Feriel ABDI
Manel BENKORTEBI
Cassandra GBABOUA
Ines LEBIB

Compte Rendu - AIAO

Child Mind Institute - Detect Sleep States

Introduction

Le sommeil, composante essentielle de notre vie quotidienne, influence profondément notre bien-être, notre développement et nos capacités cognitives. Pourtant, malgré son importance, les études sur le sommeil se sont souvent confrontées à des obstacles, notamment en raison de l'absence de données naturelles fiables et d'annotations précises. Les méthodes actuelles ne parvenant pas toujours à cerner les subtilités du cycle sommeil-éveil, nous demeurons en quête d'une meilleure compréhension de ses mécanismes et des conséquences sur la santé.

Dans ce contexte, le projet lancé par le Global Center for Child & Adolescent Mental Health au Child Mind Institute, vise à exploiter les données d'un accéléromètre, porté au poignet pour détecter avec précision les phases d'endormissement et de réveil chez les enfants et adolescents.

L'objectif principal de ce projet est ainsi de mettre en place un système innovant capable de déterminer l'état de sommeil chez les jeunes en se basant sur ces données.

Ce projet présente donc une opportunité unique de relever ce défi en exploitant les capacités de l'apprentissage automatique et de l'analyse des données. En concevant un modèle de détection précis pour l'endormissement et le réveil grâce aux données d'accéléromètre, nous ouvrons la voie pour des recherches enrichies sur le sommeil.
Une telle avancée permettrait aux chercheurs de conduire des études sur le sommeil plus larges et fiables, notamment chez les populations de jeunes, dont le sommeil influence de manière significative l'humeur, les émotions, et le comportement.

L'ensemble des données de ce projet, apportées par le Healthy Brain Network, comprend environ 500 enregistrements. L'accéléromètre porté aux poignées des enfants permet d'annoter deux types d'évènements : 'wakeup' (réveil) et 'onset' (endormissement).

Le but de ce projet est ainsi de détecter ces deux événements au sein de ces séries temporelles d'accélérations.

Matériels et méthodes

Matériels

Datasets

Nous avons à notre disposition 3 fichiers : train_series.parquet, train_events.csv, test_series.parquet.

Le dataset train_series est à utiliser comme données d'entraînement. Chaque série est un enregistrement continue des données de l'accéléromètre par sujet chaque jour.

Series_id : identifiant unique pour chaque série d'accéléromètres

Step : Un pas de temps entier pour chaque observation au sein d'une série

Timestamp : Une datetime correspondant au format ISO 8601 : %Y-%m-%dT%H:%M:%S%z

Anglez : L'angle z fait référence à l'angle du bras par rapport à l'axe vertical du corps.

Enmo : est la norme euclidienne moins un de tous les signaux d'accéléromètre, avec des valeurs négatives arrondies à zéro.

Le dataset train_events nous renseigne sur les apparitions des évènements 'onset' et 'wakeup' :

Series_id : identifiant unique pour chaque série d'accéléromètres

Night : Une paire d'événements au maximum peut se produire pour chaque nuit

Event : Le type d'événement : 'onset' ou 'wakeup'

Step : L'heure d'occurrence enregistrée de l'événement dans la série d'accéléromètres.

Le dataset test_series est le jeu de données test contenant les mêmes champs que 'train_series'.

L'ensemble du projet a été codé en Python et les scripts de deep learning ont été lancés via un cluster ssh offrant une capacité de calcul supplémentaire pour les tâches intensives en ressources.

Logiciels et packages

Pour assurer une manipulation efficace de nos données, nous avons utilisé plusieurs bibliothèques. Par exemple, **numpy** pour les opérations numériques, pandas pour la gestion des données structurées, et **fastparquet** pour gérer le format Parquet de nos données brutes. La visualisation a été réalisée grâce à **matplotlib** et **seaborn**, offrant une représentation

flexible de nos données. Notons que ces outils sont une sélection parmi d'autres que nous avons utilisée, sans citer l'ensemble de nos ressources.

Quant à notre analyse principale, son cœur repose sur **Keras** pour la mise en œuvre de techniques de deep learning. Keras est une bibliothèque open-source qui facilite la création de modèles de Deep Leanrning et offre une interface afin de créer des modèles d'apprentissage automatique. Nous avons, par exemple, employé des couches comme Dense et LSTM pour la construction de nos modèles. L'optimisation a été réalisée via tensorflow et d'autres fonctions de Keras. **Tensorflow** est aussi une bibliothèque open-source d'apprentissage automatique afin de faciliter la création et l'apprentissage de réseaux de neurones. Cette bibliothèque permet d'effectuer des calculs numériques de manière efficace. En parallèle, **scikit-learn** nous a assistés dans la préparation des données, notamment pour la normalisation, la conversion des labels et la séparation des datasets en données d'entraînement et de tests.

Méthodes

Prétraitement

La première étape, dite de Preprocessing, consiste à nettoyer et organiser nos données.

Dans notre cas nous avions les données de train_events avec 14 508 lignes représentant soit un événement d'endormissement (onset), soit un événement de réveil (wakeup) associés à un identifiant de série (series_id), reflétant une nuit spécifique, et accompagnés d'un horodatage (timestamp). Le train_series contenait 127 946 340 lignes renseignant sur des informations continues de sommeil telles que enmo et anglez. Ainsi, nous avons associé de manière chronologique et précise, chaque intervalle de temps des données continues du fichier train_series avec les événements d'endormissement et de réveil du fichier train_events.

Suite à la fusion des deux datasets, nous avons constaté que certaines périodes temporelles n'avaient pas d'événement associé. Pour pallier cette absence, et garantir la continuité de notre jeu de données, nous avons attribué la catégorie wakeup ou onset aux intervalles manquants. Cette décision était basée sur la logique que si un individu ne s'endort pas ou ne se réveille pas explicitement à un moment donné, il est plus probable qu'il soit simplement éveillé.

Les données initiales présentaient également des horodatages sous le format "2018-08-14T22:26:00-0400". Ce format inclut non seulement la date et l'heure, mais également un décalage horaire indiqué par "-0400". Pour simplifier la manipulation des données, nous avons transformé cet horodatage en "2017-08-15 16:45:00". Ce nouveau format, plus lisible, sépare clairement la date de l'heure à l'aide d'un espace, sans inclure le décalage horaire.

Enfin, face à la problématique des données manquantes, nous avons adopté une approche stricte : pour un individu présentant des valeurs manquantes, nous avons choisi de supprimer la totalité de ses données. Cette décision vise à garantir l'intégrité de notre analyse, évitant ainsi d'introduire des biais ou des fausses interprétations. Ainsi, nous obtenons un dataset qui ne contient plus que 14334881 lignes.

Une étape cruciale de prétraitement est la normalisation de nos données, impliquant de les mettre à une échelle commune entre 0 et 1, afin de garantir de ne pas biaiser les résultats d'analyses. La normalisation des données est effectuée sur l'ensemble des données d'entraînement et de validation, puis appliquée de la même manière aux données de test pour garantir la cohérence des échelles entre les ensembles de données.

Analyses préliminaires

Nos variables explicatives X sont 'enmo' et 'anglez', tandis que notre variable à prédire Y est 'event'.

Pour notre étude, nous avons commencé par réaliser une Analyse en Composantes Principales (ACP) afin de visualiser nos données bidimensionnelles. L'ACP est une technique statistique qui permet de transformer des données originales, souvent corrélées, en un ensemble de valeurs de variables linéairement non corrélées appelées composantes principales. Dans notre cas, nous avons seulement deux composantes principales à considérer, ce qui simplifie la visualisation, mais limite aussi notre capacité à interpréter des structures plus complexes au sein des données.

Étant donné que 'event' est une variable qualitative discrète, nous avons choisi de faire une régression logistique. La régression logistique est idéalement adaptée aux variables cibles binaires, ce qui nous permet d'avoir une vision initiale claire de la structure et des tendances des données. C'est une méthode qui permet d'obtenir rapidement des indications préliminaires. Ainsi, elle constitue un point de référence essentiel, posant les bases pour de futures modélisations plus élaborées et l'exploration de modèles plus sophistiqués.

Deep learning

Réseaux de Neurones Récurrents (RNN) :

Le premier algorithme de Deep Learning que nous avons implémenté est un modèle de réseaux de neurones récurrents (RNN). Les RNN sont adaptés pour traiter des données séquentielles telles que des séries temporelles. Ils sont capables de capturer les dépendances présentes dans ces données en utilisant des connexions récurrentes. Ces connexions permettent au modèle de prendre en compte les informations des données précédentes lors du traitement des données, ce qui lui permet d'apprendre à partir d'informations passées.

La première étape a consisté à appliquer un processus de "padding" à nos données, dans le but d'assurer l'uniformité et d'obtenir une dimension appropriée pour notre modèle. Initialement, nous avons constaté une disparité dans le nombre de points de données parmi les 37 individus, une fois le prétraitement effectué. Afin de remédier à cette incohérence, nous avons identifié l'individu possédant le plus grand nombre de lignes de données, puis nous avons ajouté des valeurs "-1" aux autres individus pour les aligner sur cette référence. De plus, pour réduire la dimension de nos données tout en conservant leur pertinence, nous avons échantillonné les données à un intervalle de 50, ce qui nous a permis d'obtenir une observation toutes les 4 minutes au lieu de 5 secondes.

le modèle est un réseau de neurones séquentiel comprenant des couches de masquage, SimpleRNN, dropout et une couche dense, utilisant l'optimiseur Adam avec un taux d'apprentissage de 0.0001, et est compilé pour effectuer une classification binaire avec une métrique d'accuracy pour évaluer la précision des prédictions.

Son architecture se présente ainsi :

- Création du modèle séquentiel : Le modèle est construit de manière séquentielle en ajoutant les couches de neurones les unes après les autres.
- Couche de masquage (Masking) : La première couche ajoutée est la couche de masquage, utilisée pour ignorer les valeurs ayant une valeur de -1 lors de l'apprentissage du modèle.
- Couche SimpleRNN : Une couche SimpleRNN avec 64 neurones est ajoutée. Elle prend en entrée la longueur de l'ensemble d'entraînement (1260) et le nombre de features (2). Le paramètre "return_sequences = True" permet à la couche de renvoyer une séquence de sortie, préservant ainsi l'information temporelle.
- Couche de dropout : Une couche de dropout avec un taux de 0.2 est ajoutée pour prévenir le surapprentissage en désactivant aléatoirement 20% des neurones de la couche précédente pendant l'entraînement.
- Autre couche SimpleRNN : Une autre couche SimpleRNN est ajoutée, cette fois avec 128 neurones et également configurée pour renvoyer une séquence de sortie.
- Deuxième couche de dropout : Une autre couche de dropout avec un taux de 0.2 est ajoutée après la deuxième couche SimpleRNN.
- Troisième couche SimpleRNN : Une troisième couche SimpleRNN avec 64 neurones est ajoutée, renvoyant une séquence de sortie.
- Troisième couche de dropout : Une troisième couche de dropout avec un taux de 0.2 est ajoutée après la troisième couche SimpleRNN.
- Couche dense : Enfin, une couche dense entièrement connectée est ajoutée avec une seule sortie et une fonction d'activation sigmoïde pour obtenir une sortie binaire entre 0 et 1.
- Optimiseur : L'étape suivante consiste à créer un optimiseur, en l'occurrence l'optimiseur Adam. Adam combine la descente de gradient stochastique (SGD) et la descente de gradient adaptatif (Adagrad) pour ajuster les poids et les biais du modèle. Un taux d'apprentissage (learning_rate) de 0.0001 est choisi pour contrôler la vitesse d'apprentissage du modèle.

- Compilation du modèle : Enfin, le modèle est compilé à l'aide de la méthode "compile". On spécifie l'optimiseur, la fonction de perte (dans ce cas, "binary_crossentropy" pour une classification binaire) et les métriques à utiliser pour évaluer les performances du modèle pendant l'entraînement (dans ce cas, "accuracy" pour mesurer la précision du modèle).

La méthode "fit" est utilisée pour entraîner un modèle d'apprentissage automatique en ajustant ses paramètres internes à partir des données d'entraînement, avec pour objectif de minimiser la fonction de perte. Les données d'entraînement (X) et les étiquettes correspondantes (Y) sont fournies en entrée. On peut également spécifier un paramètre "validation_split" pour réserver une partie des données d'entraînement pour la validation du modèle, généralement fixé à 20%. Le paramètre "epochs" indique le nombre d'itérations complètes sur l'ensemble des données d'entraînement, et il est important d'en choisir un nombre suffisamment grand pour que les courbes d'accuracy convergent. Enfin, le paramètre "batch_size" représente le nombre d'échantillons d'entraînement utilisés dans une itération, et le modèle est mis à jour après chaque lot. Un exemple donne "epochs" à 40 et "batch_size" à 256.

La méthode "model.evaluate()" est ensuite utilisée pour évaluer la performance d'un modèle d'apprentissage automatique sur un ensemble de données de test en calculant des métriques de performance telles que la perte (loss) et l'accuracy. Elle compare les prédictions du modèle avec les valeurs de test pour évaluer sa performance.

Enfin, la méthode "model.predict()" est utilisée pour obtenir les prédictions d'un modèle d'apprentissage automatique sur des données qui n'ont pas été utilisées lors de l'entraînement du modèle. Cette méthode renvoie les prédictions du modèle sur les données de test ou de validation, permettant ainsi d'évaluer les performances du modèle sur des données indépendantes de l'ensemble d'entraînement.

Long Short-Term Memory (LSTM)

Le modèle LSTM (Long Short-Term Memory) est un type de réseau de neurones récurrents (RNN) qui est spécialement adapté pour traiter des séquences de données et capturer les dépendances à long terme. Contrairement aux RNN traditionnels, les LSTM sont capables de mémoriser des informations sur une longue période.
Les LSTM sont conçus pour résoudre les problèmes de vanishing gradient présents dans les RNN traditionnels.

Afin de pouvoir appliquer le modèle LSTM sur nos données, nos données doivent avoir trois dimensions alors qu'actuellement, elles en ont seulement deux. Pour pallier cela, nous avons créé des fenêtres (windows) avec un pas de 50. Cela a plusieurs avantages, premièrement d'ajouter la dimension manquante mais également de diminuer le nombre de données avec lesquelles nous allons faire l'entraînement. Nous n'aurons plus des données toutes les 5 secondes mais toutes les 4 minutes environ ce qui peut nous aider à éviter le surapprentissage.

A présent, la taille de notre X est (286698, 1, 2) : 286698 lignes, 1 la taille de la fenêtre et 2 nos colonnes ‘enmo’ et ‘anglez’.

Nous construisons un modèle séquentiel en ajoutant une première couche de réseau de neurones LSTM avec 74 neurones, une fonction d'activation ReLU, et une sortie séquentielle. Ensuite, nous appliquons un dropout de 20% pour éviter le surapprentissage. Une deuxième couche LSTM et un autre dropout sont ajoutés. Enfin, une couche dense avec un seul neurone, utilisant une fonction d'activation sigmoïde, permet une classification binaire entre 0 et 1.

L'optimiseur utilisé est Adam avec un taux d'apprentissage de 0.0000001, et le modèle est compilé avec les mêmes paramètres que précédemment, notamment la fonction de perte binaire ('binary_crossentropy') et la métrique d'accuracy. Le modèle compilé est renvoyé à la fin de la fonction.

La méthode fit prend en entrée les ensembles de données d’entraînement (X) et les étiquettes correspondantes (Y), la validation_split est fixée à 20, l’epochs à 20 et enfin le batch_size à 128.

Nous avons ensuite évalué et prédit nos modèles de la même manière que la méthode RNN avec model.evaluate() et model.predict() .

Résultats

Analyse en composante principale

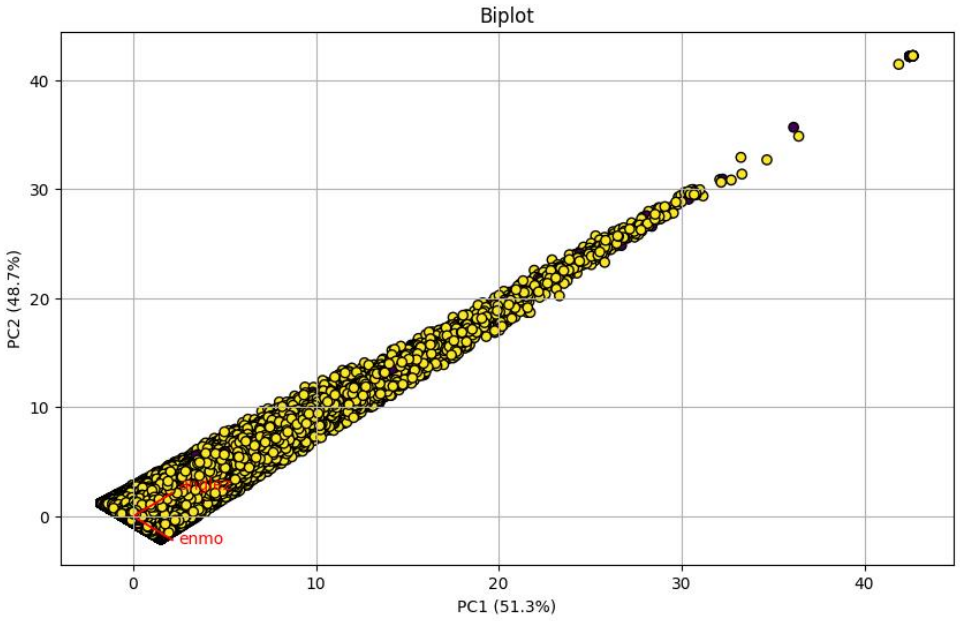

Figure 1: Graphe d'Analyse en Composante Principale

Ici nous pouvons voir que les événements wakeup (représentés en jaune) et onset (en violet) sont visualisés en fonction des composantes principales 1 et 2. La première composante principale (PC1) explique 51,3% de la variance, tandis que la deuxième composante (PC2) en explique 48,7%. Les données montrent une trajectoire diagonale marquée, dominée en grande partie par les événements wakeup. Ces événements jaunes semblent se superposer et

prendre le dessus, particulièrement dans la zone où la variable enmo est mise en évidence. Les événements onset, bien que présents, sont nettement moins dominants et semblent être plus dispersés, particulièrement vers les zones périphériques du graphique.

Régression logistique

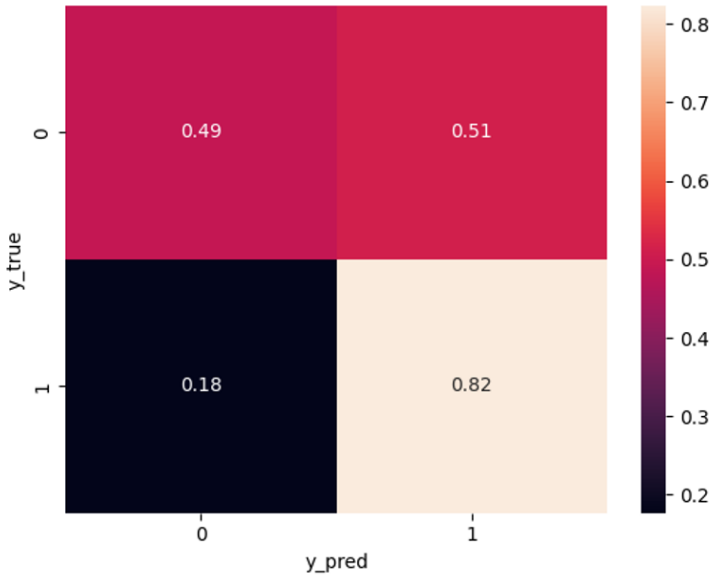

Figure 2: Matrice de confusion des y_true et des y_test obtenus après entrainement et test de validation

Les colonnes représentent les prédictions (y_pred) avec les classes 0 (onset) et 1 (wakeup).

Les lignes représentent les vraies valeurs (y_true), également avec les classes 0 et 1.

Nous avons :

0,49 : Proportion des vraies instances de classe 0 qui ont été correctement prédites comme 0 (Vrai Négatif).

0,51 : Proportion des vraies instances de classe 0 qui ont été incorrectement prédites comme 1 (Faux Positif).

0,18 : Proportion des vraies instances de classe 1 qui ont été incorrectement prédites comme 0 (Faux Négatif).

0,82 : Proportion des vraies instances de classe 1 qui ont été correctement prédites comme 1 (Vrai Positif).

Les valeurs dans la diagonale (0,49 et 0,82) indiquent les prédictions correctes tandis que les valeurs hors de la diagonale (0,51 et 0,18) montrent les erreurs de prédictions.

Dans l'ensemble, le modèle semble assez performant pour prédire la classe 1 avec un taux de succès de 82%. Cependant, il a plus de difficulté à distinguer les deux classes lorsqu'il s'agit de

la classe 0 avec un taux de succès de seulement 49%. Cela peut signifier que le modèle a tendance à prédire davantage la classe 1 par rapport à la classe 0 dans certains cas.

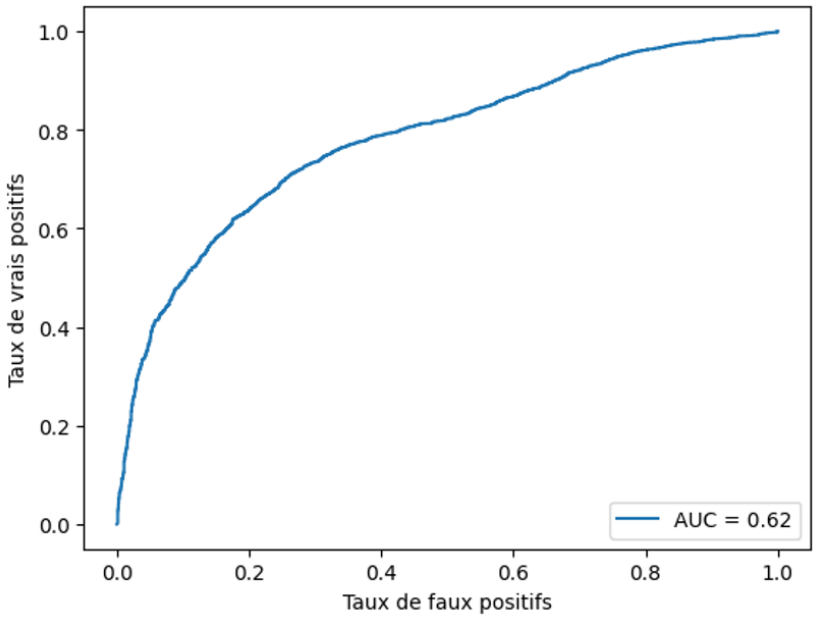

Figure 3 : Courbe ROC de régression logistique

La courbe ROC présentée illustre la capacité du modèle à distinguer les classes ‘onset’ et ‘wakeup’.

Dans ce graphique, l'axe des ordonnées représente le "Taux de vrais positifs" (aussi appelé sensibilité), qui est la proportion de véritables positifs correctement identifiés par le modèle. L'axe des abscisses représente le "Taux de faux positifs" (1-spécificité), qui est la proportion de négatifs qui sont incorrectement identifiés comme positifs.

Nous pouvons observer que la courbe ROC commence au point d'origine (0,0) et se termine au point (1,1). Idéalement, une courbe ROC se situerait le plus près possible du coin supérieur gauche (point (0,1)), ce qui indiquerait une performance parfaite.

L'aire sous la courbe de notre modèle (AUC) est de 0.62. L'AUC est une mesure qui résume la performance globale du modèle : une AUC de 0.5 indique une performance équivalente à celle du hasard, tandis qu'une AUC de 1.0 signifie une performance parfaite. Une AUC de 0.62 indique que le modèle a une capacité modérée à distinguer entre nos deux classes ‘onset’ et ‘wakeup’.

La courbe s'écarte progressivement de la diagonale, ce qui indique que le modèle a une certaine capacité à discriminer nos classes, mais elle n'atteint pas le coin supérieur gauche.

En résumé, le modèle a montré une capacité modérée à classer correctement les échantillons ‘onset’ et ‘wakeup’.

Deep Learning

Pour nos deux modèles RNN et LSTM, nous avons d'abord entraîné le modèle sur 80% de nos données (train), puis évalué ses performances sur les 20% restants (test) avec des taux d'accuracy (précision) et de loss (perte). Ces indicateurs reflètent la capacité du modèle à classer correctement les données de test et à minimiser l'erreur lors de cette classification : Les graphes d'accuracy montrent la précision du modèle au fil des epochs. Il permet de visualiser l'évolution de notre modèle.

Les graphes de loss (perte) nous montre comment la fonction de perte du modèle évolue au fil des epochs. La fonction de perte mesure les différences entre les prédictions du modèle et les valeurs réelles.

Réseaux de neurones récurrents

Nous avons obtenu les résultats suivants pour notre modèle RNN :

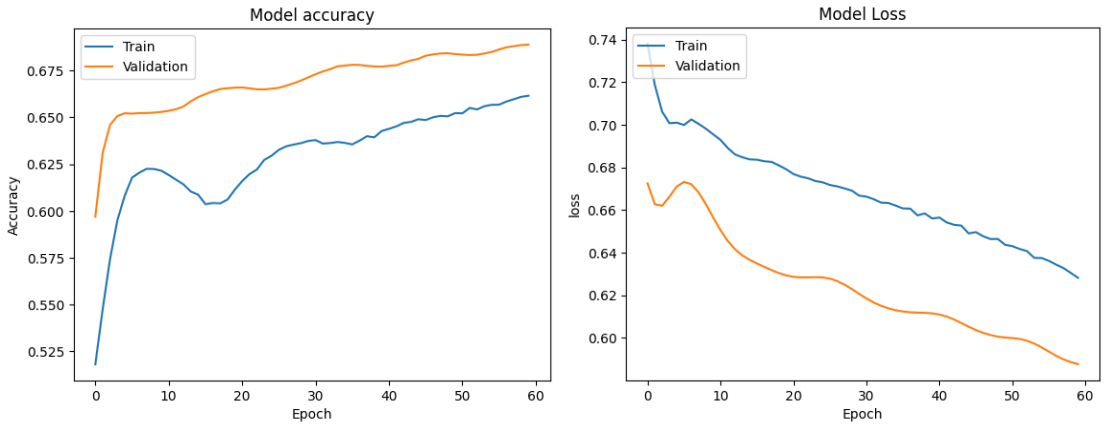

Figure 4 : plots d'accuracy et loss pour le modèle RNN

Nous avons obtenu une précision (accuracy) de 0,64 et une perte (loss) de 0,64 pour notre modèle. Cette précision de 0,64 indique que notre modèle a correctement classé 64 % des données de validation, ce qui est un résultat significatif, compte tenu de la complexité de la tâche.

En analysant notre graphe de précision, nous constatons que les courbes 'train' et 'validation' ont une évolution similaire, avec de légères fluctuations observées aux epochs 10 à 20. Il est intéressant de noter que la courbe 'train' augmente rapidement en début d'entraînement, poursuivant ensuite une augmentation progressive au fur et à mesure que le modèle s'adapte. Cependant, elle ne semble pas converger vers un plateau stable.

En ce qui concerne la courbe de perte ('loss') pour les données d'entraînement, elle diminue de manière constante et régulière à chaque epoch, reflétant l'apprentissage continu du modèle. Néanmoins, elle ne semble pas atteindre un plateau de perte minimale stable

Les graphes ci-dessous représentent le nombre de ‘onset’ et ‘wakeup’ prédit avec la méthode RNN et le nombre de ‘onset’ et ‘wakeup’ dans notre jeu de données de validation :

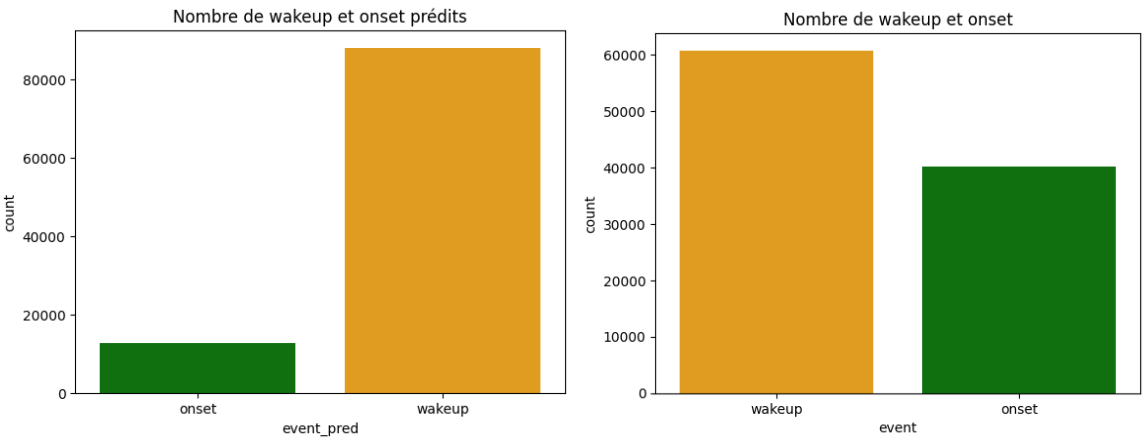

Figure 5 : Comparaison du nombre d'événements prédits avec le nombre réel dans notre Validation

Nous pouvons observer que nous avons prédit : 12 741 onset et 88 142 wakeup.

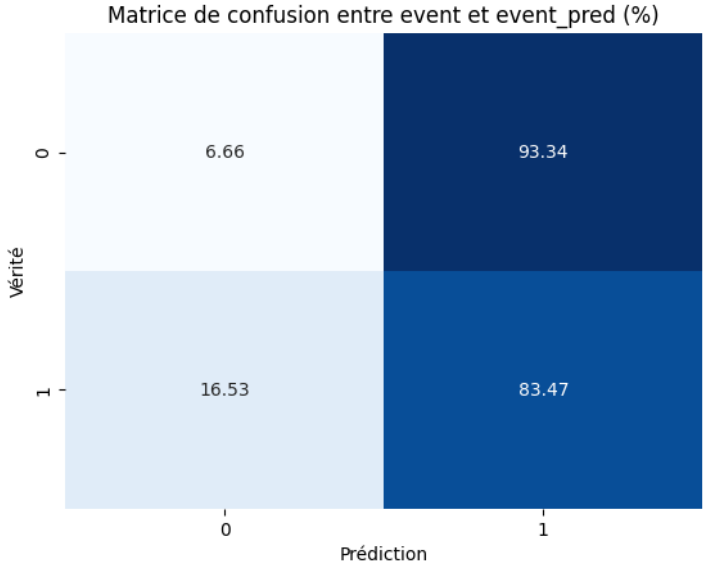

Figure 6 : Matrice de confusion entre nos évènements prédits et réels

La matrice de confusion représente la performance de notre modèle sur les données de test. Les lignes représentent les vraies classes, tandis que les colonnes représentent les classes prédites.

La première ligne représente la classe "onset": Nous avons correctement classé 6,66 % des échantillons comme appartenant à la classe "onset".

93,34 % des échantillons ont été incorrectement classés comme appartenant à la classe "wakeup" alors qu'ils appartenaient en réalité à la classe "onset".

La deuxième ligne représente la classe "wakeup" : Nous avons mal classé 16,53 % des échantillons comme appartenant à la classe "onset" alors qu'ils appartenaient en réalité à la classe "wakeup". 83,47 % des échantillons ont été correctement classés comme appartenant à la classe "wakeup".

Notre modèle semble avoir une meilleure performance pour la classe "wakeup" avec une précision de 83,47 %, tandis que pour la classe "onset", il ne classifie correctement que 6,66 % des échantillons. Cela pourrait indiquer un déséquilibre dans nos données d'apprentissage, ou que le modèle a du mal à distinguer "onset" de "wakeup".

Nous avons voulu voir quels étaient les patients pour lesquels notre modèle a le mieux prédit les événements :

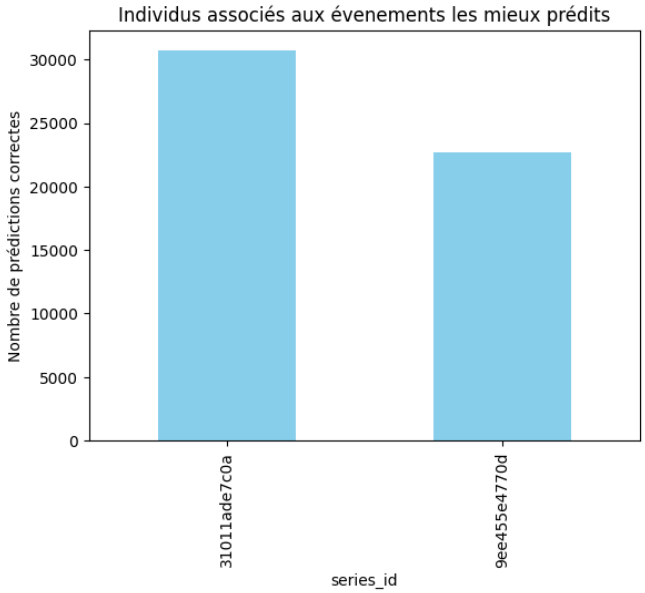

Figure 7A : Individus associés aux évènements les mieux prédits

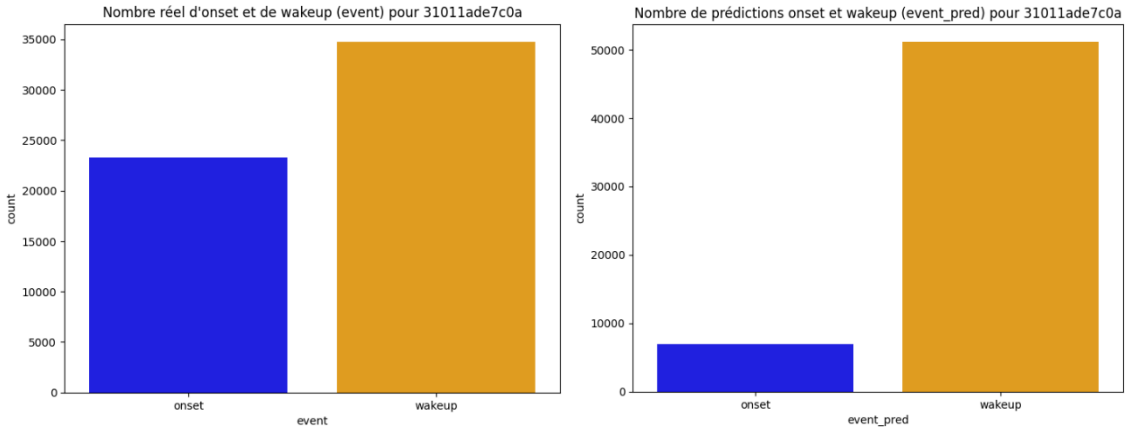

Figure 7B : Comparaison du nombre d'évènements prédits avec le nombre réel pour 1 individu

L'individu 31011ade7c0a se distingue comme étant celui pour lequel nos prédictions sont les plus précises (figure 7A). Cette performance peut être attribuée au fait qu'il dispose du

nombre de lignes le plus élevé dans notre ensemble de validation, lui conférant ainsi plus de données pour une prédiction précise. En approfondissant l'analyse, on constate une tendance déjà observée à l'échelle globale (figure 6, 7B) : les événements "wakeup" sont correctement anticipés, tandis que les prédictions des "onset" s'avèrent plus complexes. Cette spécificité pour l'individu 31011ade7c0a est en cohérence avec les tendances générales de nos prédictions sur l'ensemble des données.

Long Short-Term Memory

Nous avons obtenu les résultats suivants pour notre modèle LSTM :

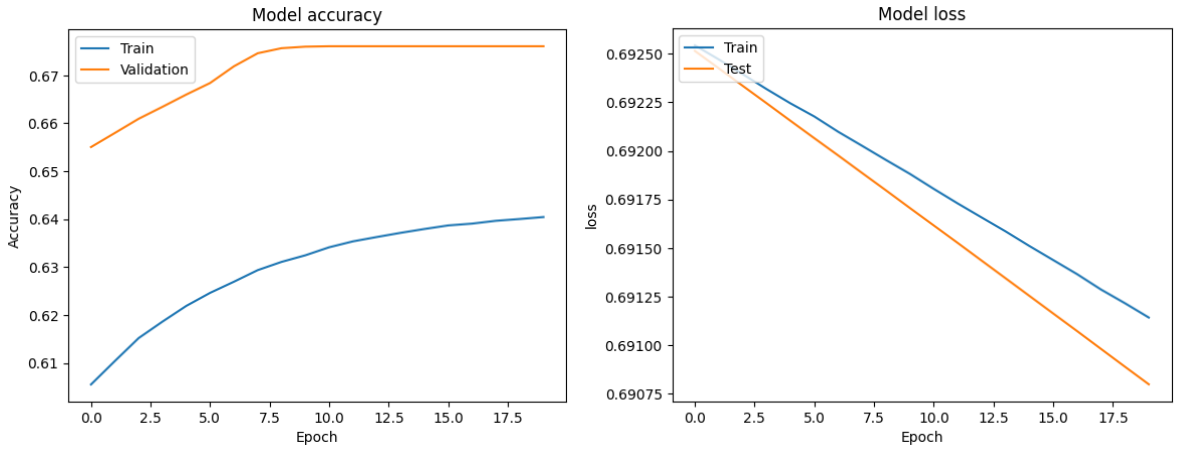

Figure 8 : plots d'accuracy et loss pour le modèle LSTM

Nous constatons que notre modèle affiche une précision (accuracy) de 0,65 avec une perte (loss) de 0,69.

En ce qui concerne le graphique de la précision, les courbes 'train' et 'validation' suivent une trajectoire similaire, bien que la précision de la courbe 'train' soit inférieure. La précision augmente progressivement au fur et à mesure que le modèle s'entraîne, mais elle n'a pas encore atteint une convergence après 20 epochs.

En ce qui concerne la courbe de perte ('loss') des données d'entraînement, elle diminue rapidement et presque linéairement à mesure des epochs. Cette diminution n’est pas progressive et régulière et n’atteint pas de plateau.

Discussion

Le but de notre ACP était de visualiser les événements wakeup et onset en fonction de la variable enmo. Cependant, étant donné le nombre important de données et le fait que nous n'avions que deux composantes principales à examiner, l'ACP pourrait ne pas être la méthode la plus appropriée pour une analyse détaillée. En effet, la concentration massive des données le long d'une seule trajectoire diagonale suggère que les variations subtiles entre les points ne sont peut-être pas entièrement capturées par les deux composantes principales. Il serait intéressant de tester d'autres méthodes de classification comme UMAP ou k-means.

Les deux modèles d'apprentissage affichent une précision similaire. Cependant, il est important de noter que ces niveaux de précision ne sont pas exceptionnels. Une précision de 64 à 65 % signifie que nos modèles classent correctement seulement environ les deux tiers des données de validation. Cela peut être considéré comme un résultat significatif, mais dépend également du contexte de l'application.

Pour la convergence, il est intéressant de noter que ni le modèle RNN ni le modèle LSTM n'ont atteint une convergence claire après 60 epochs pour RNN et 20 epochs pour LSTM. Les courbes de précision 'train' continuent de monter, ce qui peut indiquer que nos modèles ont la capacité d'apprendre davantage.

En ce qui concerne la courbe de perte ('loss'), il est notable que les modèles affichent des comportements différents. Le modèle RNN montre une diminution constante et régulière de la perte, tandis que le modèle LSTM a une diminution rapide mais presque linéaire. Cela peut indiquer que le modèle LSTM apprend plus rapidement au départ, mais il est possible que cela soit dû à un ajustement trop précis aux données d'entraînement. D'autre part, le modèle RNN semble apprendre de manière plus stable et continue, bien que sa courbe de perte ne converge pas non plus vers un plateau. Encore une fois, cela soulève des questions sur la capacité de généralisation de ces modèles.

Conclusion

Dans l'ensemble, notre analyse des données de détection des états de sommeil a révélé plusieurs aspects importants. Nous avons utilisé deux types de modèles de réseaux de neurones récurrents, le RNN et le LSTM, pour prédire les événements "onset" et "wakeup" Nos modèles ont démontré une performance modérée avec une précision avoisinant les 65%. Nous avons constaté que leur performance varient en fonction des individus, avec une meilleure précision pour ceux qui ont un plus grand nombre de données mais également que les événements "wakeup" sont plus facilement prévisibles que les événements "onset". Bien que le LSTM semble apprendre plus rapidement, le RNN montre une progression plus stable. Mais aucun des deux modèles n'a atteint une convergence claire, indiquant une marge d'amélioration, et les courbes d'apprentissage soulèvent par ailleurs des questions sur la capacité de généralisation des modèles.

Ainsi, une optimisation supplémentaire de nos modèles et de leurs hyperparamètres pourrait être nécessaire pour améliorer leur performance. De plus, il serait judicieux d'envisager l'exploration d'autres méthodes de deep learning telles que le CNN pour explorer des méthodes plus adaptées à la classification d'événements temporels.

Dans un contexte général, ce projet, ouvert à tous, a le potentiel de révolutionner la recherche sur le sommeil en fournissant des informations plus riches et plus détaillées sur les habitudes de sommeil.

De plus, les retombées positives de ce projet ne se limitent pas à la recherche. Il pourrait avoir un impact significatif sur la santé des enfants et des jeunes, en particulier ceux qui souffrent de troubles de l'humeur et du comportement. En identifiant avec précision les périodes de sommeil et d'éveil grâce aux données d'accéléromètre porté au poignet, les chercheurs seront mieux armés pour comprendre les perturbations du sommeil chez les enfants, ce qui peut avoir un impact profond sur leur bien-être émotionnel et comportemental.

Il revêt une importance considérable pour la science du sommeil et la santé des enfants. Il ouvre la voie à des avancées significatives dans la détection du sommeil grâce à l'analyse des données d'accéléromètre porté au poignet, avec le potentiel de transformer notre compréhension du sommeil, d'améliorer nos capacités de recherche et de fournir des avantages tangibles pour la santé des enfants et des jeunes.

Référence

https://www.kaggle.com/competitions/child-mind-institute-detect-sleep-states