%

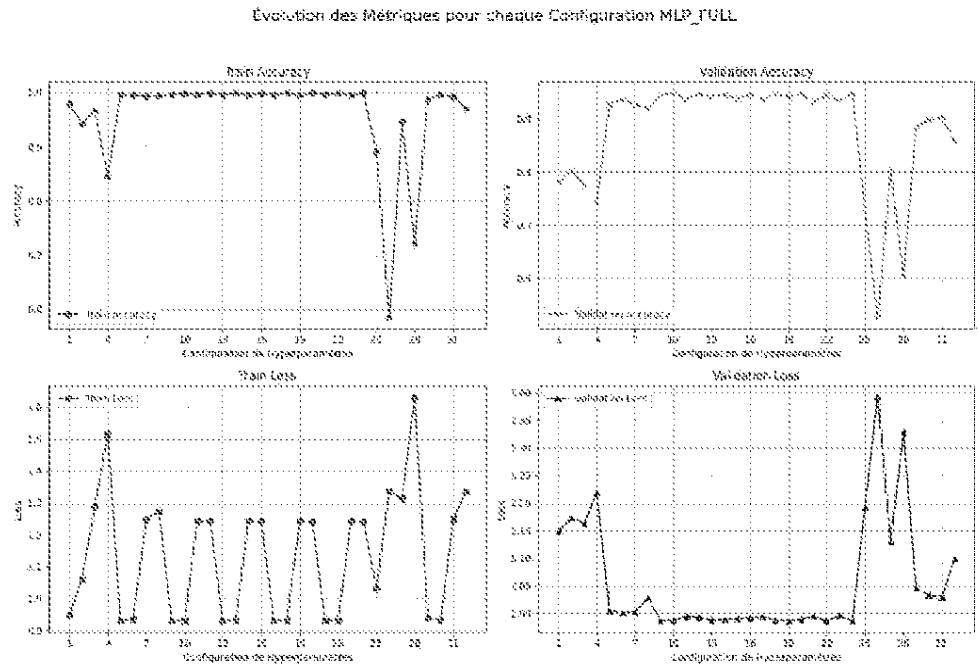


Figure 6: Accuracy de train et de validation en fonction de la configuration des hyperparamètres

| Hyperparamètre | Accuracy Moyenne Train | Accuracy Moyenne Test |
|---|---|---|
| hidden_layers=(8,) | 0.996 | 0.943 |
| hidden_layers=(32,) | 0.996 | 0.942 |
| hidden_layers=(16,) | 0.996 | 0.941 |
| hidden_layers=(64,) | 0.996 | 0.938 |
| hidden_layers=(4,) | 0.993 | 0.926 |
| learning_rate=0.05 | 0.981 | 0.889 |
| hidden_layers=(4, 4) | 0.985 | 0.884 |
| batch_size=512 | 0.962 | 0.878 |
| batch_size=256 | 0.958 | 0.876 |
| learning_rate=0.005 | 0.939 | 0.865 |
| hidden_layers=(2,) | 0.932 | 0.774 |
| hidden_layers=(2, 2) | 0.785 | 0.666 |

Table 6: Analyse des Hyperparamètres

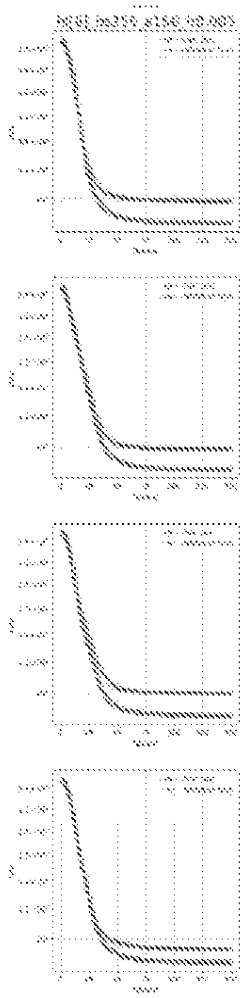

Figure 7: Loss du meilleur modèle MLP_full

La Table 7 représente les losses d'entraînement et de validation pour les 4 folds du meilleur modèle. Les losses de tous les modèles entraînés sont disponibles en Annexe (figure 9).
On voit que l'ajout des données textuelles permet une prédiction quasiment parfaite du genre de la musique. Ceci est certainement dû à la feature "playlist_name" qui donne beaucoup d'informations sur le genre de la musique.
On retrouve, sur les loss de train, le même patern qu'avec les modèles MLP qui est dû au batch size.

## 4.6 Comparaison

Les résultats présentés dans la Table 7 indiquent que le modèle MLP_full atteint une accuracy moyenne de 94,7%, surpassant nettement les autres modèles. Ceci met en évidence l'apport des données textuelles encodées via *Sentence Transformers* dans la classification des genres musicaux. Les modèles basés uniquement sur les caractéristiques numériques, tels que le SVC, le Random Forest et le MLP, affichent des performances similaires, avec des accuracies moyennes autour de 49%. Cela suggère que les seules données numériques ne suffisent pas pour une classification précise, et que l'inclusion des informations textuelles est essentielle pour améliorer les performances du modèle.

Table 7: Comparaison des Modèles

| **Modèle** | **Hyperparamètres Optimaux** | **Accuracy Moyen** |
|---|---|---|
| SVC | C=1, kernel='rbf' | 49,1% |
| Random Forest | max_depth=20, min_samples_leaf=4, min_samples_split=5, n_estimators=50 | 49,9% |
| MLP | hidden_layers=[32], batch_size=256, epochs=150, learning_rate=0,005 | 48,3% |
| MLP_full | hidden_layers=[64], batch_size=256, epochs=150, learning_rate=0,005 | **94,7%** |

# 5 Conclusion et Perspectives

Le jeu de données avec uniquement les données numériques ne permet pas d'obtenir une accuracy de plus de 50%. Ces features ne sont pas assez corrélées au genre des musiques. Les features textuelles permettent d'obtenir un bon score de prédiction, mais on peut se demander si c'est réaliste de les considérer comme acquises. On a supprimé le sous-genre de la musique puisque c'est comme si l'on devait déterminer le type de légume, exemple : carotte, à partir du sous-type, exemple : carotte des sables, comme on l'a discuté dans la partie 2.1.
Les features textuelles contiennent aussi le nom de la playlist. Cette feature est très informative et contient souvent le nom du genre de la musique. La cohérence de prendre cette information dépend du processus d'acquisition de la donnée. Si les données sont récupérées sur un site via le téléchargement d'informations musicales contenues dans des bases de données rangées par playlists, alors considérer que l'on a accès au nom de la playlist peut faire sens. Mais des techniques de NLP plus classiques et moins coûteuses pourraient alors certainement suffire à obtenir une classification performante.

Une autre raison de nous interroger sur l'intérêt d'utiliser des réseaux de neurones pour résoudre cette tâche est que l'on a les mêmes résultats avec les features numériques, avec ou sans réseaux de neurones, mais ces réseaux, à la différence de SVC et de random forest, ne permettent pas d'interpréter les résultats et de juger de la pertinence du processus de prédiction. Il faudrait voir si les méthodes de ML classiques obtiennent des scores similaires à MLP_full si l'on ajoute les données textuelles. Ce travail est laissé à de prochaines investigations.
Un second désavantage des réseaux de neurones est qu'ils nécessitent un entraînement coûteux et le choix de beaucoup plus d'hyperparamètres que les méthodes de ML classiques.

# A Annexe

Il faut un peu zoomer ...

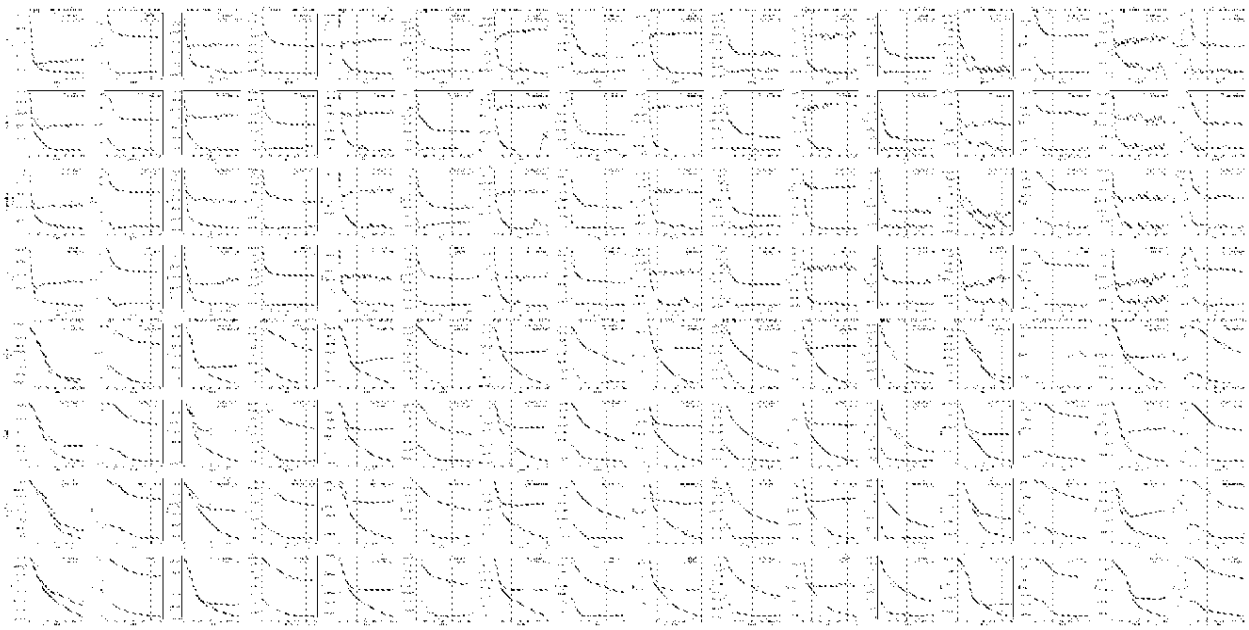

Figure 8: Loss de train et de validation de tous les folds et configurations de MLP

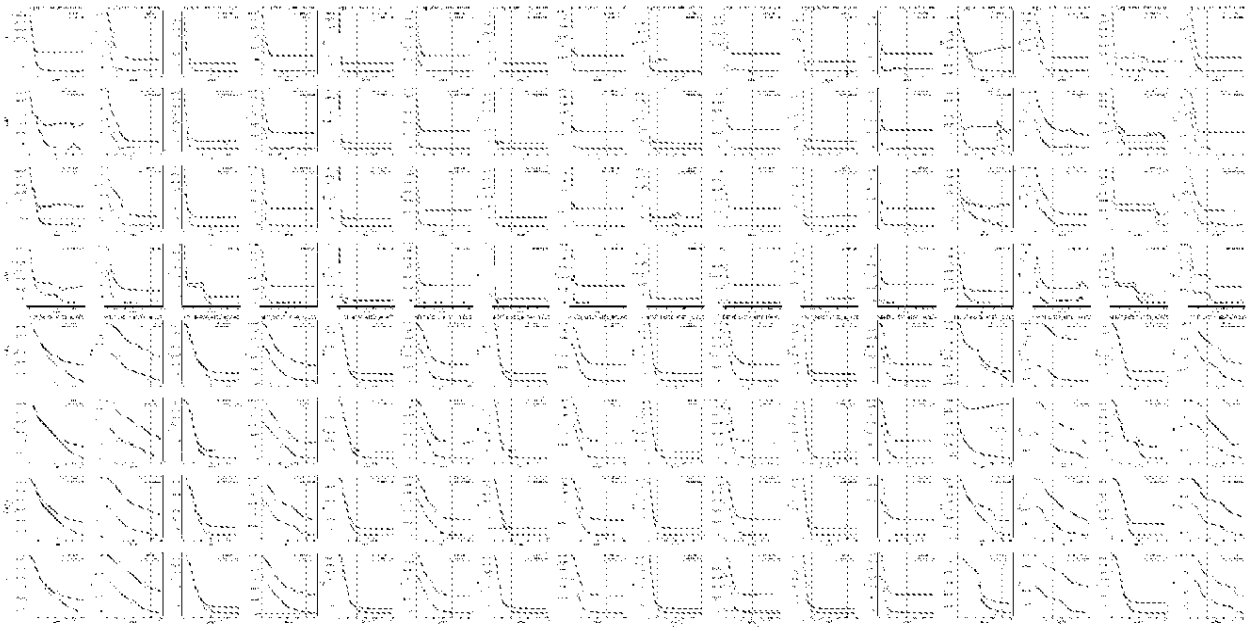

Figure 9: Loss de train et de validation de tous les folds et configurations de MLP full