CentraleSupélec

MACHINE LEARNING

CENTRALESUPÉLEC

OPTION MATHÉMATIQUES APPLIQUÉES

Projet - Kaggle Dreem Competition

Auteurs:

Laurent Sekonian
Antoine Gruson

Encadrants:

Arthur Tenenhaus
Valentin Thorey

January 4, 2021

Contents

Introduction

Note : Le document sera composé de 12 pages, soit 10 pages de rapport ne prenant pas en compte les pages de garde et de table des matières. Le code sera en annexe.
Le notebook python est soumis par mail en plus du rapport. Les librairies *numpy*, *pandas*, *scipy* et *scikit-learn* sont utilisées tout au long du projet.

Motivation

Ce challenge Kaggle repose sur une problématique de classification multi-classe de phase du sommeil, appelée **Automatic Sleep Stage Classification (ASSC)**.

A l'aide de 10 jeux de données réelles EEG (électroencéphalographie) (30sx50Hz pour 24688 epochs), d'un acceleromètre (x, y, z) et d'un pulsomètre, notre but est d'entraîner un modèle de classification afin de déterminer pour de nouvelles données la répartition de chaque epoch dans les 5 classes suivantes : Wake, N1, N2, N3 ou REM.

Plan

L'EEG est un des types de signaux les plus parlant et les plus utilisés dans ces applications, et il est judicieux d'en extraire les features les plus importantes afin de disposer des informations qui nous permettront de réaliser notre classification. Nous devons translater l'activité cérébrale du sommeil en informations interprétables. Nous allons donc tout d'abord extraire les features pouvant être intéressantes.

De plus, lors de problèmes de Machine Learning avec beaucoup de features, il est également important d'en réduire le nombre et de garder les plus importantes afin d'éviter l'overfitting, cela est possible en appliquant différentes méthodes (sélection de feature) ou en encore l'ACP (réduction de dimension). Nous étudierons donc également cette possibilité.

Enfin, le nerf de ce problème est l'utilisation des méthodes de Machine Learning afin de résoudre notre problème de classification, nous détaillerons donc les derniers modèles utilisés et exposerons les résultats obtenus.

Ce rapport s'articulera donc de la manière suivante :

- Extraction des features,

- Selection de feature,

- Présentation des modèles utilisés et comparaisons des résultats.

Pre-processing : Scaling

Nous n'utiliserons pas de méthode particulière de preprocessing ici : Avant et après scaling des features avec la fonction *RobustScaling* (Transforme les features en échelle $[0,1]$ prenant en compte les outliers) de *scikit-learn*, **nos résultats n'étaient pas améliorés.**

Partie 1

Features Extraction

La précision finale et la rapidité des modèles dépendant grandement des features, cette partie est une **étape cruciale de la réalisation du projet**. Nous distinguerons les features propres aux domaines temps et fréquence (resp. time-domain et frequency-domain).

1.1 Moments statistiques & Quantiles

Note : La librairie *scipy* est utilisée pour cette partie.

Les moments, liées au domaine "temps", sont parmis les plus simples que nous pouvons extraire pour chaque epoch : **absolute mean, variance, skew (assymétrie), kurtosis (épaisseur des queues de distribution)**.
Elles font également partie des features les plus utilisées pour différencier les signaux EEG. En effet, ces moments sont connus pour bien interpréter les statistiques sous-jacentes des données.
Ici nous considérons chaque epoch comme étant indépendante et ne tiendront pas compte de leur potentielle corrélation.

Nous considérons donc ces 4 moments ainsi que les quantiles 0.25, 0.5(médiane) et 0.75 pour les 7 signaux EEG ainsi que pour les données de l'accéléromètre x, y, z et du pulsomètre.

1.2 Bandes de fréquences

Note : Les librairies *yasa* et *scipy* sont utilisées pour cette partie.

Les EEG peuvent etre différenciés en 5 bandes de fréquences : **Delta** (0.5-4Hz), **Theta** (4-8Hz), **Alpha** (8-12Hz), **Beta** (12-30Hz) et Gamma (>30Hz), ces bandes fournissent des informations pour effectuer des diagnostics neurologiques. Ces features sont donc liées au frequency-domain, on **cherche à caractériser la structure spectrale du signal**. Nous ne prendrons pas en considération le Gamma ici car nos bandes de fréquence ne sont jamais supérieures à 30Hz.

Ces bandes sont connues pour être représentative des différents états du sommeil, nous pouvons, par exemple, dire que les bandes delta représentent la phase de sommeil profond (N3) ou encore pour alpha et beta les phases d'éveil.

Nous allons donc calculer la puissance moyenne (μV^2 par Hz), ou **average bandpower** du signal pour chaque bande de fréquence. Avant tout, nous avons besoin d'estimer la densité spectrale de puissance en utilisant la **méthode de Welch**, moyennant des transformations de Fourier consécutives sur des fenêtres réduites de notre signal.
Cette méthode permet de pallier au fait qu'un périodigrame classique recquiert que le signal soit stationnaire, ce qui n'est biensur pas le cas pour nos signaux EEG. Utiliser cette méthode permet de réduire la variance car cela effectue une moyenne des periodigrammes obtenus sur des fenêtres plus courtes. Avec une fréquence de résolution de :

$$F_{res} = \frac{F_s}{N} = \frac{F_s}{tF_s} = \frac{1}{t}$$

Avec F_s la fréquence de notre signal, N le nombre d'échantillons et t la durée.
Si nous avions utilisé la durée totale, nous aurions eu une résolution de $1/30 = 0.033Hz$. Dans notre cas nous souhaitons avoir une fenêtre suffisement longue pour considérer au moins deux cycles de notre fréquence

d'intérêt la plus basse ($0.5Hz$ du delta). Donc nous utiliserons des fenêtres glissantes de $2/0.5 = 4$ secondes, nous donnant ainsi des résolution de $1/4 = 0.25Hz$.

La puissance moyenne d'un bande représente sa contribution par rapport à la puissance totale du signal. Nos valeurs pour chaque seront donc **relatives** vis-à-vis de la puissance totale. Cela est intéressant pour notre problème car nous pouvons extraire, pour chaque epoch d'un signal EEG, 4 features numériques correspondants à nos 4 bandes de fréquence.

Voici un exemple de bandes de fréquence pour delta (0.5-4Hz) pour le deuxième epoch du jeu de données eeg_1.
L'axe des x représente les fréquences et l'axe des y, en échelle log pour plus de visibilité, les puissances spectrales (micro-Volts-carré par Hz (uV^2Hz)).

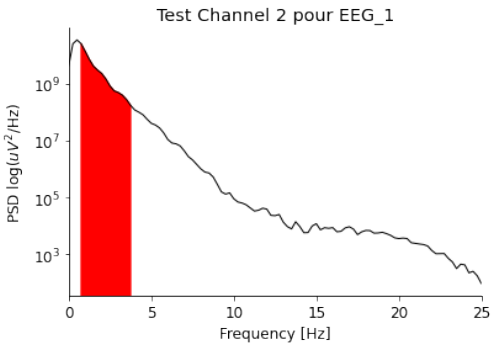

Figure 1.1: Exemple delta band power (0.5 - 4Hz) pour le 2ème epoch du set EEG_1.

De plus, pour intégrer l'air représentée en rouge ci-dessus, nous utilisons la règle de Simpsons (avec *scipy*) dont l'idée est de décomposer l'aire en un certain nombres de paraboles afin de l'approximer.

1.3 Features non linéaires

Note : Les entropies et les dimensions fractales sont calculées avec le package *entropy*.

1.3.1 Entropies

L'entropie, dans un sens statistique général, mesure l'incertitude équivalente aux différentes possibilités d'un système ou d'un évenement. Appliquée aux signaux EEG, elle en mesure donc l'incertitude du pattern, cela revient à mesurer la quantité d'information du signal.

Voici les différentes entropies que nous extrayons :

- **Approximate entropy (ApEn)** (time domain) : Technique utilisée pour quantifier la régularité et de non-predictabilité des fluctuations de time series. PLus la valeur de cette entropie est faible, plus le signal est régulier/répétitif.

- Sample entropy (SampEn) (time-domain): Modification de l'ApEn, utilisée pour évaluer la complexité d'un signal time-series. **Feature non selectionnée du à sa corrélation de** 0.99 **avec l'ApEn**.

- **Permutation entropy** (time-domain) : Quantitife la complexité d'un système dynamique en capturant la relation d'ordre entre les valeurs, et en extrayant la distribution de probabilité des patterns.

- **Singular value decomposition entropy** : Plus la valeur de cette entropie est grande, plus de vecteurs propres ortoghonaux sont requis afin d'expiquer les données de manière adéquat. En d'autres termes, cela mesure la dimensionnalité des donnéees.

- **Complexité de Lempel-Ziv** : Mesure la quantité d'incertitude présente dans une time series. En particulier, cela mesure "à quel point" les patterns présent dans le signal sont diverses.

1.3.2 Dimension Fractales

Ce sont des mesures statistiques indiquant la complexité, ou encore la quantité de similarité de certaines partie de l'epoch considérée.

- **Méthode Katz** : Utilise la somme et la moyenne des distances euclidiennes entre les points successifs de l'échantillon, et calcule la distance maximale entre le premier point et le point maximisant cette distance.

- **Méthode Higuchi** : Réputée pour être la méthode la plus précises. C'est une méthode itérative également, très utile pour tenir compte des formes d'ondes.

- **Méthode Petrosian** : Utilisée pour un calcul rapide des dimensions fractales d'un signal en translatant la série en séquences binaires.

- **Detrented fluctuation analysis** : Méthode mesurant la dimension fractale statistique d'un signal. utile pour analyser des time series non stationnaires et également à longues mémoires.

1.4 Paramètres Hjorth

Note : Ces paramètres sont calculés avec le package *pyeeg*.

Ce sont des indicateurs de propriétés statistiques de signal basé sur le time-domain. les paramètres sont l'Activité, Mobilité, et la Complexité.

- Activité : Réprésentre la puissance du signal, à vrai dire la variance. Cela indique la surface du spectre de puissance du domaine de fréquence. **Feature non selectionnée car nous utilisons déja la variance.**

- **Mobilité** : Représente la proportion d'écart-type du spectre du puissance du signal. C'est la racine carré du rapport de la variance de la dérivée du signal, sur la variance du signal.

- **Complexité** : Représente le "changement" en fréquence, c'est à dire que cela compare la similarité du signal à une onde sinusoïdale (converge vers 1 si le signal s'en rapproche). Cest calculé comme le rapport de la mobilité de la dérivée du signal sur la mobilité du signal.

1.5 Information de Fisher

Note : Ce paramètre est calculé avec le package *pyeeg*.

L'information de Fisher représente l'information relative à un paramètre contenue dans une distribution. C'est l'espérance de l'information observée.
Elle résulte d'une différentiation locale de l'entropie de Shannon dans l'espace des distributions de probabilité. **Malgré la forte corrélation de cette feature avec plusieurs autres, avions décidé de la garder et l'enleverons par la suite pour de meilleurs résultats.** Voir Partie 2.

1.6 Spindles

Note : La librairie *yasa* est utilisée pour cette partie.

Les spindles sont connus pour définir l'état N2 du sommeil (étape pendant laquelle nous passons le plus de temps). Ce sont des parties du signal totalement distinct, de fréquences 11-16Hz durant plus de 0.5s.

Nous avons décidé de selectionner une moyenne des fréquences, amplitudes ainsi que le nombre de spindles détectés par epoch comme features. Cependant, toutes les fréquences n'étaient pas retournées. **Nous avons donc choisi de ne garder que les moyennes d'amplitude et le nombre de spindles comme features**.

Partie 2

Features Selection

L'extraction de features détaillée dans la partie précédente nous permet de disposer de 196 features au total, après concaténation de toutes les features pour les 7 EEG et des moments et quantiles pour l'accéléromètre et le pulsomètre.

L'objectif de la selection de feature est d'augmenter les performances de prédiction, réduire l'overfitting et le temps d'entraînement des modèles ainsi que d'en fournir une meilleur interprétation des résultats. En effet, la redondance de certaine feature et le bruit présent peut affecter la classification.

Il est également important d'établir la différence entre la selection de feature et la réduction de dimension. Une méthode de réduction de dimension telle que l'ACP créé de nouvelles combinaisons de ces features (transformation), alors que la selection de feature en exclue certaines sans changer celles selectionnées. **Nous n'utiliserons pas de méthode de réduction de dimension ici**.

On distingue trois méthodes de selection de features :

- **Filtres** : Basé sur les caractéristiques des features, on les selectionne avant de construire le modèle : Filtres basiques (Constant/Quasi-Constant), corrélation, statistiques (F-val, chi-deux...)

- **Wrapped** : Utilise un algorithme glouton pour chaque set de features (donc très couteux), mais réputé comme étant la meilleur méthode pour un modèle de machine learning donné : Forward/Backward/Exhaustive feature selection...

- **Embedded** : Prend aussi en considération l'interaction de features. La sélection se fait pendant l'entrainement du modèle de manière itérative (Régularisation / Tree-based feature importance / RFE (Recursive Feature Elimination)...).

Du fait des coûts des algorithmes (malgré long temps d'attente et malheureuses mises en veilles des machines..) et des méthodes embedded que nous avons privilégiées, nous ne **détaillerons pas les méthodes wrapped**.

Une bonne approche serait d'**utiliser une approche hybride**, c'est à dire de combiner les différentes méthodes afin de selectionner le meilleur jeu de features possibles. Cependant nous pensons que réduire trop fortement le nombre de feature dans notre cas ferait perdre beaucoup en information.

2.1 Filtres

Les méthodes de filtres sont en général très utiles dans les cas ou on a un très grand nombre de features. Nous serons donc vigilant quant aux conclusions des différents tests ici.

Constant/quasi-constant : déterminent les features qui n'ont aucun pouvoir explicatif dans le modèle. Nous décidons de ne pas supprimer de feature via cette méthode car privilégierons les méthodes embedded.

Corrélation : Si des features sont trop corrélées, nous pouvons en exclure afin d'éviter la redondance d'informations et éviter la perte en généralisation du modèle. Nous avons quand même supprimé la feature sample entropy car corrélée à 0.99 avec l'ApEn. Nous remarquons en ayant affiché la matrice pour les features de l'eeg_1 concaténé à x,y,z et pulse, que tous les moments statistiques sont très peu corrélés entre eux et au reste. Nous décidons de ne pas supprimer d'autres features.

Tests Statistiques : Tests évaluant chaque features individuellement leur affectant un score. Ces méthodes sont efficaces si le nombre de features est très grand (500, 1000...). Les tests mutual information et ANOVA

ne nous donnent pas de résultats assez en accord pour que nous puissions supprimer des features. Nous n'avons pas réalisé de test chi-deux car il doit s'appliquer sur des données positives et certaines des notres ne le sont pas toutes (detrended_fluctuation)

2.2 Embedded (Tree-based)

Les algorithmes tree-based offrent de bonnes performances mais nous fournissent également une méthode de séléction de feature selon les **features importance**. Cela permet de savoir quelle variables sont les plus utilisées pour la prédiction de notre modèle lors de son entraînement.

La régularisation L1/L2/Elastic-Net étant plus adapté pour la régression, nous nous portons sur les méthodes de features importances pour les modèles à arbres.

2.2.1 Feature Importance

Méthode "mean decrease impurity" : permet de classer les features par **classement d'impureté** (score utilisé par l'arbre de décision pour diviser un noeud) pendant l'entrainement du modèle. Il s'agit de savoir dans quelle mesure le score a été amélioré lors du fractionnement de l'arbre à l'aide de cette variable spécifique. La variable avec laquelle un noeud est divisé est celle qui génère la plus grande amélioration sur l'impureté. La métrique utilisée en classification est la Gini par défaut (variance pour les regression trees). Sachant que cette méthode doit être utilisée avec précaution (souffre de biais et comportement inattendu en ce qui concerne les features très corrélée.), **nous retirons les features fisher_info car toutes (pour les 7 eeg) sont classées en dernier dans le classement de notre modèle XGBoost**.

2.2.2 Shap (SHapley Additive exPlanation)

SHAP est une approche basée sur la théorie des jeux et basée sur les valeurs de Shapley (calcule les contributions marginales moyennes pour chaques éléments pour obtenir une importance globale des features) Ayant pour but d'expliquer les prédictions de modèles (utile pour les règlements RGPD par exemple...), elle est très utile pour les tree-based model que nous utilisons car compatible avec XGBoost, Random Forest, etc... Nous pouvons donc avoir un apercu du classement des features importances, réputé plus fiable que ceux issus directement des modèles, car les valeurs de "Shapley" sont réputées pour fournir les résultats de classement global les plus précis et les plus cohérents.

Ci dessous nous tracons la feature importance pour le modèle XGBoost présenté ultérieurement (avec toutes les features) :

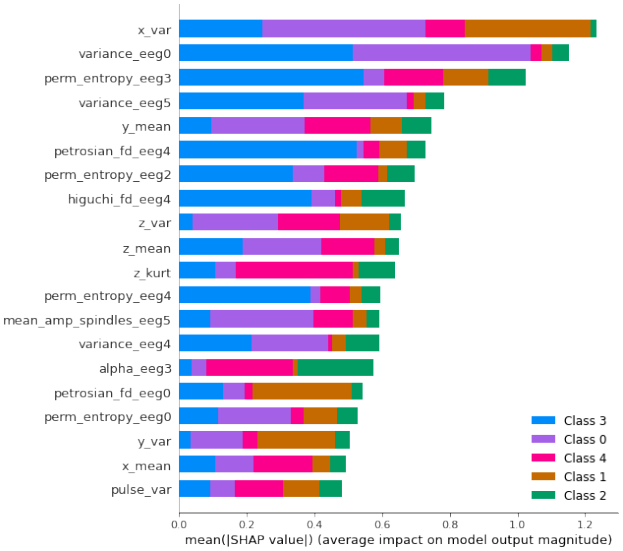

Figure 2.1: Feature importance avec SHAP pour XGBoost

Par exemple pour la classe 0 (Wake), nous voyons que les features x_var et y_mean sont relativement importantes. Sachant que l'on bouge plus en étant réveillé, cela parait cohérent.
De plus la variance du premier EEG a une forte importance pour les casses Wake et N3 (sommeil profond), nous pourrions en conclure que cet EEG (ou cette zone du cerveau) est "plus" représentatif de l'état du sommeil quant à l'amplitude du signal qu'il transmet.

2.3 Boruta / BorutaShap

Note : Le package *boostARoota* (compatible avec XGBoost), *BorutaPy* (pas compatible avec XGBoost donc utilisé avec Random Forest) et *BorutaShap* sont évoqués dans cette partie.

L'algorithme Boruta, basé sur le Random Forest et réputé pour délivrer le sous-ensemble de features le plus précis, comporte deux idées : les shadow-features et la distribution binomiale.

Le principe est d'ajouter du bruit en créant une copie des features en les mélengeant aléatoirement, ce qui donne les shadow-features. Ensuite on compare l'importance de chaque caractéristique d'origine avec un seuil (qui est la plus grande importance issue des shadow-features). Lorsque l'importance d'une caractéristique est supérieure à ce seuil, nous avons un "hit". A vrai dire, une caractéristique est utile seulement si elle peut "faire mieux" que la meilleure des shadow-features.

Pour pallier au caractère aléatoire des shadow-features, on utilise la loi binomiale avec le principe ci-dessus pour plusieurs itérations. Admettons que pour chaque itération on a une probabilité de 50% que la feature soit rejetée ou non, n itération indépendantes en forme donc une loi bonimiale.

De plus, il n'y a pas de seuil frontière entre acceptation/rejet, nous acceptons ou rejetons les features en fonction de la queue de la loi de distribution (avec $p = 0.05$ en général).
Nous avons donc une zone de rejet, d'irrésolution, et d'acceptation.

Nous sommes parti avec l'idée de seulement rejeter celles de la zone de rejet.

Cependant, cet algorithme est très coûteux... Mais il peut être amélioré en terme de métrique d'importance utilisée, c'est pour cela que le relier à la librairie SHAP présentée ci-dessus peut être judicieux.

Nous pensons que le meilleur aurait donc été d'utiliser la librairie *BorutaSharp* afin d'avoir le classement des features selon l'algorithme Boruta et la métrique des valeurs de Shapley. L'exécution de l'algorithme BorutaShap sur notre jeu de données était bien trop long ($\tilde{5}$0h d'après la barre de progression de la première itération, monopolisant nos machines pendant deux jours), donc nous ne pouvons pas comparer les résultats précédents avec celui-ci.

Nous avons utiliser la librairie *BorutaPy* avec toutes nos features basé sur un modèle Random Forest, et les 196 ont été acceptées.
De même avec la librairie *boostARoota* pour notre modèle XGBoost.

Les deux sont donc en accord, mais cela reste tout de même surprenant, car en enlevant toutes les features *Fisher_info* du au classement d'importance du XGBoost, notre score a été amélioré.

Ainsi, du à la variance de la cross-validation (nous utilisons 5 folds), nous ne pouvons pas conclure significativement que le rejet des features de fisher ait augmenté notre score (sachant que les features importances souffrent de biais et baissent fortement le score des features trop corrélées...)

Partie 3

Classification - Résultats

Note : Les librairies *scikit-learn* et *xgboost* sont utilisées ici.
Nous détaillons brièvement le Random Forest car l'avons utilisé pour Boruta et certains tests mais nous porterons surtout sur celui nous ayant donné de meilleurs résultats : XGBoost.

3.1 Cross-Validation

Pour chacun de nos entrainement de modèle, nous utilisons une cross-validation en 5 folds afin d'avoir une erreur cross-validée avant de soumettre. Après variation du nombre de folds, nous avons considéré que 5 représentait bien notre erreur de prediction après soumission. Nous avons donc choisi de garder ce paramètre pour chaque tests suivants.

3.2 Random Forest

Les forêts aléatoires proposent une amélioration du bagging spécifique aux modèles définis par des arbres binaires (CART). L'objectif est de rendre plus indépendants les arbres de l'aggrégation en ajoutant du hasard dans le choix des variables qui interviennent dans les modèles. Nous avons utilisé cette première approche pour comparer les méthodes basées sur les arbres de decisions aux autres méthodes de type K-NN, SVM, régression logistique. L'avantage est qu'elle ne travaille pas sequentiellement, elle nous à donc permis d'avoir des résultats rapidement avant d'utilsier des modèles plus complexe basés sur les arbres de décision. Nous avons opté pour une méthode améliorée du Gradient Boosting après plusieurs tests sur des modèles de ce type.

3.3 XGBoost

Le Boosting est une méthode d'ensemble permettant de réduire le biais et la variance. L'idée principale du Boosting est de s'attaquer au tradeoff Biais/Variance par le bais d'une sequence de classificateur qui permet de se concentrer au fur et à mesure sur les erreurs de classifications persistantes.

XGBoost(eXtreme Gradient Boosting) est une implémentation optimisée de l'algorithme d'arbres de boosting de gradient. Le Boosting de Gradient est un algorithme d'apprentissage supervisé dont le principe et de combiner les résultats d'un ensemble de modèles plus simple et plus faibles afin de fournir une meilleur prédiction. On parle d'ailleurs de méthode d'agrégation de modèles. L'idée est donc simple : au lieu d'utiliser un seul modèle, l'algorithme va en utiliser plusieurs qui serons ensuite combinés pour obtenir un seul résultat.

Contrairement, par exemple au Random Forest, cette façon de faire va le rendre plus lent biensur mais il va surtout permettre à l'algorithme de s'améliorer par capitalisation par rapport aux exécutions précédentes. Il commence donc par construire un premier modèle qu'il va bien sur évaluer (on est bien sur de l'apprentissage supervisé). A partir de cette première évaluation, chaque individu va être alors pondérée en fonction de la performance de la prédiction. Etc.

Il est tout de même différent car utilise une formalisation plus régularisée pour controler l'over-fitting, cela donnant de meilleure performance.

De plus, il est réputé pour sa précision. C'est un modèle linéaire et tree-based parallélisé sur une machine. Nous pouvons également y effectuer de la cross-validation via *scikit-learn* de la même manière que pour un modèle Random Forest et en extraire les features importance.

D'un point de vue code, XGBoost accepte les sparse matrix en input (pour linear booster et tree booster), accepte aussi les objectifs spécifiques et fonctions d'évaluations, et présente une forme propre de matrice "DMatrix", d'une structure optimisatn la performance.

XGBoost diffère du Bagging (Bootstrap Aggregating) car dans cette dernière le principe est de selectionner des échantillons aléatoires avec remise et d'effectuer des moyennes d'estimations afin d'augmenter la précision.

Enfin, XGBoost nous pemet de spécifier la valeurs de nombreux paramètres permettant ansi d'obtenir de meilleurs résultats. Les paramètres les plus connus pour les arbres d'apprentissages tel que le XGBoost sont :

- *Booster* : Spécification de quel booster utiliser (*gbtree* par défaut, *gblinear* ou *dart*). *gbtree* et *dart* sont basés sur des abres de décision, tandis que *gblinear* est basé sur la linéarité. Nous utilisons le paramètre par défaut *gbtree*.

- *objective* : "objectif" du modèle, classifiaction (multi-classe ou non), etc...

- *eta* : taille du pas de shrinkage utilisé dans la mise à jour pour prevenir l'overfitting. Après chaque étape de boosting, eta shrinks les poids des features pour rendre le processus de boosting plus conservateur.

- *num_class* : le nombre de classe si le paramètre objective est mis sur multi-classe (toujours 5 pour nous.

- *learning_rate* : taille du step de shrinkage utilisé pour limiter l'over-fitting. Compris entre 0 et 1.

- *max_depth* : La profondeur maximum de l'arbre. Augmenter ce nombre rend le modèle plus complexe et augmente possiblement l'overfitting.

- *n_estimators* est le nombre d'arbres boostés à fitter.

Avec nos 196 features et un modèle XGBoost de parmamètres :

- *objective* 'multi:softprob', (pour le training multiclass)

- *n_estimators* (ou *num_bosst_round*) = 500 (nombre d'itérations)

- *eta* = 0.3, (training step pour chaque itération)

- *max_depth* = 5, (profondeur de chaque arbre

- *learning_rate* 0.1,

Nous obtenons un score de 71.977% avec les features *Fisher_info*, et 72.444% après les avoir supprimées suite au features importances données par XGBoost.

3.3.1 Grid-search

Le **grid search** est une technique utilisée afin d'améliorer le modèle, le principe est d'effectuer une recherche exhaustive sur différents paramètres spécifiés.

Nous avons utilisé la librairie *scikit learn* et plus précisement la fonction *GridSearchCV* sur les plages de paramètres suivantes :

- *n_estimators* : [500, 600, 700],

- *eta*: [0.05, 0.1, 0.3],

- *max_depth*: [6, 9, 12],

- *learning_rate*: [0.1, 0.15]

Nous avons finalement obtenu les paramètres comme étant optimaux :

- *n_estimators* = 500

- *eta* = 0.1,

- *max_depth* = 6,

- *learning_rate* = 0.15

Après entrainement de nos dernières données puis soumission (sans les features *Fisher_info*, **nous obtenons un score de 73.150% (score final)**

3.4 Hypnogramme

Voici l'hypnogramme issu des dernières prédictions du modèle XGBoost :

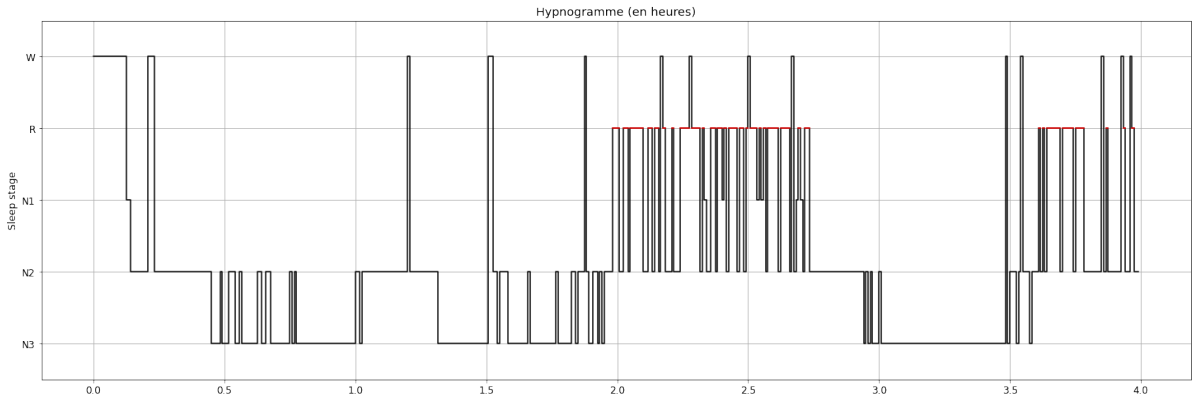

Figure 3.1: Hypnogramme (4 heures) réalisé sur notre dernère prédiction

3.5 Conclusion

L'utilisation et l'extraction des features représente une grande partie du travail effectué lors de ce défi *Kaggle*. En effet l'analyse et l'identification d'indicateur pouvant etre exogènes a necessité à la fois d'acquerir des notions en traitement du signal mais aussi de faire un état de l'art des elements pouvant caractérisér un signal (EEG en particulier). La recherche du modèle nous a ensuite amené à tester de nombreux classificateurs tel que le SVM, Knn, Decison Tree pour au final approfondir les modèles se basant sur les arbres decisionnels. Il s'est avéré que les methodes d'arbres comme le Random Forest et le modèle final XGboost nous on permis d'améliorer notre score jusqu'a 73.150%.

Il reste cependant des méthodes tel que le stacking qui restent à explorer. L'utilisation d'un empilements de plusieurs modèles afin de joindre leurs capacités prédictives et atténuer leurs faiblesses respectives est une piste à approfondir.