Rapport de projet - Prédiction de fins de phrases musicales

Loïcia Robart & Gaspar Henniaux

github (https://github.com/pargass/projet_fin_de_phrases_musicales/tree/main)

Voici notre rapport pour le projet de Prédiction de fins de phrases musicales. Il y a un notebook associé qui retrace notre cheminement de pensée et notre démarche. Il est intéressant d'avoir le rapport et le notebook sous les yeux pour visualiser notre démarche.

Analyse préliminaire du projet

Nous avons commencé par bien étudier les données. Nous avons lu les descriptions des différents attributs donnés dans le dataset pour avoir une vision globale des types de données disponibles et ce qu'elles représentent en terme de musique.

On comprend que chaque ligne du jeu de données correspond à une mélodie, et chaque colone à une caracteristique de cette mélodie.
Dans une mélodie/chanson, on retrouve plusieurs phrases et donc des départs et fin de phrase. Dans chacune des séquences (mélodies), on peut trouver plusieures notes correspondant à des fins de phrase (une note marquée comme une fin de phrase est la dernière note d'une phrase mélodique).
Nous avons ensuite réfléchis sur la notion de "fin de phrase" en terme de classification. Cela nous a permis de trouver des attributs tels que phrase_end, phrasepos, beatinphrase et beatinphrase_end qui pourraient nous aider dans cette classification.

Une autre particularité des données est que la plupart des attributs ont des listes pour valeur. Ces listes correspondent en réalité à une représentation de la séquence note par note. Pour prendre un exemple concret et en reprenant notre attribut cité plus haut 'phrase_end', on peut afficher, pour la première mélodie du jeu de donnée, sa valeur (df_phrase['phrase_end'][0]). On récupère alors une liste de booléens, chaque booléen indique si cette note est considérée comme une fin de phrase. Il faudra donc trouver une manière de gérer ces données sous forme de liste.
Chaque liste a donc une taille équivalente au nombre de notes dans la séquence.

PROF

Nous avons affiché pour une certaine ligne l'ensemble de ses valeurs par attribut pour avoir une vision d'ensemble.
Pour visualiser les différentes valeurs possibles que les attributs peuvent contenir, nous avons bouclé sur toutes les lignes du dataset et ajouté au fur et à mesure chaque nouvelle valeur qui aparaissait dans une liste (voir notebook). Cela nous permettait d'avoir une meilleure vision sur l'ensemble de valeurs qu'un attribut peut prendre.

Préparation des données

Tri manuel dans les features

De nombreux atributs sont directement liés les uns aux autres (voir détail dans le notebook), par exemple, duration, duration_frac et duration_fullname, qui se distinguent simplement par des différences dans la notation. On peut alors ne s'en tenir qu'à un seul. Ces corrélations entre attributs sont déductibles depuis la documentation fournie sur les features.
D'autres attributs de par leur nature n'ont rien à voir avec les placements de fin de phrase et peuvent donc directement être évincés. 'tonic' et 'mode' n'ont pas été gardés car il ne donnent pas d'informations sur les fin de phrase car ils ont la même valeur tout au long d'une même séquence.

Ce pré-tri nous a permis de sortir la liste d'attributs réduite suivante : 'midipitch', 'chromaticinterval', 'scaledegree', 'timesignature', 'beatstrength', 'metriccontour', 'imaweight', 'imacontour', 'duration', 'durationcontour', 'beatfraction', 'beat', 'restduration_frac', 'phrase_end'. (voir code : **liste des features corrélées selon la doc**)

Découpage en sous-séquences

L'objectif de notre modèle sera de prédire si une sous-séquence est une fin de phrase. Notre modèle doit donc s'entrainer sur des sous-séquences de notes qui sont pour certaines des fin de phrase (target à 1) et d'autres non (target à 0) (voir choix d'étiquetage plus bas).
Nous avons dans un premier temps réfléchi à un découpage basé sur la notion de mesures en musique (une unité de temps qui organise les rythmes dans une composition musicale). En effet, une phrase musicale s'étend généralement sur un certain nombre de mesures.
Les fins de phrases musicales coïncident souvent avec la fin d'une ou plusieurs mesures, renforçant le sentiment de structure, mais cela dépend du style et de la composition.

Il serait également possible de s'imaginer diviser chaque séquence par le nombre de fin de phrases de la séquence. Si notre séquence contient 5 notes marquées à 1 pour l'attribut 'phrase_end', alors on divise toutes les notes de cette séquences en 5 pour créer 5 sous-séquences mélodiques. Mais l'on obtientrait uniquement des sous-séquences qui sont des fins de phrase alors que nous avons besoin également de sous-séquences qui n'en sont pas pour entrainer nos modèles. De plus, nous aurions des sous-séquences de taille différente.

Une autre approche est de définir un nombre de notes fixe, et de diviser chaque mélodie en plusieurs sous-séquences au nombre de notes égal à ce nombre fixé.

PROF

La problématique d'aplatissement des données (voir aplatissement des données plus bas) qui sont sous forme de listes nous contraint de choisir cette seconde solution de découpage en sous-séquences par nombre de notes fixe.
Au départ nous avions fixé notre nombre de notes à 8 mais nous sommes revenus en arrière pour tester l'impact de différentes valeurs pour ce choix de longueur de sous-séquence.
Ce nombre de notes par sous-séquences peut ainsi être vu comme un paramètre à ajuster lors de nos apprentissages pour optimiser les performances des modèles. Nous avons pu ainsi tester des sous-séquences avec un total de 4 à 20 notes. (voir code **generate_subsequences**)

Vers la fin de nos apprentissages de modèles nous avons remarqué qu'avec cette approche de division de séquences de notes, si l'on divisait en commençant par prendre le début de notre séquence et en étiquettant les sous-séquences selon si la dernière note est une fin de phrase selon l'attribut 'phrase_end' (le détail de l'étiquettage se trouve plus bas), on perdait alors beaucoup d'exemples de classe 1 (fin de phrase) et cela augmentait le déséquilibre de notre dataset. En effet le nombre de notes

d'une séquence n'est pas nécessairement divisible par notre taille de notes de sous-séquences. Ainsi, en effectuant notre division de séquence, on ne gardait que les sous-échantillons avec la taille de notes souhaitée. Cela nous faisait perdre énormément de fins de séquence qui correspondaient aux nombre de notes restantes après notre division. Hors ces fins de séquences correspondent à des fin de phrases musicales. Nous perdions ainsi de nombreux échantillons qui auraient été étiquetés positif comme fin de phrase.
Ainsi, pour améliorer l'apprentissage de nos modèles qui s'entrainaient avec beaucoup plus de sous-séquences étiquetées à 0 qu'à 1, nous avons inversé notre méthode de division. De ce fait, nous avons essayé de diviser en commençant par la fin de la séquence. En prenant donc par exemple les 8 dernières notes de chaque séquences puis les 8 suivantes avant celles-ci, etc. Cela a grandement augmenté notre nombre de sous-séquences qui étaient des fin de phrases, et donc nos scores de prédiction à la fin. (voir code **generate_subsequences(reverse=True)**)

Choix d'étiquetage

L'étiquetage consiste ici à décider pour chacune de nos sous-séquence si sa classe (colonne target -> la valeur que notre modèle devra prédire) est 1 (sous-sequence correspondant à une fin de phrase) ou 0 (la sous-séquence n'est pas une fin de phrase).

La notion de séquence musicale qui correspond à une fin de phrase était assez floue pour nous au début. Nous avons ainsi décidé de tester plusieures approches.
Notre première idée à été d'étiqueter chaque sous-séquence contenant au moins une note marquée comme fin de phrase (phrase_end à true) à 1. Si la sous-séquence ne contient aucune fin de phrase parmis toute ses notes, la classe est notée à 0. Nous l'avons appelé 'any' (voir code **transform_target_any**)

Après quelques calculs de scores moyens depuis cet étiquetage, cette approche ne nous semblait pas optimale en terme de logique. En effet si la note marquant la fin de phrase est en début de sous-séquence, on peut considérer que toutes les notes suivantes dans cette sous-séquence correspondent en réalité à une progression de note pour le début de la phrase musicale suivante. Cela brouille l'apprentissage car la sous-séquence aurait quand même été considérée comme une fin de phrase alors que la majorité des notes dans cette séquence sont en réalité en lien avec un début de phrase.

PROF

Nous avons donc décidé de revenir en arrière pour étiqueter nos sous-séquences différement. Pour ce faire, on ne marque à 1 une sous-séquence seulement si la dernière note de celle-ci est elle-même marquée comme une fin de phrase (selon l'attribut 'phrase_end'). Nous l'avons appelé 'end' (voir code **transform_target_end**)

A la fin, nous sommes une nouvelle fois revenus sur cette étape pour tester l'impact de ce choix d'étiquetage de manière plus précise. Cela a été possible grâce à un étiquetage "hybride". Nous avons fait varier le nombre de note de la sous-séquence à prendre en compte pour regarder si une fin de phrase s'y trouvait. Cela nous donne un nouveau paramètre à ajuster pour tester des scores, nous avons essentiellement essayé jusqu'à 4. Nous l'avons appelé 'hybrid' (voir code **transform_target_hybrid**)

Nous avons retiré l'attribut 'phrase_end' et ses équivalents de nos attributs dans notre dataframe pour l'apprentissage des modèles. Cela aurait été équivalent à donner la réponse au modèle et lui demander de prédire cette même réponse.

Transformation des données

Une partie des données n'était pas en valeurs numérique, il a donc fallu transformer les attributs en question pour n'avoir à la fin que des valeurs numériques dans le dataframe. (voir code : **transform_features**)

Les règles sont les suivantes :

- 'chromaticinterval': remplace None par 0.
- 'timesignature': convertit les fractions en nombres décimaux.
- 'beatfraction': convertit les fractions en nombres décimaux.
- 'metriccontour', 'imacontour', 'durationcontour': transforme les chaînes '+' en 1, '-' en -1 et '=' en 0.
- 'restduration_frac': convertit les fractions en nombres décimaux, remplace None par 0.

L'intêret de convertir une valeur de type string (une valeur catégorielle) en vecteur binaire, vient du soucis de ne pas attribuer de poids à certaines valeurs. En effet, si on prend par exemple l'attribut 'mode', qui peut prendre une valeur parmis : 'major', 'minor', 'dorian', 'phrygian', 'lydian', 'mixolydian'. Si on avait remplacé chaque mode par un numéro, cela aurait implicitement engendré un ordonnancement entre les modes. Pour rester neutre, il vaut mieux représenter toutes les possibilités dans un vecteur binaire, ici de taille 6, une valeur binaire pour chaque mode selon si il est selectionné (1) ou pas (0).

Applatissement des données

L'applatissement des données est une étape clé pour transformer des structures de données complexes, comme nos colonnes contenant des listes, en un format tabulaire classique adapté à l'apprentissage de modèles de machine learning.
Pour ce faire, dans notre code, on parcourt les différentes colonnes du dataframe et on considère 2 cas :

- l'attribut contient des listes de valeurs uniques, on crée alors un attribut pour chaque valeur.
- l'attribut contient des listes de vecteurs(nos vecteurs binaires par exemple). Dans ce cas, chaque élément de cette liste est extrait et attribué à une nouvelle colonne avec un suffixe d'index (_0, _1,).

PROF

Avec ce traitement, si la longueur de nos sous-séquences est de 8, alors pour chaque attributs, 8 nouveaux seront crées, prenant les valeurs de la liste dans l'ordre. (voir code : **flatten_dataframe**)

Étant donné que la taille des listes dans les colonnes de notre DataFrame correspond au nombre de notes de chaque sous-séquences, il est essentiel que toutes les sous-séquences contiennent le même nombre de notes. Cela garantit que les colonnes du DataFrame restent cohérentes et alignées avec les notes de chaque ligne.
Cela justifie notamment notre choix final de découper nos sous-séquences selon le même nombre fixe de notes, indépendemment des spécificités des différentes séquences d'origine.

Equilibrage des classes

L'intêret de cette étape nous est apparu après le calcul de premières métriques après l'entrainement de nos modèles. A ce moment de notre projet, nous avions alors déjà un très haut score sans beaucoup de traitement, mais cela s'expliquait par le déséquilibre entre nos classes.
En effet, avec notre étiquetage de 1 plus strict (en acceptant que des sous-séquence avec la dernière note marquée comme fin de phrase), on se retrouve avec énormément plus de sous-séquence de classe 0 que de classe 1.
L'effet de classe majoritaire biaise ainsi nos résultats (la classe majoritaire étant ici 0). On le remarque notamment avec un recall (capacité à identifier correctement les instances positives) très bas pour la classe minoritaire. Nos modèles ont tendance à beaucoup prédire la classe majoritaire pour améliorer le score mais on passe à coté de beaucoup d'instances positives. Il est donc crucial de ne pas se concentrer sur le score global uniquement mais de se pencher sur les autres métriques, notemment le recall qui est la proportion des vrais positifs parmi tous les positifs. Le F1 score est une meilleure métrique que le score classique car il est la moyenne harmonique de la précision et du rappel. (voir partie sur les metriques)

Pour contrer ce biais, nous avons rajouté un traitement des données qui consiste à équilibrer les sous-séquences selon les classes. (voir code **balance_classes**). C'est ici que le fait de commencer à prendre les sous-séquences par la fin d'une séquence lors du découpage du dataset en sous-séquences prend son sens.

Utilisation du scaler

Comme dernière étape de notre pré-traitement des données, nous avons également eu recours à un scaler (standardScaler). Un scaler permet de normaliser nos valeurs numériques. Cela permet de mettre à la même échelle les différents attributs et donc de minimiser les biais de convergence des algorithmes, qui auraient pu être induits par des échelles différentes des variables. (voir code **standardScaler()**)
Nous avons ainsi pu effectuer des comparaisons de performance avec et sans utilisation du scaler.

Apprentissage des modèles

Pour mieux comprendre la démarche il suffit de suivre le notebook dans l'ordre de haut en bas.

PROF

Démarche

Une fois le premier tri des données effectué, nos attributs transformés et applatis, nous nous sommes lancé dans l'apprentissage de nos premiers modèles

Dans un premier temps, pour tester et avoir une idée de ce qui fonctionnait et ce qui était cohérent dans notre traitement des données, nous avons essentiellement utilisé un modèle MLPClassifier (Multi-Layer Perceptron). En sachant que c'est un modèle partciulièment efficace pour des problèmes de classification, comme le notre. D'autant plus avec des données complexes, ce qui est le cas dans ce projet. Nous avons également considéré avoir suffisament de données pour l'entrainer. Ce premier choix explique également l'emploi d'un scaler. En effet, ce type de modèle est connu pour être très sensible aux échelles des données en raison de l'utilisation de fonctions d'activation comme sigmoid ou tangente hyperbolique.

Nous avons tout de même essayé avec d'autres modèles comme RandomForest, KNN et LogisticRegression pour avoir un premier aperçu. Bien que nous ayons obtenu un meilleur score avec RandomForest, Nous avons tout de même décider de poursuivre nos recherches avec MLP pour les raisons citées ci-dessus. Nous pensions qu'il était encore possible de mieux traiter les données pour augmenter le score avec MLP.

A partir de ce premier choix de modèle, nous avons pu tester les différents paramètres de notre traitement des données. Notamment comme évoqué plus haut, le choix de la classification, l'étiquetage, la méthode de découpage en sous-séquences (inversé ou non), le taille sous-séquences, l'équilibrage ou non des classes, l'utilisation du scaler. (voir résultats).

A chaque découverte nous vérifions et constations si le score augmentait et comment on pouvait interpréter les différentes métriques. Ainsi nous avons d'abord fait un premier vrai modèle avec un MLP ou nous faisions varier quelques paramètres clés comme la fonction d'activation, le nombre d'itérations ou encore le pas de gradient. (voir **premier vrai modèle** dans le notebook)

Nous avons ensuite remarqué que notre étiquetage n'était pas optimal. Nous avons alors essayé l'étiquetage en prenant en compte uniquement la dernière note de la sous-séquence ce qui a amélioré notre score. Mais nous avons remarqué grâce au recall de la classe minoritaire qu'il y avait un biais dû au déséquilibre des classes. Nous avons alors essayé d'équilibrer les classes ce qui a fait baisser le score global mais a augmenté le recall pour la classe minoritaire. (voir **deuxième modèle avec un autre étiquetage (end)** et **équilibrer les classes** dans le notebook)

Nous avons ensuite voulu tester un étiquetage hybride, c'est à dire qui prend en compte uniquement un certains nombre de note en commançant par la fin pour voir si une fin de phrase s'y trouve. Les deux autres fonctions d'étiquetage déjà trouvées jusqu'à présent ne sont que des cas particuliés de cette troisième (any -> hybrid(toutes les notes), end -> hybrid(une seule note)). après des recherches en faisants varier les tailles de sous-séquences et les tailles de prise en compte pour l'étiquetage nous avons remarqué un meilleur score pour un étiquetage 'end'. Concernant la taille des sous séquences nous sommes restés sur 8. Il y aura plus de détails sur cet aspect dans la partie interprétation (voir interprétation).

PROF

Nous avons ensuite voulu augmenter le nombre d'exemples dans notre matrice d'apprentissage. Pour cela, nous avons décidé de commencer le sous-séquençage par la fin des séquences. en effet chaque fin de séquence est également une fin de phrase. Comme nous voulions des sous-séquences de même taille pour les raisons éxpliquées auparavant, nous amputions la plupart des fins de séquences de notre jeu de données. Cela a augmenté le nombre de données restante après un équilibrage et aussi nos scores.

A la suite de cette première analyse et après avoir obtenu des scores relativement satisfaisants, nous avons souhaité élargir notre apprentissage à d'autres modèles.

Nous avons donc créé une liste avec les modèles à tester, on retrouve :

```
- Logistic Regression
- SVM
- Random Forest
- KNN
```

```
- MLP
- Naive Bayes
- Decision Tree
- SGD
```

Puis en parcourant cette liste de modèles, pour chacun d'eux nous avons créé une pipeline nous permettant d'appliquer le scaler puis l'apprentissage. Nous avons ensuite observé le score moyen pour chaque modèle, grâce à la moyenne des scores calculés avec une cross-validation.
Dans tous nos calculs de score, nous avons fait appel à la cross-validation. Cela permet une évaluation plus fiable du modèle et de sa capacité à généraliser sur de nouvelles données non vues.

A cette étape, randomForest présentait le meilleur score (0.934), suivi de près par SVM (0.931) et MLP (0.928). (voir résultats).

Nous voulions à partir de là mieux prendre en compte l'ajustement des hyperparamètres pour optimiser au maximum les performances de nos modèles. Nous avions déjà manuellement testé, grâce à des boucles, les meilleurs hyperparamètres pour notre modèle MLP. Notamment pour comparer l'emploi de relu face au tanh, la valeur d'alpha, le learning_rate et le nombre d'itérations maximum.

Après avoir vu que random forest semblait être le plus performant, nous avons également utilisé une grid search sur celui-ci pour ajuster les hyperparamètres tel que la profondeur maximale de l'arbre (model__max_depth) ou encore le nombre minimum d'échantillons requis pour qu'un nœud devienne une feuille (model__min_samples_leaf).(voir résultats)

Pour finir nous avons effectuer un grid_search de nouveau sur MLP mais aussi sur RandomForest pour essayer d'en dégager les meilleurs hyper-paramètres possible.

Résultats

Choix des métriques de scores

PROF

Dans un premier temps, nous nous referions simplement aux scores donnés par la fonction 'score' des différents modèles importés de scikit learn. Cette fonction se base sur une métrique par défaut associée à la classe de modèle utilisée. Elle varie donc en fonction du type de problème. Avec des problèmes de classification, la métrique par défaut est l'accuracy (proportion de prédictions correctes effectuées par le modèle par rapport au nombre total d'exemples).
Nous avons rapidement réalisé que cette métrique n'était pas suffisante dans notre contexte de par le déséquilibre dans nos classes (si on ne les équilibre pas). Dans ce cas, le score peut être très élevé bien que le modèle soit peu performant. Nous avons donc regardé d'autres métriques.

Pour évaluer les performances du modèle, et visualiser simplement et succintement un maximum d'informations, nous avons utilisé la fonction classification_report de la bibliothèque Scikit-learn. Cette fonction permet d'obtenir un résumé très complet des performances du modèle pour chaque classe, en calculant plusieurs métriques clés. Cela nous a servi à avoir un premier aperçu des modèles lorsque l'on fait varier beaucoup de paramètres et donc que nous créions beaucoup de modèles.

Nous avons également attentivement regardé le rappel (recall), soit la proportion des vrais positifs identifiés correctement parmi tous les exemples réellement positif. Ici il nous aide à mesurer la capacité

du modèle à détecter les instances de la classe minoritaire. De ce fait on peut vérifier que le modèle n'est pas trop biaisé par notre classe majoritaire et arrive bien à detecter et prédire des fin de phrases.

Nous avons notamment étudié le f1 score, moyenne harmonique de la précision et du rappel. Cette métrique est efficace face à des classes déséquilibrées comme cela a pu être le cas dans notre projet.

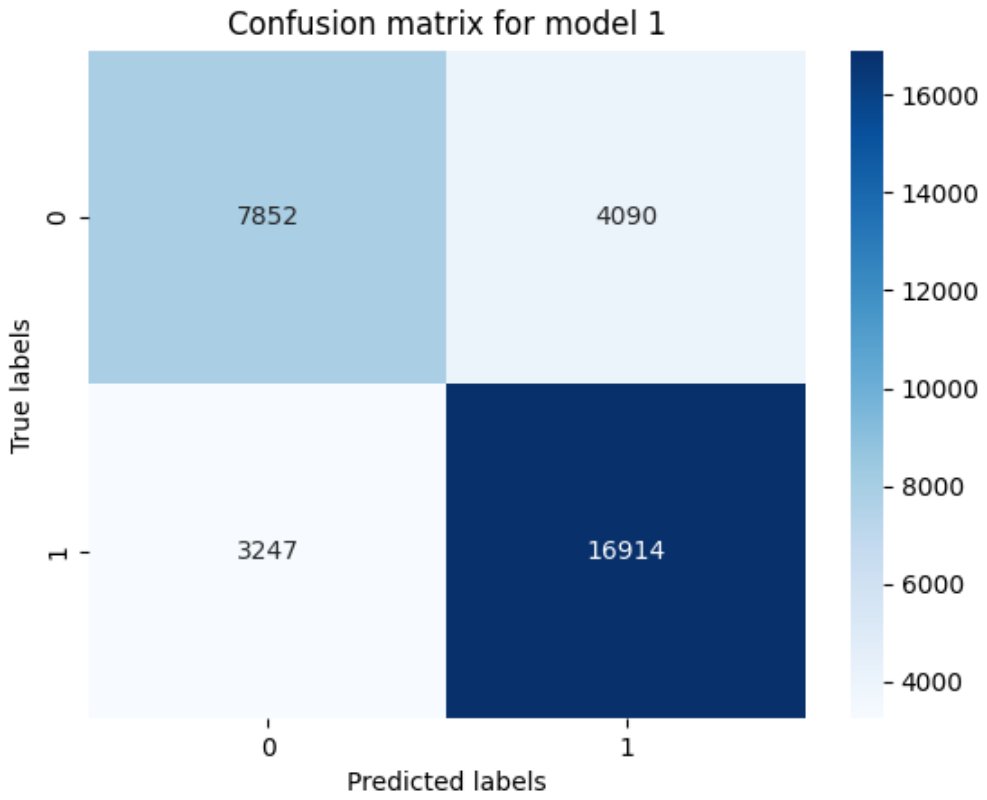
Pour visualiser de manière plus graphique les prédictions concrètes des différents modèles, nous avons opté pour la matrice de confusion. Pour simplifier l'affichage dans nos différents tests, nous avons créé une fonction intermédiaire (voir code **plot_confusion_matrix**). Cela nous a permis, à chaque affichage, d'ajouter un titre mais également de facilement sauvegarder en format png nos matrices.

Valeurs des résultats

Par soucis de clarté, nous ne présentons ici que les résultats clés et pertinents dans notre démarche et notre progression. D'aures résultats plus détaillés sont cependant disponibles dans le notebook.

- Premiers essais, avec un modèle MLP et un étiquetage 'any' (si une fin de note se trouve dans la sous-séquence, elle est classée 1) : score autour de 0.6 et 0.7 selon les hyperparamètres testés.
    - any-label - sans scaler :

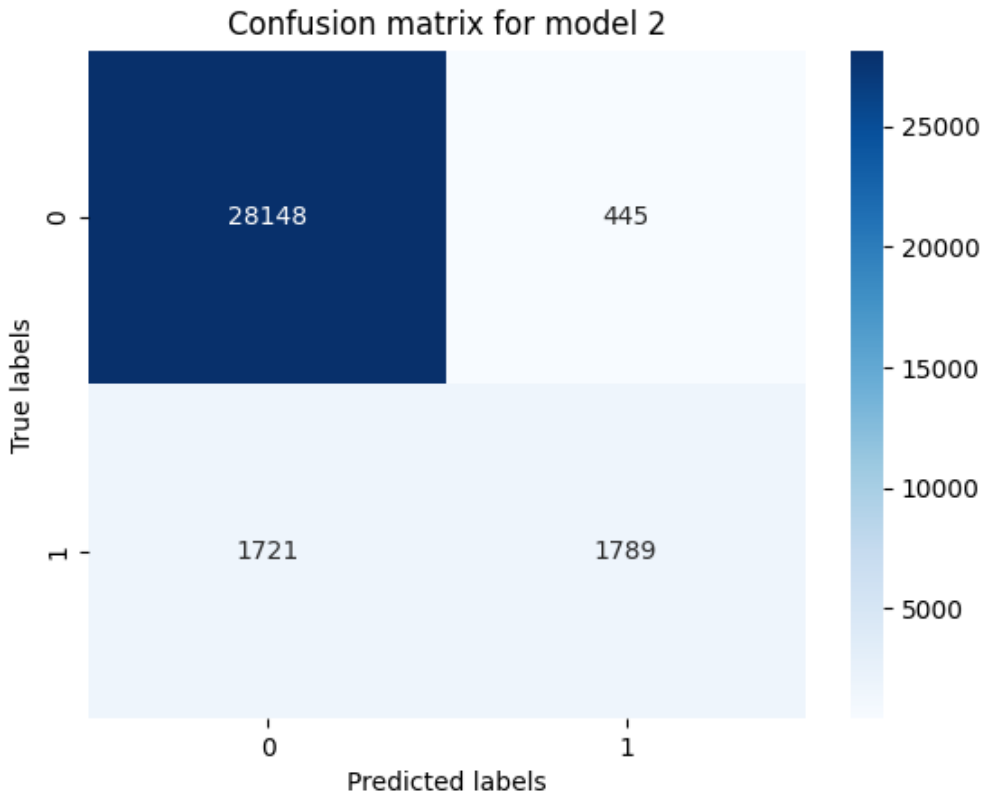
```
              precision    recall  f1-score   support

           0       0.63      0.55      0.58     11942
           1       0.75      0.81      0.78     20161

    accuracy                           0.71     32103
   macro avg       0.69      0.68      0.68     32103
weighted avg       0.70      0.71      0.71     32103

score : 0.7099648007974333
```



- On change donc l'étiquetage à 'end' (la sous-séquence est classée à 1 si la dernière note est une fin de phrase), toujours avec MLP.
    - sans scaler :

```
              precision    recall  f1-score   support

           0       0.63      0.55      0.58     11942
           1       0.75      0.81      0.78     20161

    accuracy                           0.71     32103
   macro avg       0.69      0.68      0.68     32103
weighted avg       0.70      0.71      0.71     32103

score : 0.7099648007974333
```

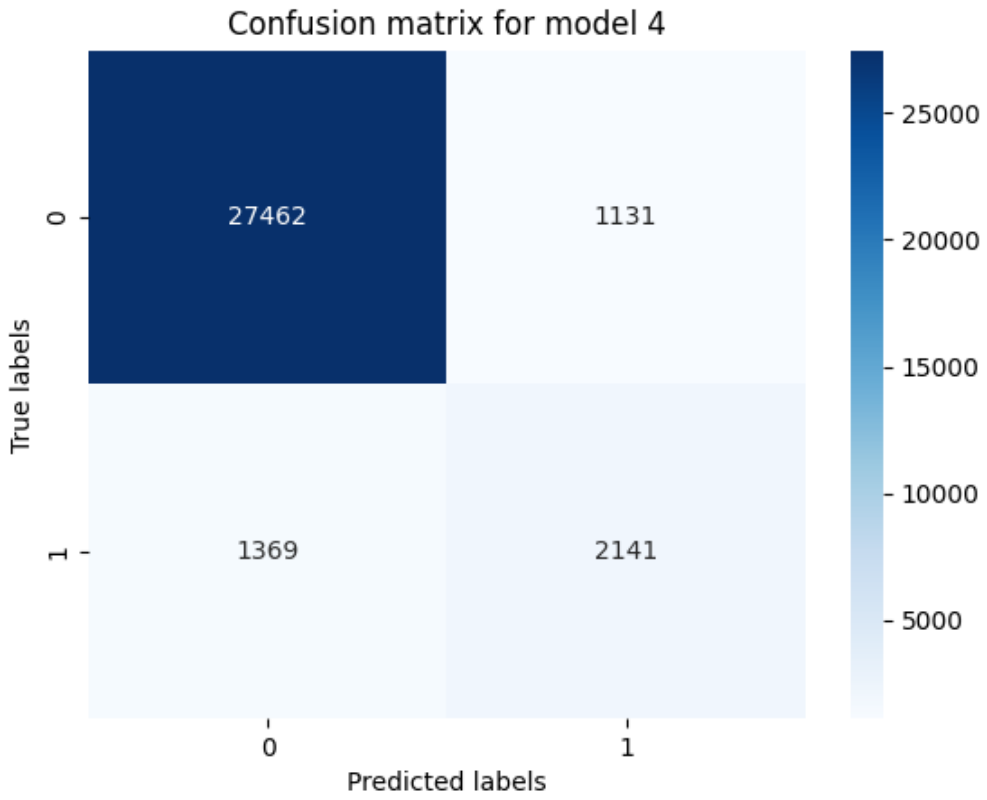
PROF



- avec scaler :

```
              precision    recall  f1-score   support

           0       0.95      0.98      0.96     28593
           1       0.76      0.59      0.67      3510

    accuracy                           0.94     32103
   macro avg       0.86      0.78      0.81     32103
weighted avg       0.93      0.94      0.93     32103

score : 0.9351150982774196
```

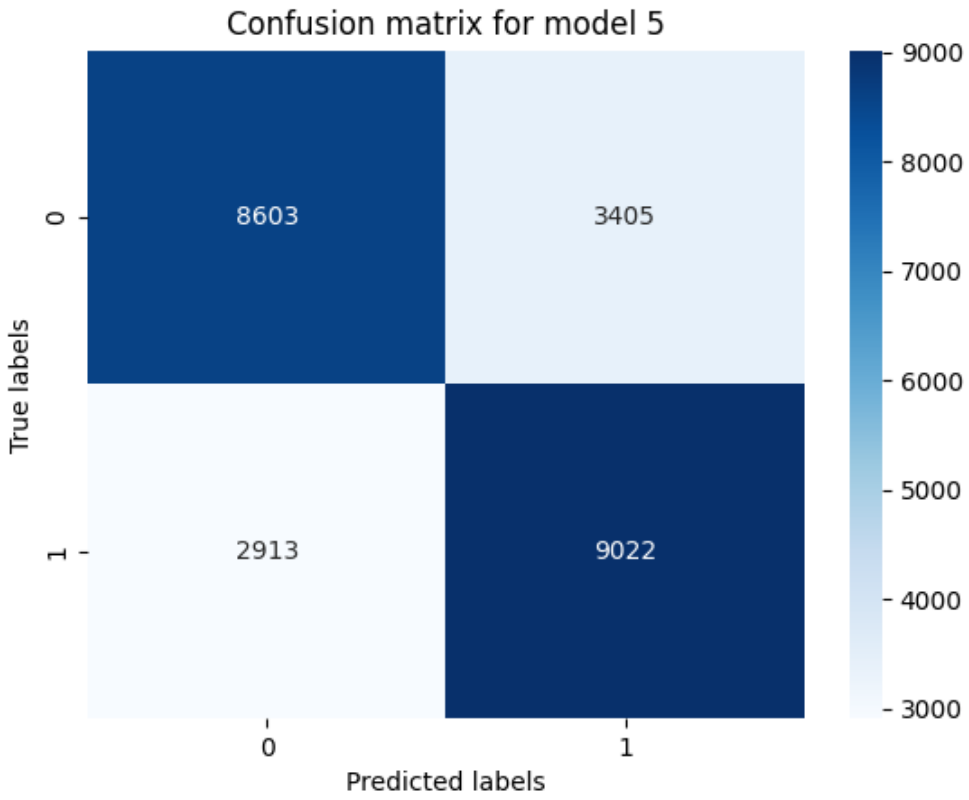
PROF



- Les résultats nous donnent l'impression que le modèle est biaisé par la classe majoritaire. On teste à nouveau le modèle MLP avec scaler mais avec l'équilibrage des classes. On peut noter que le nombre de nos données diminue.
    - any label :

```
                precision    recall  f1-score   support

           0       0.75      0.72      0.73     12008
           1       0.73      0.76      0.74     11935

    accuracy                           0.74     23943
   macro avg       0.74      0.74      0.74     23943
weighted avg       0.74      0.74      0.74     23943

score : 0.736123292820448
```

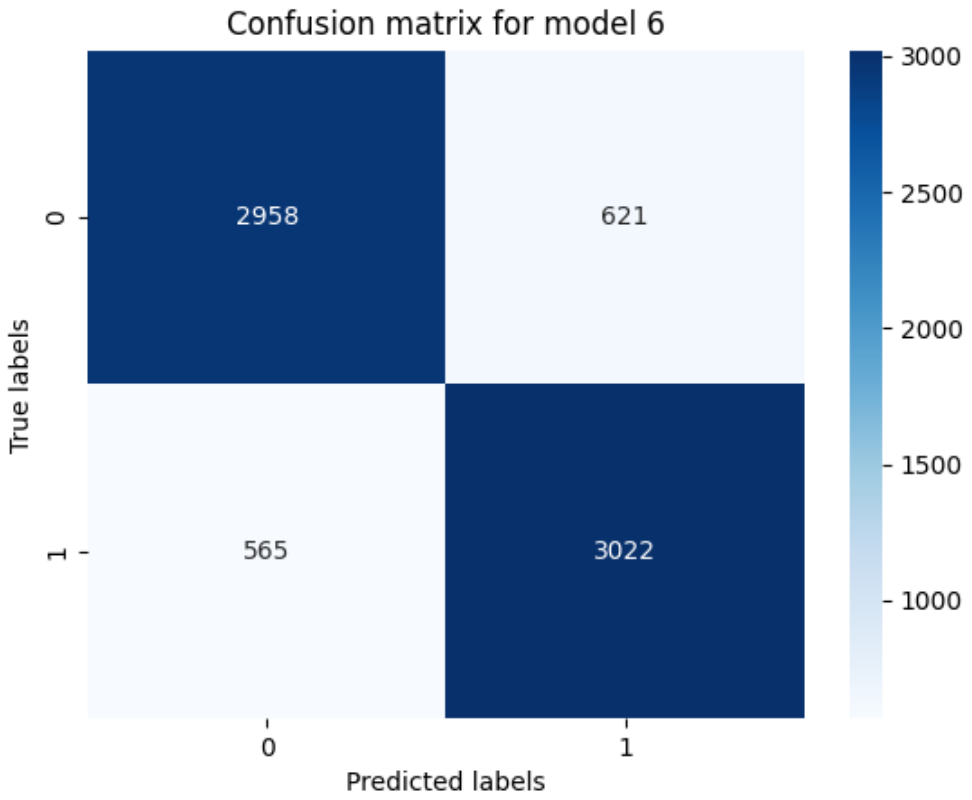
PROF



- end label :

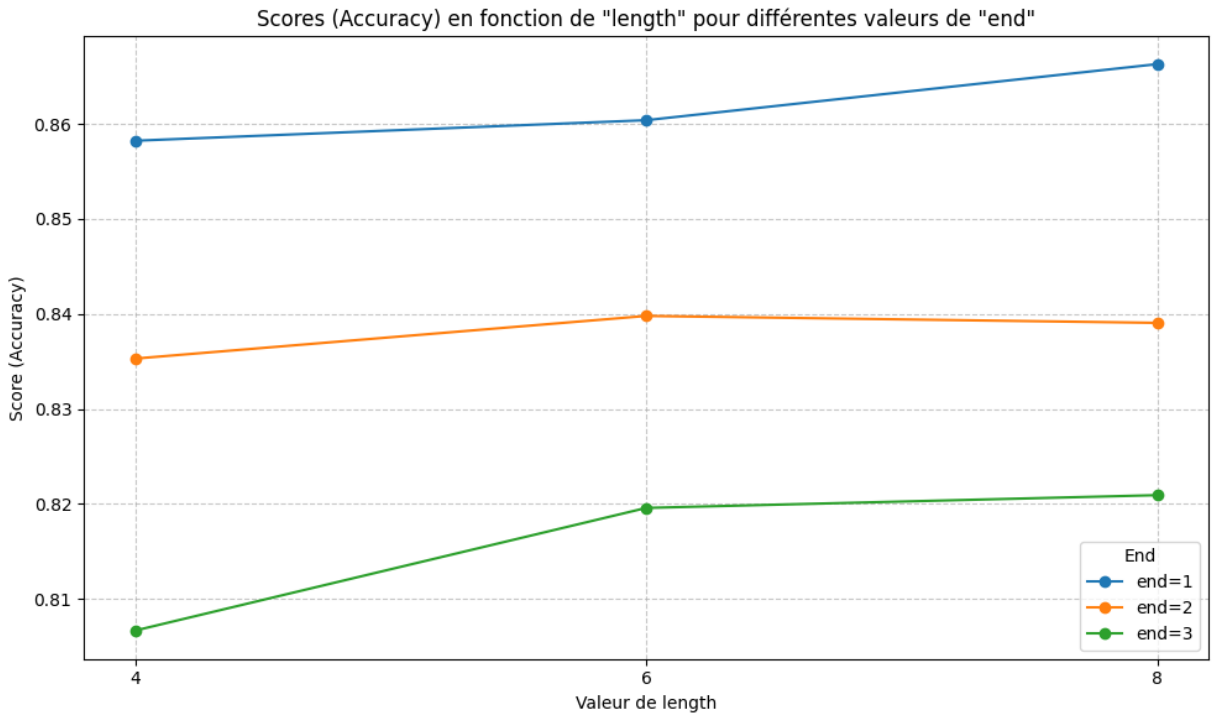
```
              precision    recall  f1-score   support

           0       0.84      0.83      0.83      3579
           1       0.83      0.84      0.84      3587

    accuracy                           0.83      7166
   macro avg       0.83      0.83      0.83      7166
weighted avg       0.83      0.83      0.83      7166

score : 0.8344962322076472
```

PROF

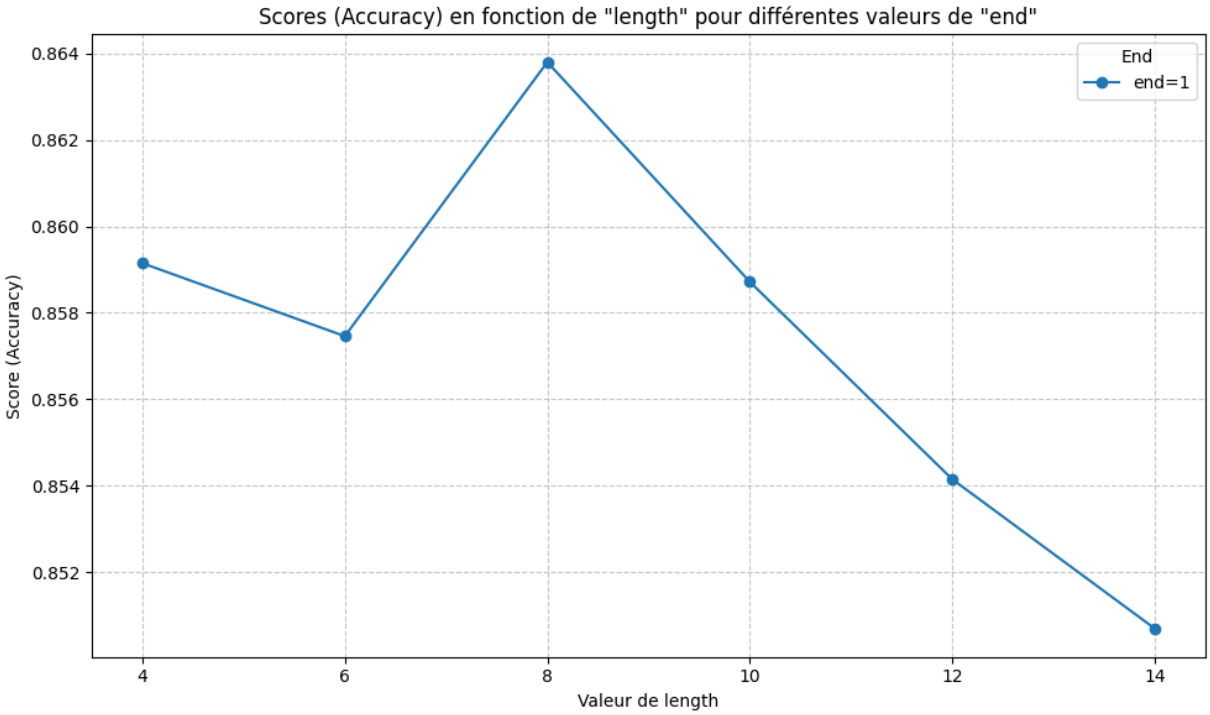


- On essaye à présent de croiser l'impact de longueurs de sous-séquences différentes avec des étiquetages différents, toujours un modèle MLP avec scaler et avec des classes équilibrées. (voici quelques résultats)

PROF



Ici nous pouvons voir que l'étiquetage 'end' ou 'hybrid' avec comme paramètre 1 donne le meilleur score

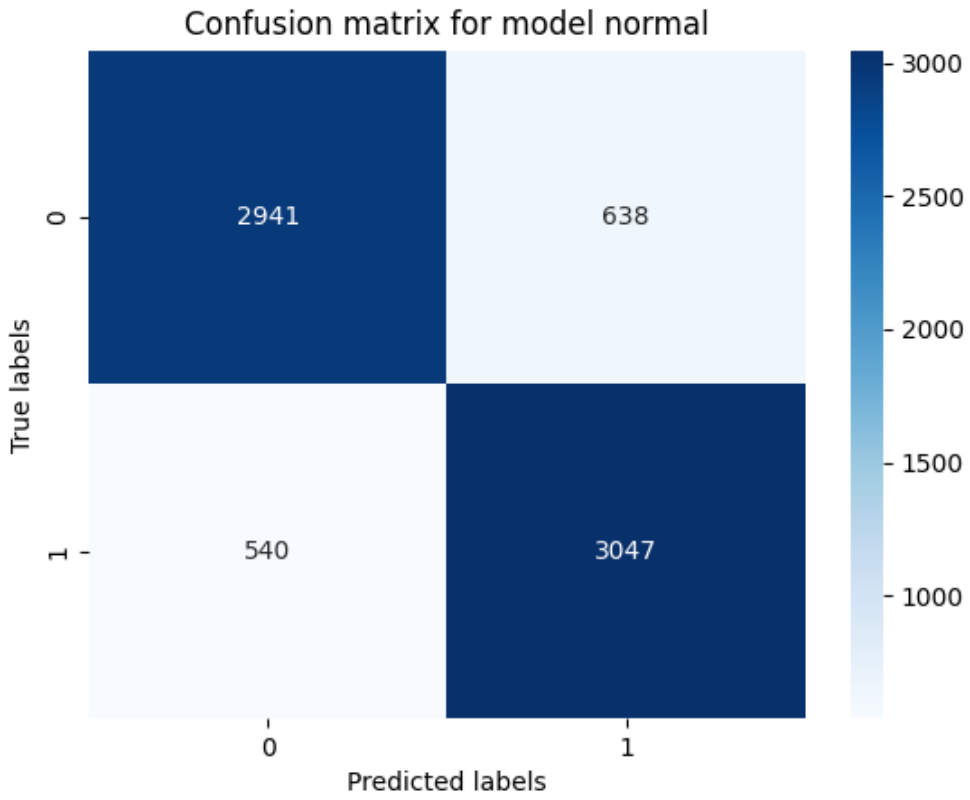


Ici nous voyons que le score monte pour atteindre un pic à 8 puis redescend ensuite. On donne plus de détails dans la suite dans la partie (interprétation)

- Comparaison entre un sous-séquençage classique et en commençant par les notes de fin (reversed) (voir explication au dessus). Nous avons gardé les meilleures paramètres des résultats déjà obtenus, donc toujours avec MLP, les classes sont éqilibrées, avec scaler, un étiquetage end label et une longueur de séquence 8 :
  - classique :

```
              precision    recall  f1-score   support

           0       0.84      0.82      0.83      3579
           1       0.83      0.85      0.84      3587

    accuracy                           0.84      7166
   macro avg       0.84      0.84      0.84      7166
weighted avg       0.84      0.84      0.84      7166

score : 0.8372871895060006
```

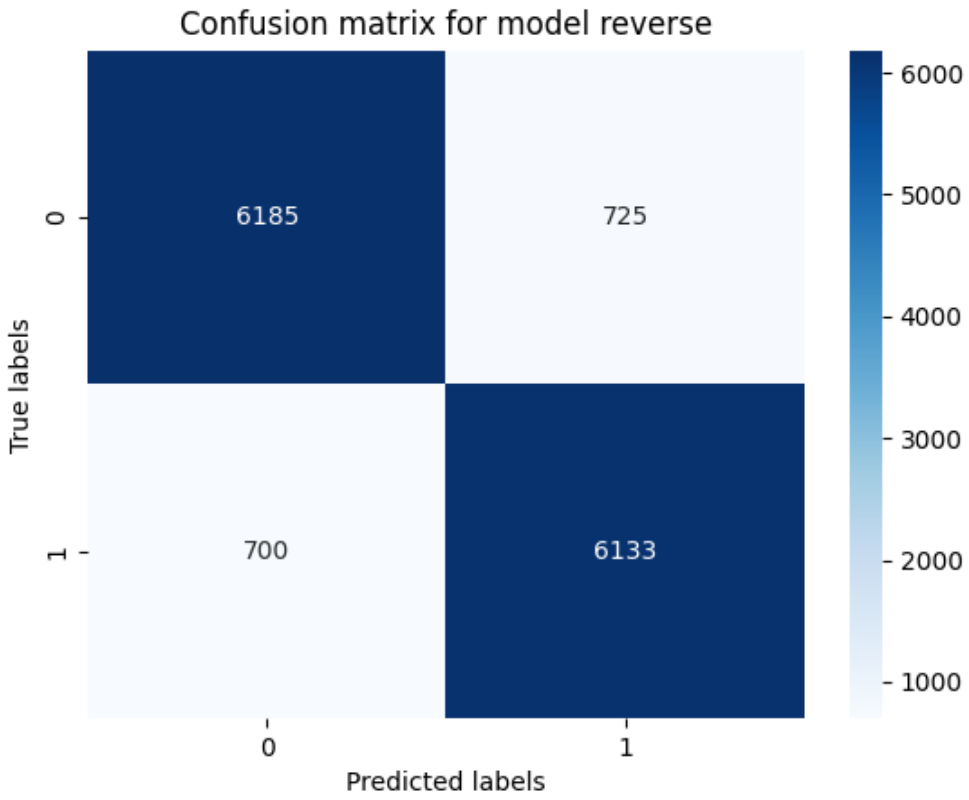
PROF



- reverse :

```
              precision    recall  f1-score   support

           0       0.88      0.87      0.87      6910
           1       0.87      0.88      0.87      6833

    accuracy                           0.87     13743
   macro avg       0.87      0.87      0.87     13743
weighted avg       0.87      0.87      0.87     13743

score : 0.8745543185621771
```

PROF



- Comparaison maintenant entre différents modèles. Avec les mêmes paramètres que juste au dessus, en ajoutant le sous-séquençage "reversed". Les Mean cross-validation scores sont :
    - Logistic Regression: 0.9255508778938977
    - SVM: 0.9310906794335526
    - Random Forest: 0.9344145044735676
    - KNN: 0.7979809001560058
    - MLP: 0.9280437632496057
    - Naive Bayes: 0.7999196789088607
    - Decision Tree: 0.8895727689905273
    - SGD: 0.9131173861016381
- Random Forest obtient le meilleur score, on regarde ses résultats après une grid search:
    - Best parameters found by grid search: {'model__max_depth': None, 'model__min_samples_leaf': 1, 'model__min_samples_split': 2, 'model__n_estimators': 300}
    - Mean cross-validation score with best model: 0.9345992098317941

PROF

Interprétation

En regardant ces résultats, on peut apprendre plusieurs choses, que ce soit sur les modèles ou les choix de traitement des données.

- Impact de l'étiquetage ('any' ou 'end') :

On remarque que le chois de l'étiquetage a un impact très fort sur les résultats. Les résultats sont systématiquements inférieurs lorsque l'on applique l'étiquetage 'any'. Cela a tendance a brouiller l'apprentissage du modèle. (voir explication au dessus)

- Longueur des sous-séquences :

En général, si nous disposons de suffisament de sous-séquences, alors il vaut mieux envisager donner de plus longues sous-séquences au modèle pour lui donner plus de contexte, en terme de notes, pour l'apprentissage.En effet, plus les sous-séquences sont longues, moins nous en avons. Il faut donc trouver le juste milieu entre grand nombre de sous-séquences ou grande longueur de sous-séquence. Il faut cependant ne pas prendre de trop longues sous-séquence car elles pourraient contenir des phrases entières ce qui n'a pas vraiment de sens ici car l'on souhaite des morceaux de phrases. Ici, selon les graphiques, 8 semble idéal.

- Importance du prétraitement des données (Scaler)

Les résultats montrent une amélioration des performances lorsque les données sont normalisées avec un scaler. Par exemple, avec l'étiquetage 'end', le score passe de 0.91 (sans scaler) à 0.93 (avec scaler). Cela nous confirme que les méthodes de normalisation sont essentielles pour les modèles sensibles à l'échelle des données, comme avec celui choisi au départ, MLP.

- Classe majoritaire et impact de l'équilibrage

Les performances initiales observées sur certaines configurations (par exemple, 0.91-0.93) étaient particulièrement élevées et semblaient victimes du biais de la classe majoritaire (visible également dans les matrices de confusion). En équilibrant les classes, les scores sont devenus plus représentatifs de la vraie capacitée des modèles, notamment pour les configurations avec 'any' (0.75) et 'end' (0.87).

- Squençage classique ou inversé (reversed)

PROF

Le séquençage inversé des données, en commençant par récupérer des sous-séquences de notes à la fin de la séquence, a permis d'observer une nette différence dans les scores (séquençage inversé : 0.90, séquençage classique : 0.87). Cela s'explique, comme on peut le voir dans les matrices de confusion, par l'augmentation notable du nombre de sous-séquences classées à 1. Après équilibrage on peut donc garder plus d'exemples des deux classes.

- Comparaison des modèles

Nos comparaisons entre les modèles montrent que la Random Forest obtient les meilleurs résultats avec un score moyen en cross-validation de 0.9344, suivi de près par les SVM (0.9311) et les MLP (0.9280). Les algorithmes restants tels que le KNN (0.7979) et le Naive Bayes (0.7999) sont, dans notre contexte, beaucoup moins performants. Ce sont des modèles qui capturent moins facilement des relations complexes entre les données.

- Importance des hyperparamètres

Avec des paramètres optimaux, le meilleur modèle (random Forest) a atteint une performance en validation croisée de 0.9346, ce qui augmente très légèrement notre résultat précédant. L'ajustement des hyperparamètres permet ainsi d'optimiser un maximum les performances d'un modèle. Cette étape n'est pas à négliger car comme vu dans les résultats, cela peut même changer l'odre de performance des modèles.

Notre meilleur score global a ainsi été obtenu grâce au modèle Random Forest, avec une étiquetage 'end', une longueur de sous-séquence de 8 notes fixes, un sous-séquençage en partant de la fin, un équilibrage des 2 classes, l'application d'un scaler et enfin l'ajustement des hyper-paramètres grâce à notre gridSearch. Pour un score de 0.9346.

Conclusion

Ce projet a été particulièrement enrichissant en terme de méthodologie. En effet, nous avons du prendre en main une problématique et un jeu de données de zéro, sans indication de solutions ni aide sur l'approche à adopter comme on peut souvent le retrouver en tp.
Cela signifiait donc prendre en main et gérer tous les aspects de l'apprentissage d'un modèle. Du choix du traitement des données jusqu'aux choix techniques sur les hyperparamètres des modèles.

Ce qui a pu être difficile au début, a été de toujours revenir en arrière dans nos démarches pour reprendre certaines étapes et tester de nouvelles possibilités.
En effet, au fur et à mesure du projet, il a fallu remettre en question nos propres choix pour ne pas rester enfermés dans une solution et cloisonnés à un score.

A cela s'ajoute donc l'importance d'être organisé dans notre notebook pour ne pas se perdre parmi tous les différents tests.

Le projet a également été enrichissant d'un point de vue technique car nous avons découvert de nouvelles bibliothèques et fonctions très pratiques, par exemple classification_report.

De plus, nous avons noté l'importance de rester critique face aux scores et leurs significations. Un bon score ne garantie pas systématiquement un bon modèle. Il est important de croiser les résultats et les métriques de scores pour réellement comprendre ce qui se cache derrière les chiffres et ce que cela signifie concrétement en terme de prédiction pour le modèle.

PROF

Nous avons également appris qu'il faut choisir intelligement la façon de trouver les meilleurs paramètres et de ne pas simplement tester toutes les possibilités car cela serait trop coûteux en temps.

Une dernière amélioration que nous aurions pu faire est que nous aurions pu tester encore plus d'attributs. Cela aurait pu permettre d'avoir plus de données en entré de nos modèles, c'est une piste d'amélioration. D'autres modèles de gestion de series temporelles sont également une piste d'amélioration.

De manière plus globale, il serait intéressant de tester ce projet sur des jeux de données encore plus enrichis, avec des styles de musiques plus diversifiés et récents, de la pop actuelle par exemple.