Machine learning pour la classification de phases de sommeil

Alexandre Herbert et Baptiste Turpin

Sommaire

I Contexte

Le sommeil tient une place majeure dans notre santé et notre bien-être. Il se décompose en une succession de phases répétées par cycle :

— Phase 0 : endormissement (individu réveillé)
— Phase 1 : sommeil léger
— Phases 2 et 3 : sommeil profond
— Phase 4 : sommeil paradoxal ou REM (Rapid Eye Movement)

Le travail de DREEM repose sur la mise en place de solutions permettant de diagnostiquer les problèmes de sommeil. Il s'agit d'identifier lors d'une session de sommeil l'enchainement des différentes phases pour comprendre l'origine du trouble. A cette fin, DREEM a lancé sur le marché un bandeau permettant de récupérer 3 catégories de signaux pendant le sommeil de l'individu :

— L'activité éléctrique du cerveau (par 7 électroencéphalographie)
— L'accélération (par 3 accéléromètrie)
— L'absorption de la lumière infrarouge par l'hémoglobine, caractéristique de la teneur en oxygène du sang (par 1 oxymétrie colorimétrique)

L'objectif est de prédire la phase du sommeil à partir de ces 11 signaux relevés. A cette fin, DREEM travaille sur un modèle d'apprentissage supervisé pour la classification en 5 phases de sommeil. Des spécialistes du sommeil ont donc été sollicité pour étiqueter un ensemble d'échantillons, chaque échantillon étant constitué des 11 signaux capturés sur une période de 30".
L'objectif du présent projet est :

— De sélectionner un modèle de Machine Learning et des hyperparamètres efficaces pour le problème de classification proposé
— D'identifier des features pertinents pour chacun des signaux permettant de discriminer les phases du sommeil
— D'appliquer la technique d'apprentissage par validation croisée et un prétraitement des données adapté au problème

Nous reviendrons dans ce qui suit sur les différents choix que nous avons opérés : modèle, prétraitement, extraction de features, entrainement. Nous présenterons ensuite de manière critique les résultats obtenus, avant d'envisager des perspectives d'amélioration.

II Choix d'implémentation

1 Choix du modèle

Dans un tout premier temps, nous avons utilisé un apprentissage non-supervisé de type k-means. Cette méthode, très peu performante, visait juste à évaluer la qualité des features extraits, pour savoir s'ls permettaient de discriminer efficacement les phases du sommeil. Aucune performance probante n'a pu être tirée de cette analyse.

L'analyse de l'existant nous a ensuite rapidement orienté vers les modèles évalués comme étant les plus performants pour ce problème : le Random Forest ou le simple decision tree.

Nous nous sommes finalement tournés vers le Random Forest, afin d'obtenir une classification plus fine qu'un simple decision tree, et puisque nous ne cherchons pas ici à déterminer un sens "physique" pour les noeuds.

Nous avons cherché à optimiser les hyperparamètres suivants (attributs de la classe sklearn.ensemble.RandomForestClassifier) :

— Le nombre d'arbres de la forêt (n_estimators)
— Le critère de qualité d'une division (criterion)
— La profondeur maximale (max_depth)
— Le nombre minimal d'échantillons pour une division (min_samples_split)
— Le nombre minimal d'échantillons par feuille (min_samples_leaf)
— Le taux minimal de réduction du critère de qualité autorisant une division (min_impurity_decrease)

TABLEAU MONTRANT UN CROISEMENT DES PARAMETRES POUR OPTIMISATION ?

Finalement, les valeurs suivantes ont été choisies pour ces hyperparamètres :

Hyperparamètre	**Valeur**
n_estimators	?
criterion	?
max_depth	?
min_samples_split	?
min_samples_leaf	?
min_impurity_decrease	?

2 Sélection des méthodes de validation algorithmique

L'apprentissage est réalisé par validation croisée. Le découpage du dataset d'entrainement est réalisé avec la classe

sklearn.model_selection.StratifiedKFold. Le nombre de folds (n_folds) a été fixé à 5. Le dataset est mélangé alétaoirement (shuffle = True).

Nous calculons à partir de cet apprentissage le F1-score ainsi que la matrice de confusion obtenus par la validation.

III Selection de features

Nous avons dans un premier temps affiché les signaux temporels du dataset d'entrainement afin d'établir des caractéristiques des différentes phases de sommeil.

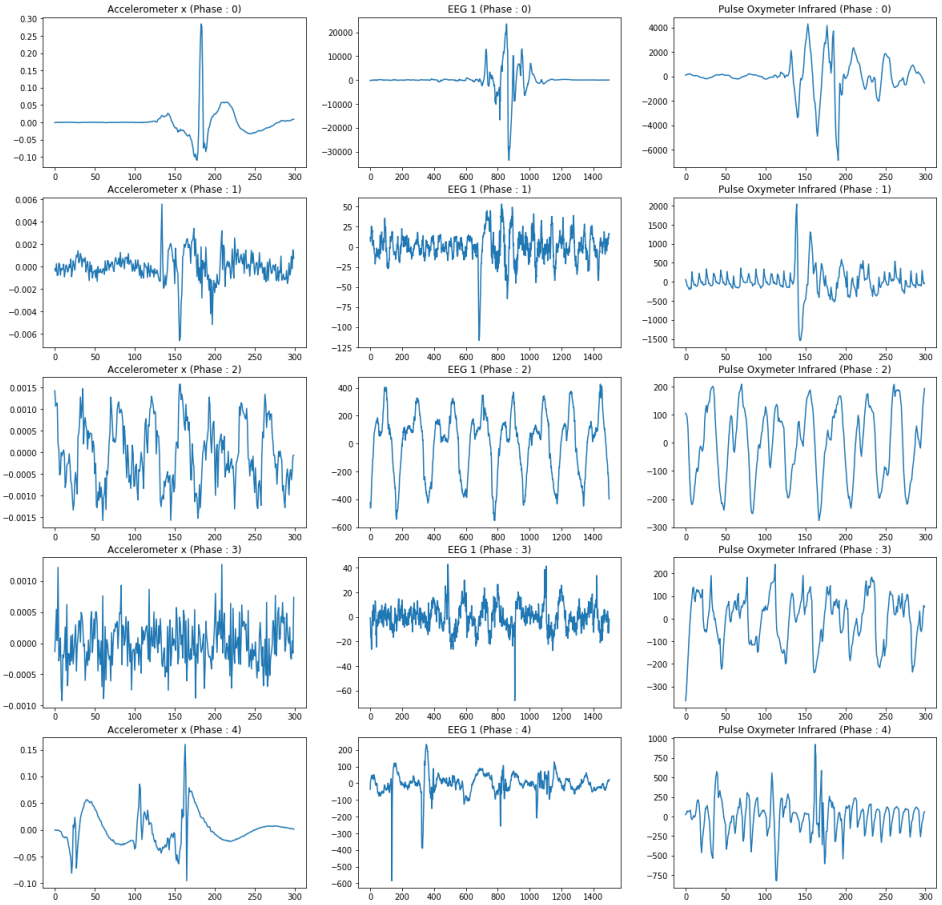

FIGURE 1 – Signaux temporels pour différentes phases

Nous pouvons faire les premières observations suivantes :

— Les signaux de la phase 0 sont d'amplitude très élevée, mais avec de faibles fluctuations, et sans motif répété.
— L'accéléromètre de la phase 4 ne présente que peu de fluctuation, et aucun motif répété.

Plus généralement, on constate que :

— Les signaux se composent de motifs répétés pour les phases 1, 2 et 3.
— La forme des motifs, leur amplitude et leur fréquence d'apparition semble varier en fonction de la phase

Cela nous amène à considérer en supplément les caractéristiques fréquentielles des signaux. Nous avons calculé la valeur absolue de la transformée de Fourier des signaux temporels (figure 2).

On constate sur la comparaison des FFT que l'amplitude maximale observée varie beaucoup en fonction de la phase. En outre, en ne considérant que la forme de la FFT et la distribution des points dasn la fenêtre d'affichage, on tire les observations suivantes :

— EEGs : la FFT est plus ou moins bien dessinée en fonction de la phase : pour une plage de fréquences réduite donnée, il semblerait que la variance en amplitudes des différents points soit plus ou moins forte. Dans le cas ci-dessus, la phase 2 est bien dessinée, tandis que les phases 3 et 4 présente un nuage de points étendu.
— Accéléromètres : des points isolés en haut à droite de la fenêtre d'affichage peuvent être observés sur les phases 0, 1 et 3.
— Oxymétrie colorimétrique : on distingue plusieurs pics, qui sont plus ou moins espacés et d'amplitudes plus ou moins équilibrées en fonction des phases.

De toutes ces observations, nous avons décidé d'extraire les features suivants. Dans la description, "signal" est la liste des ordonnées du signal : ordonnées temporelles (resp fréquentielles), suivant que la méthode est appliquée au signal temporel (resp à la FFT)) :

distanceMinMax
La liste signal est partitionnée en n intervalles disjoints : $signal = \bigcup_{i=1}^{n} signal_i$.
Puis distanceMinMax $= \sum_{i=1}^{n} max(signal_i) - min(signal_i)$.

maxAmp

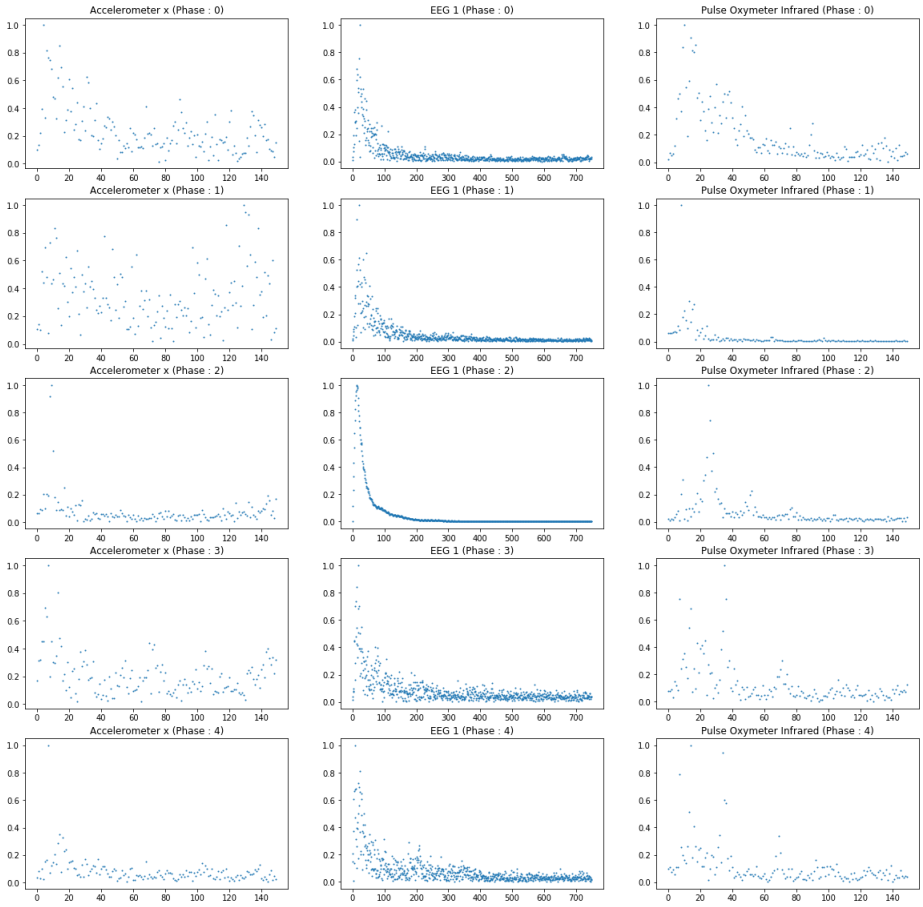

FIGURE 2 – FFT des signaux pour différentes phases

maxAmp = $max(signal)$

freqMinLimitAmp

Soit une amplitude A donnée.
freqMinLimitAmp = $min(f_t)$ tq $\forall f > f_t, signal(f) < A$

nbPikes

Soit une amplitude A et une largeur d'intervalle W données.
Un intervalle de largeur W $signal_{ext}$ extrait de signal est un unique pic ssi $min(signal_{ext}) > A$.
nbPikes est le nombre de pics distincts de signal.

indexMaxAmp

Soit une largeur d'intervalle W.

indexMaxAmp = $\underset{\substack{signal_{ext} \subset signal \\ len(signal_{ext})=W}}{\text{argmax}} \; mean(signal_{ext})$

meanDiffNeighb

Soit une largeur d'intervalle W.

stdDeviationNb

upperRight

Soit une amplitude A et une abscisse x. upperRight = $\sum_{a>x} (\mathbf{1}(signal(a) > A))$.

mean

mean = mean(signal)

meanOfAbs

meanOfAbs = mean(abs(signal))

maxOfAbs

maxOfAbs = max(abs(signal))

minOfAbs

minOfAbs = min(abs(signal))

IV Pré-traitement des données

Nous avons dans un premier temps pu constater que le dataset d'entrainement est déséquilibré :

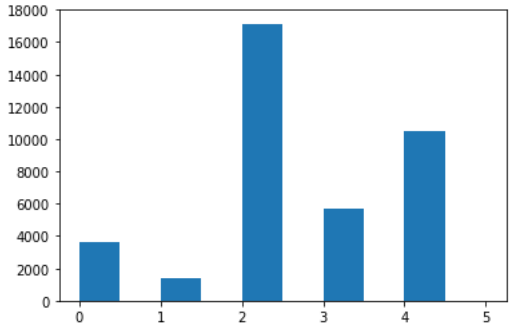

FIGURE 3 – Distribution du dataset d'entrainement entre les classes

Pour cette raison, nous avons appliqué deux méthodes :

— Réaliser la validation croisée sur un dataset équilibré entre les classes : tous les échantillons de la classe 1 (la moins représentée avec 1353 échantillons) sont conservés ; les échantillons des autres classes sont tirés aléatoirement. On passe ainsi d'un dataset de 38289 échantillons à un nouveau dataset équilibré de 6765 échantillons.
— Réaliser la validation croisée sur l'ensemble du dataset, et appliquer des poids aux classes pour pénaliser les mauvais classements associés aux classes les moins représentées.

Par ailleurs, la valeur absolue de la FFT d'un signal est symétrique. Nous pouvons donc nous contenter de la première moitié de cette FFT.

Enfin, pour certains features, on désire extraire de l'information dans la forme du signal ou de la FFT. Dans ce cas précis, les amplitudes sont normalisées dans la fourchette [0,1].

V Optimisation des hyperparamètres des features

Certains features listés plus tôt présentent des hyperparamètres : amplitude, largeur d'intervalle, abscisse. Nous avons décidé d'optimiser les valeurs de ces hyperparamètres séparément. Nous avons donc réalisé des apprentissages par validation croisée pour toutes les matrices de design élémentaires. Chaque matrice de design élémentaire est constituée d'un seul feature, appliqué à une seule catégorie de signal (EEG, accéléromètre ou oxymétrie colorimétrique), soit sur le signal temporel ou sur la FFT.

Le but de cette optimisation est double :

— Identifier parmi les familles (feature, catégorie de signal, temporel/fréquentiel) celles qui apportent les meilleurs résultats (cad dont le score-F1 atteint est le plus élevé)
— Trouver les valeurs des hyperparamètres les plus pertinentes pour chaque famille (feature, catégorie de signal, temporel/fréquentiel)

Voici ci-dessous un extrait de l'analyse opérée :

EXTRAIT DU TABLEAU

A l'issue de cette analyse, nous avons conservé les features suivants pour notre modèle :

TABLEAU

La matrice de design finale, obtenue par concaténation des matrices élémentaires, contient donc XXX colonnes.

VI Résultats

VII Critiques et perspectives

Comme expliqué précédemment, nous avons optimisé les hyperparamètres des features en validation croisée. En travaillant ainsi, nous savions qu'il y avait un risque d'overfitter les set de validation. Ce phénomène s'est révélé être bien plus important que nous ne le craignions, et le F1-score obtenu sur le dataset de test était nettement plus faible que celui obtenu en validation croisée.

```
X_test_fft = h5py.File('data/X_test_fft.h5')

def buildAndSaveMatrix(h5file_freq, methodOne, param,
    list_bool_extract_signal, name_save):
    rep = extractFeatureAll(h5file_freq , methodOne , param ,
        list_bool_extract_signal)
    temp_var_file = open("design_matrix/elem/" + name_save + '.txt','wb')
    pickle.dump(rep , temp_var_file)
    temp_var_file.close()
     len(bidule)
```