MASTER EN SCIENCES MATHÉMATIQUES À FINALITÉ SPÉCIALISÉE EN DATA SCIENCE

Une approche spectrale pour la classification d'états de conscience

DARDENNE, Charline

Award date:
2022

Awarding institution:
Universite de Namur

UNIVERSITE DE NAMUR

Faculté des Sciences

UNE APPROCHE SPECTRALE POUR LA CLASSIFICATION D'ÉTATS DE CONSCIENCE

Mémoire présenté pour l'obtention du grade académique de master en

« Sciences Mathématiques à finalité spécialisée en Data Science »

Charline DARDENNE
Promoteurs : Alexandre MAUROY et Benoît FRÉNAY

Juin 2022

Remerciements

J'aimerais commencer par remercier mes deux promoteurs Monsieur Alexandre Mauroy et Monsieur Benoît Frénay qui m'ont encouragée tout au long de l'élaboration de ce mémoire. Merci de m'avoir encadrée avec bienveillance, d'avoir pris le temps de répondre à toutes mes questions et d'avoir lu (et relu) ce travail avec attention.

Je remercie également Marvyn Gulina pour son aide précieuse lors de l'écriture et la correction de ce manuscrit. Merci pour ta gentillesse, pour ta disponibilité et pour tes nombreux conseils.

Ensuite, j'aimerais remercier Madame Muriel Demonthy qui a réussi à me communiquer sa passion pour les mathématiques durant mes deux dernières années dans l'enseignement secondaire. Vous m'avez encouragée à faire des études dans ce domaine et malgré les difficultés traversées pour arriver jusqu'ici, je ne regrette pas d'avoir fait ce choix grâce à vous.

Je tiens également à remercier ma famille et plus particulièrement mes parents qui, malgré les moments de doute, ont toujours cru en moi et en mes capacités à terminer ce cursus universitaire. À votre façon, vous m'avez poussé à me dépasser pour me permettre d'arriver là où j'en suis.

Enfin, je remercie mes amis et mon compagnon pour leur soutien inconditionnel lors de la réalisation de ce mémoire mais pas uniquement. Merci pour les nombreux messages d'encouragement durant les blocus et les sessions d'examens, vous avez été une source de motivation dans les moments difficiles. Merci également d'avoir été mes acolytes de guindaille (d'ailleurs, gloire au cercle math), vous me permettez de finir ces nombreuses années d'étude avec des souvenirs extraordinaires plein la tête.

Merci à tous, Charline.

Résumé

Dans ce mémoire, nous proposons une méthode permettant de détecter les états de conscience de personnes n'étant pas dans un état physiologique normal, à partir de données temporelles récoltées via la technique d'imagerie par résonance magnétique fonctionnelle. Cette approche se base sur l'opérateur de Koopman permettant de caractériser la dynamique de séries temporelles, et sur des techniques de *Machine Learning* permettant d'automatiser l'analyse de ces caractérisations. Plus précisément, nous avons combiné des méthodes de décompositions en modes dynamiques approximant les propriétés spectrales de l'opérateur de Koopman avec des techniques de classification supervisée telles que les arbres de décision. Tout au long de ce mémoire, nous avons tenté d'optimiser la performance de la classification des états de conscience grâce à plusieurs pistes d'améliorations. Celles-ci sont la sélection de *features* adéquates, la variation d'un paramètre du modèle de classification *Extra-trees*, l'utilisation d'autres classificateurs, l'augmentation des données, le modèle du sac de mots, la méthode de Hankel et la combinaison de plusieurs méthodes de classification.

Mots-clés : Classification, États de conscience, Opérateur de Koopman, Décomposition en modes dynamiques, *Extra-trees*, Modèle du sac de mots, Augmentation des données, Hankel.

Abstract

In this thesis, we propose a method for detecting states of consciousness of people who are not in a normal physiological state. The work is based on temporal data collected with functional magnetic resonance imaging. We use the Koopman operator to characterise the dynamics of time series, and Machine Learning techniques to automate the analysis of these characterisations. More precisely, we have combined dynamic mode decomposition approximating the spectral properties of the Koopman operator with supervised classification techniques. Throughout this thesis, we have attempted to optimise the performance of the classification of states of consciousness through several improvement ideas. These are the selection of suitable features, the variation of a parameter of the Extra-trees model, the use of other classifiers, data augmentation, the bag-of-words model, the Hankel method and the combination of classification methods.

Key-words : Classification, States of consciousness, Koopman operator, Dynamic mode decomposition, Extra-trees, Bag-of-words model, Data augmentation, Hankel.

Table des matières

Introduction

Depuis des siècles, la médecine progresse pour allonger la vie des gens et ces dernières décennies, cette évolution a été encore plus fulgurante grâce au développement de nouvelles technologies. Par exemple, les techniques d'électrocardiographie améliorent l'étude du coeur, tandis que les techniques d'imagerie médicale produisent des images très précises de l'intérieur du corps humain. Parmi ces dernières se trouve l'imagerie par résonance magnétique fonctionnelle qui permet d'étudier le fonctionnement du cerveau. Grâce à cette technique d'imagerie médicale, les médecins peuvent visualiser l'activité cérébrale des patients à tout moment. Elle fournit donc de précieuses informations sur l'état de santé de ceux-ci. Un des domaines dans lequel cette technologie est particulièrement prometteuse est la détection des états de conscience d'un patient n'étant pas dans un état physiologique normal. C'est sur cette problématique que se concentre ce mémoire.

De nombreuses causes peuvent plonger un patient dans un état de conscience altéré et, malheureusement, il n'existe pas d'outil permettant de mesurer directement la conscience. L'imagerie par résonance magnétique fonctionnelle fournit des signaux temporels qui caractérisent l'évolution de l'état de conscience d'un patient, mais il est compliqué d'établir un diagnostic sur base de ces signaux bruts. La détermination de l'état de conscience d'une personne est pourtant primordiale pour permettre au personnel médical d'apporter à celle-ci une prise en charge et des soins adéquats.

Dans ce contexte, le but de ce mémoire est de proposer des méthodes permettant d'automatiser l'analyse des signaux obtenus via l'imagerie par résonance magnétique fonctionnelle, afin de faciliter la détection des états de conscience de patients n'étant pas dans un état physiologique normal. Pour ce faire, nous allons d'abord faire appel à la théorie de l'opérateur de Koopman, car il permet de caractériser la dynamique de signaux temporels. Ensuite, nous allons employer des techniques de *Machine Learning* pour tirer parti des informations fournies par l'opérateur de Koopman, dans le but d'effectuer une classification des états de conscience.

Ce mémoire s'inscrit dans la continuité d'un autre mémoire traitant du même sujet et présenté il y a quelques années [8]. Durant la recherche, nous avons d'abord reproduit les expériences effectuées dans le mémoire précédent, puis nous avons exploré plusieurs pistes pour améliorer la classification des états de conscience. Plus précisément, nous avons augmenté les données alimentant la classification, mis en pratique un modèle permettant d'effectuer une nouvelle sélection de *features* (le modèle du sac de mots), testé une méthode basée sur la matrice de Hankel, combiné plusieurs modèles de classification, etc. Nous verrons que les résultats obtenus avec ces méthodes sont encourageants et nous laissent penser que les états de conscience peuvent effectivement être détectés.

Le présent document comporte cinq chapitres. Le premier chapitre est dédié à l'aspect médical de la recherche. Nous expliquons d'abord les enjeux principaux que représente la détection des états de conscience d'une personne dans un état de conscience altéré. En effet, elle permet au personnel médical de suivre l'état de santé d'un patient dans un état végétatif et de lui administrer les soins adéquats. Ensuite, nous introduisons la technique d'imagerie par résonance magnétique fonctionnelle avec laquelle les données réelles utilisées dans ce mémoire ont été produites. Cette technologie capture les modifications hémodynamiques dans le cerveau d'un patient lorsque son activité cérébrale varie. Nous terminons par fournir quelques explications concernant les données réelles récoltées par le groupe *Coma Science Group* de l'Université de Liège et utilisées pour classifier les états de conscience.

Dans le deuxième chapitre, nous présentons la théorie de l'opérateur de Koopman, un opérateur linaire de dimension infinie décrivant un système dynamique associé à une application non-linéaire. Comme cet opérateur est de dimension infinie, nous développons ensuite deux méthodes de décomposition en modes dynamiques permettant d'approximer les propriétés spectrales de cet opérateur. Pour terminer, nous analysons le comportement d'une approximation des valeurs propres de l'opérateur de Koopman et nous justifions le fait que des techniques de *Machine Learning* doivent être employées pour classifier des états de conscience.

Le troisième chapitre est consacré à la présentation des concepts de *Machine Leraning* utilisés dans ce mémoire, et plus spécialement à ceux de l'apprentissage supervisé. Celui-ci consiste à superviser l'apprentissage d'une machine en lui montrant un exemple de tâche à effectuer sur base de données d'entraînement, pour ensuite tester sa capacité à reproduire cette tâche sur d'autres données de test. Pour répondre à la problématique de ce mémoire, c'est-à-dire la détection d'états de conscience, nous portons notre attention sur une technique de classification supervisée basée sur les arbres de décision, appelée le modèle des *Extremely Randomized Trees*. Ensuite, nous introduisons une méthode permettant d'extraire des *features* sous forme de fréquence d'apparition de "mots", appelé le modèle du sac de mots.

Le quatrième et le cinquième chapitre sont dédiés à la présentation des résultats obtenus durant la recherche. Dans le quatrième chapitre, nous présentons d'abord les premières classifications effectuées. Pour ces classifications, nous varions la sélection des *features* parmi les informations spectrales obtenues via un algorithme de décomposition en modes dynamiques. Ensuite, nous essayons d'améliorer les résultats en modifiant la valeur de l'un des paramètres du classificateur considéré ou en utilisant d'autres modèles de classification. Dans le cinquième chapitre, nous testons des pistes d'amélioration de la classification des états de conscience avec des techniques moins conventionnelles telles que l'augmentation des données, le modèle du sac de mots, la décomposition en modes dynamiques de Hankel ou la combinaison de plusieurs méthodes de classification.

Chapitre 1

Détection d'états de conscience en imagerie médicale

Ce chapitre est dédié à l'aspect médical de ce mémoire. La première partie du chapitre introduit le motif principal du mémoire car nous y expliquons l'importance que représente l'évaluation de l'état de conscience d'une personne n'étant pas dans un état physiologique normal. Ensuite, dans la deuxième partie du chapitre, nous détaillons la technique avec laquelle les données utilisées dans ce mémoire ont été obtenues, à savoir l'imagerie par résonance magnétique fonctionnelle. Pour terminer, nous donnons quelques informations concernant ces données.

1.1 Évaluation des états de conscience

Lorsqu'une personne est ou tombe dans le coma, de nombreux membres du personnel médical sont sollicités comme les réanimateurs, les neuroradiologues, les neurologues, les neurochirurgiens et les neuropsychologues. Ces personnes ont la responsabilité d'établir la cause du coma, mais aussi de déterminer de façon aussi précise que possible le pronostic en termes d'éveil. Pour cela, ils doivent établir l'état de conscience du patient afin de prendre la décision de poursuivre les soins ou, lorsque le pronostic est plus grave, d'alléger voire d'arrêter les traitements. Cette dernière situation est appelée désescalade thérapeutique et s'oppose à un acharnement thérapeutique, c'est-à-dire une obstination à maintenir un patient en vie alors que les traitements n'améliorent plus sa qualité de vie et ne répondent pas à ses désirs.

Les troubles sévères de la conscience peuvent se déclarer de diverses manières et, dans le meilleur des cas, le patient évoluera du coma à la récupération en passant par plusieurs stades représentés sur la Figure 1.1. A l'opposé de l'état de conscience normal, le coma est caractérisé par une absence complète d'éveil ainsi qu'une absence de conscience de soi et de l'environnement. Ce coma peut durer quelques jours voire quelques semaines, mais lors du réveil, si le patient ne passe pas par une phase de récupération, il est diagnostiqué dans un état végétatif. L'état végétatif se différencie du coma car il est rythmé par des cycles de sommeil puis de réveil. Par contre, tout comme pour le coma, le patient ne démontre aucun signe de conscience, ni de lui-même, ni de son environnement. Il y a donc une dissociation entre le niveau d'éveil et la conscience de soi et de l'environnement. Cet état végétatif peut être permanent ou une transition vers une récupération. Ensuite, nous parlerons d'état de conscience minimale lorsque les patients quittent cet état végétatif et montrent des signes d'une présence de conscience. Ces signes peuvent être

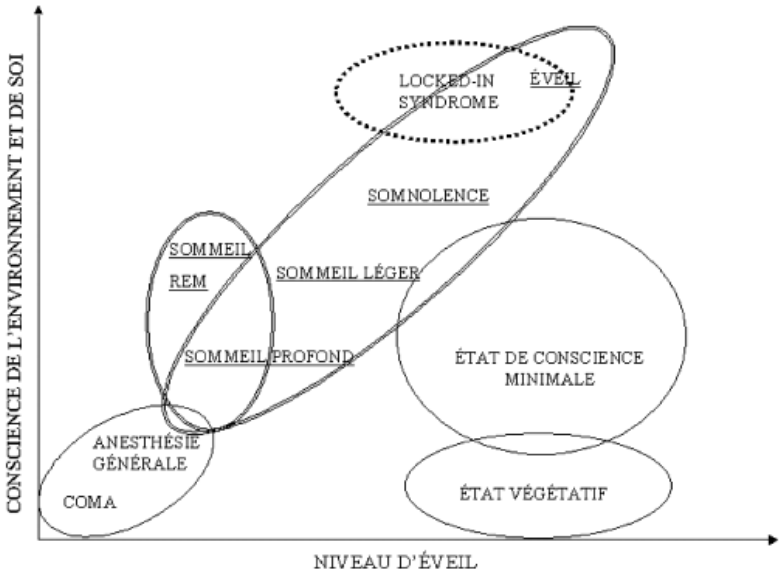

FIGURE 1.1 – Schéma représentant la corrélation du niveau d'éveil avec la conscience de soi et de l'environnement qui sont deux composantes majeures de la conscience. Dans ce schéma, les états physiologiques normaux sont soulignés et le sommeil REM (*rapid eye movement sleep*) signifie "sommeil à mouvements occulaires rapides". Source : [36].

une réaction à un stimuli visuel ou auditif, bien qu'ils soient incapables de communiquer de manière fonctionnelle avec l'environnement. Cette communication fonctionnelle est celle qui permet d'échanger avec nos semblables quotidiennement. Il s'agit, par exemple, du contact visuel ou du langage corporel. Enfin, la transition de l'état de conscience minimale vers l'état de conscience normal se caractérise par une capacité du patient à communiquer de manière fonctionnelle avec son environnement et/ou de manipuler, d'utiliser des objets de la vie courante [36].

Une autre situation qui peut se produire, mais qui est plus rare, est celle dans laquelle les patients entrent en *locked-in syndrome*, aussi appelé pseudo-coma. Ce syndrome se caractérise par une paralysie presque complète, alors que les patients sont éveillés et présentent une conservation de la conscience ainsi que des facultés cognitives. En effet, sur la Figure 1.1, nous pouvons observer que leur niveau d'éveil ainsi que leur conscience de soi et de leur environnement sont positivement corrélés. De plus, ils ressentent encore le toucher et la douleur. Les personnes présentant ce syndrome ne peuvent pas bouger leurs membres ni le bas de leur visage comme la mâchoire. Ils ne peuvent donc pas mâcher, avaler ou parler, mais ils peuvent généralement voir, entendre et communiquer grâce à leurs yeux. En effet, les yeux et les paupières sont les seules parties du corps qu'ils peuvent contrôler et ils communiquent en bougeant les yeux de haut en bas ou en clignant [25].

Cette évaluation de l'état de conscience d'un patient est d'autant plus cruciale qu'elle permet d'informer la famille de celui-ci de manière fiable d'une part, et de gérer les ressources financières des instances médicales d'une autre part en apportant d'emblée des soins adéquats, ceux-ci étant assez coûteux. Cependant, il est souvent très difficile de déceler une présence de conscience chez un patient dans un état de conscience altérée et cela mène à de fréquentes erreurs de diagnostic.

1.2 Imagerie par résonance magnétique fonctionnelle

Au vu de l'enjeu que cela représente, de nombreuses techniques se sont développées au fil des années afin d'évaluer les capacités cognitives résiduelles des patients en état de conscience altérée. Dans cette section, nous allons développer une technique de neuro-imagerie fonctionnelle : l'imagerie par résonance magnétique fonctionnelle, aussi notée IRMf. À l'origine, cette technique est destinée à l'étude du fonctionnement du cerveau en général, mais dans le cadre de ce mémoire, elle sera employée afin de détecter des états de conscience.

Bien que les méthodes de neuro-imagerie fonctionnelle ne sont développées que depuis une vingtaine d'années, elles reposent sur des concepts bien plus anciens. En effet, l'idée de déduire l'activité cérébrale en mesurant les variations du flux sanguin n'est pas nouvelle. En 1890, un ouvrage de William James, *The Principles of Psychology*, est publié et nous pouvons y trouver le compte rendu d'une expérience réalisée par le physiologiste italien Angelo Mosso. Durant cette expérience, il demanda à un patient victime d'un trauma crânien d'effectuer une tâche de calcul mental et il constata que le débit sanguin et les pulsations cérébrales de celui-ci augmentaient. Il observa donc une relation entre le débit sanguin cérébral et l'activité neuronale chez le patient et cette découverte a été la première à suggérer que la mesure du flux sanguin cérébral pouvait être un moyen d'évaluer la conscience humaine. Une représentation de cette expérience est disponible à la Figure 1.2. Vers la fin du 19ème siècle, à l'Université de Cambridge, Charles S. Roy et Charles S. Sherrington ont fourni les premières preuves d'une relation entre activité neuronale et flux sanguin cérébral. En effectuant des tests sur des animaux anesthésiés, ils ont montré que le volume et le débit sanguin peuvent varier localement dans le cerveau. Cependant, il a fallu attendre l'année 1948 pour que Seymour Kety et Carl Schmidt confirment que des variations locales de l'activité neuronale étaient responsables de ces changements [9].

L'imagerie par résonance magnétique (IRM) repose sur le principe de la résonance magnétique nucléaire (RMN) découvert en 1938 par Isidor Isaac Rabi, qui reçut le prix Nobel de Physique en 1944 pour cette découverte. Au départ, la RMN était utilisée en spectroscopie qui est une technique permettant d'obtenir des informations sur la composition chimique des substances. Ensuite, grâce aux progrès fulgurants de l'informatique et de l'électronique entre les années 1970 et 1980, il est devenu possible d'appliquer cette technique à un patient. L'IRM permet donc d'obtenir des vues en deux ou en trois dimensions de l'intérieur d'une partie corps et est très utilisée pour l'observation du cerveau [18].

Le développement de l'IRMf dans les années 1990 est généralement attribué à Seiji Ogawa et Ken Kwong, et cette méthode est une application de l'IRM. En effet, l'IRMf est une adaptation de l'IRM pour mesurer les variations d'oxygénation du sang, reflet indirect de l'activité neuronale. L'IRMf se base sur le fait que l'activité neuronale augmente en réaction à la réalisation d'une tâche psychomotrice, ce qui nécessite d'accroître l'apport en énergie et en oxygène. La vascularisation cérébrale répond alors localement en augmentant le flux sanguin vers les régions d'activité neuronale accrue. Ce phénomène est appelé couplage neurovasculaire et entraîne des modifications hémodynamiques. En effet, comme l'oxygénation du sang varie en fonction des niveaux d'activité neuronale dans les zones du cerveau activées par la tâche psychomotrice, nous pouvons observer un changement concernant l'hémoglobine, la protéine des globules rouges qui transporte l'oxygène dans le sang. Cette protéine est appelée l'oxyhémoglobine lorsqu'elle est chargée d'oxygène et désoxyhémoglobine dans le cas contraire [11]. Dès lors, lorsqu'une personne effectue une tâche psychomotrice, l'apport en oxygène augmente dans les régions neuronales activées et cela entraîne une modification des concentrations d'oxyhémoglobine et de désoxyhémoglobine, en

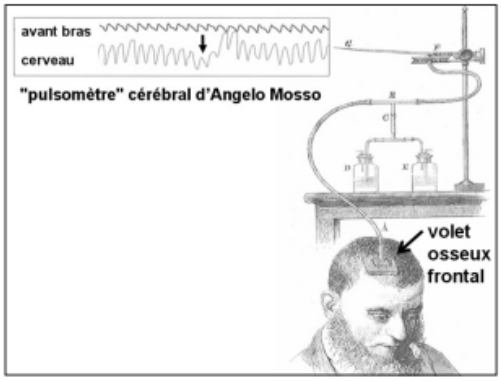

FIGURE 1.2 – Première expérience de "neuro-imagerie" effectuée par Angelo Mosso au 19ème siècle. Source : [13].

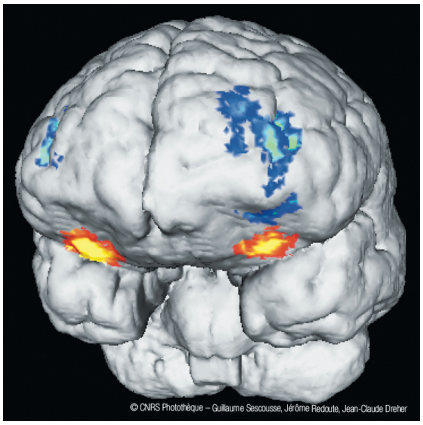

FIGURE 1.3 – Image du volume cérébral dans sa totalité obtenue par IRMf, © CNRS. Source : [18].

défaveur de la désoxyhémoglobine. Sachant que l'oxyhémoglobine et que la désoxyhémoglobine ne possèdent pas les mêmes comportements dans un champ magnétique, cette modification des concentrations va être détectée par l'IRMf et est appelée le signal BOLD (*Blood Oxygenated Level Dependent*). Le couplage neurovasculaire prend environ cinq à six secondes pour atteindre son maximum, mais des images du volume cérébral dans sa totalité (Figure 1.3) peuvent être obtenues en environ deux secondes seulement. De ce fait, l'IRMf permet de suivre l'évolution et d'enregistrer le signal BOLD en temps réel lorsque qu'une tâche cognitive est effectuée par un patient [13]. Depuis que les premières images d'un cerveau en fonctionnement ont été obtenues en 1992 grâce à l'IRMf, elle s'est imposée comme la technique de référence à travers le monde pour étudier le fonctionnement cérébral humain. Cette méthode est largement utilisée car elle offre d'indéniables avantages. Premièrement, elle permet d'étudier le fonctionnement neuronal avec une résolution temporelle de l'ordre de la seconde mais aussi une résolution spatiale de l'ordre du millimètre. Deuxièmement, elle est inoffensive pour le patient. En effet, alors que dans le passé une injection d'un traceur pouvant s'avérer toxique était requise, aujourd'hui, l'IRMf ne nécessite plus aucune injection d'un quelconque traceur radioactif. Cela implique que cette technique n'a aucune limite de répétition car elle est non-invasive et ne génère pas de radiation contrairement à d'autres techniques d'imagerie médicale. Par exemple, dans la tomographie par émission de positons, des images en coupe peuvent être obtenues à partir d'émissions produites par les positons issus de traceurs radioactifs injectés au préalable. Ces traceurs sont inoffensifs aux doses où ils sont administrés, mais il est tout de même interdit de faire des expériences à répétition sur un même sujet à cause de l'effet cumulatif des irradiations [19]. Troisièmement, l'IRMf est facile à utiliser pour l'expérimentateur.

1.3 Données issues de l'IRMf

Dans ce mémoire, nous allons travailler sur des données réelles issues de l'imagerie par résonance fonctionnelle et récoltées dans le cadre d'une étude par l'équipe du Docteur Steven Laureys du groupe *Coma Science Group* de l'Université de Liège. Le fait que ces données aient été acquises plusieurs années avant la réalisation de ce mémoire explique que certaines précisions n'ont malheureusement pas pu être obtenues quant aux opérations précises qui ont été effectuées à

l'époque.

Afin d'obtenir ces données, un certain nombre de patients se sont vus administrer un sédatif dans le but de les plonger dans un état végétatif. Différents signaux BOLD ont ensuite été récoltés dans certaines parties du cerveau appartenant au Réseau de Mode par Défaut (*Default Mode Network*, DMN), via la technique d'imagerie par résonance fonctionnelle expliquée précédemment. En effet, le DMN est un système de zones cérébrales connectées qui présentent une activité accrue lorsqu'une personne est au repos, ce qui inclut l'état végétatif. Les signaux BOLD ont dès lors été récoltés dans 28 régions de ce DMN à des moments différents afin de détecter quatre états de conscience distincts :

1. **Conscient** : Les signaux BOLD ont été récoltés avant l'administration du sédatif, c'est-à-dire lorsque les patients étaient encore conscients.
2. **Moyennement endormi** : Les signaux BOLD ont été mesurés lorsque le sédatif venait d'être administré aux patients et que, par conséquent, son effet était faible. Cet état correspond à la simulation d'un coma léger.
3. **Profondément endormi** : Les signaux BOLD ont été enregistrés lorsque le sédatif était à son efficacité maximale, dans le but de simuler un coma.
4. **En phase de réveil** : Les signaux BOLD ont été récoltés lorsque les effets du sédatif commençaient à s'estomper, ce qui correspond à une phase de réveil post-coma.

Il faut noter que ces données ont été enregistrées toutes les 2,46 secondes durant 197 ou 347 points de temps en fonction du patient auquel le sédatif a été administré.

D'un point de vue mathématique, nous pouvons visualiser les données récoltées comme de grandes matrices qui prennent la forme suivante

$$\begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,n_i} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,n_i} \\ \vdots & \vdots & \ddots & \vdots \\ x_{28,1} & x_{28,2} & \cdots & x_{28,n_i} \end{pmatrix} \quad \downarrow \text{ régions du DMN,}$$

$$\longrightarrow$$

Nombre de points de temps

où les $x_{i,j}$ représentent un signal BOLD à un temps $i \in \{1, ..., n_i\}$, pour une région du DMN $j \in \{1, ..., 28\}$, avec $n_i \in \{197, 347\}$ en fonction du patient considéré. Nous avons une matrice de ce type pour chaque état de conscience de chaque patient.

De plus, l'équipe du groupe *Coma* nous a fourni trois jeux de données correspondant chacun à un pré-traitement des données différent : *reduced*, *reduced GS* et *reduced GS filtered*. Le premier jeu de données, nommé *reduced*, comporte les données brutes, telles quelles ont été récoltées sur 17 patients par la technique d'imagerie par résonance magnétique fonctionnelle. Ce jeu de données ne concerne que 17 patients car les données du quatrième patient ne nous ont pas été fournies. Le deuxième jeu de données, nommé *reduced GS*, a été obtenu en appliquant un pré-traitement aux données récoltées sur 18 patients. Celui-ci consiste à enlever le signal global, c'est-à-dire les variations globales des signaux BOLD. Quant au troisième jeu de données, nommé *reduced GS filtered*, il est composé des signaux BOLD récoltés sur 18 patients, mais l'équipe leur a appliqué un pré-traitement supplémentaire qui supprime toutes les fréquences en dehors d'un certain intervalle. Cela permet d'éliminer les fréquences considérées comme du bruit, comme les

Forme des données	**Nombre de patients**	**Nombre de points de temps en fonction des patients**
Reduced	17	197 pour les 11 premiers et 347 pour les 6 derniers
Reduced GS	18	197 pour les 12 premiers et 347 pour les 6 derniers
Reduced GS filtered	18	197 pour les 12 premiers et 347 pour les 6 derniers

TABLE 1.1 – Récapitulatif de certaines informations concernant les jeux de données fournis par le *Coma Science Group* de l'Université de Liège.

battements du coeur par exemple.

Afin de mieux nous y retrouver dans les différentes données disponibles pour la partie détection des états de conscience dans la suite de ce mémoire, un tableau récapitulatif du nombre de patients ainsi que du nombre de points de temps considérés pour chaque forme de données récoltées par l'équipe du groupe *Coma Science Group* est présent sur le Tableau 1.1.

Dans le but de nous familiariser avec ces données, nous avons affiché les signaux BOLD du premier patient pour chaque état de conscience et pour chaque pré-traitement sur la Figure 1.4, la Figure 1.5 et la Figure 1.6. Pour chacune de ces figures, chaque courbe correspond à un signal BOLD dans l'une des 28 régions sélectionnées dans le DMN. Si nous comparons quatre figures issues d'un même jeu de données, nous ne pouvons pas observer de distinction notable entre les différents état de conscience. En effet, les signaux BOLD ont globalement la même allure même si nous pouvons noter quelques différences d'échelle entre les graphiques. Par exemple, pour la forme *reduced GS*, les valeurs des signaux BOLD correspondant à l'état "conscient" sont comprises entre -40 et 80 alors que celles correspondant à l'état "moyennement endormi" varient entre -20 et 25. Cela s'explique par le fait que nous pouvons observer des pics du signal BOLD aux alentours des points de temps 0 et 170 dans le graphique correspondant à l'état "conscient". Si nous retirons ces valeurs extrêmes, nous pouvons observer que les valeurs les signaux BOLD sont généralement comprises entre -30 et 30. Cependant, si nous comparons le graphique d'un même état de conscience dans les trois types de jeux de données fournies par l'équipe du groupe *Coma*, nous pouvons observer que les valeurs prises par les signaux BOLD sont fort distinctes. Pour le type *reduced*, nous constatons sur le premier graphique de la Figure 1.4 que les valeurs prises par les signaux BOLD varient entre 450 et 850. Par contre, ce n'est plus du tout le cas si nous regardons les graphiques correspondant à cet état, l'état "conscient", pour les autres types de données. En effet, pour le type *reduced GS* (Figure 1.5), ces valeurs varient entre -40 et 80 tandis que pour le type *reduced GS filtered* (Figure 1.6), elles sont comprises entre -1 et 1. Cela se justifie par le fait que, comme expliqué précédemment, les données brutes ont subi différents pré-traitements pour obtenir les trois formes de données. Par exemple, pour le type *reduced GS filtered*, il n'est pas surprenant de voir que les valeurs varient dans un intervalle beaucoup plus restreint car le bruit dans les signaux BOLD a été filtré.

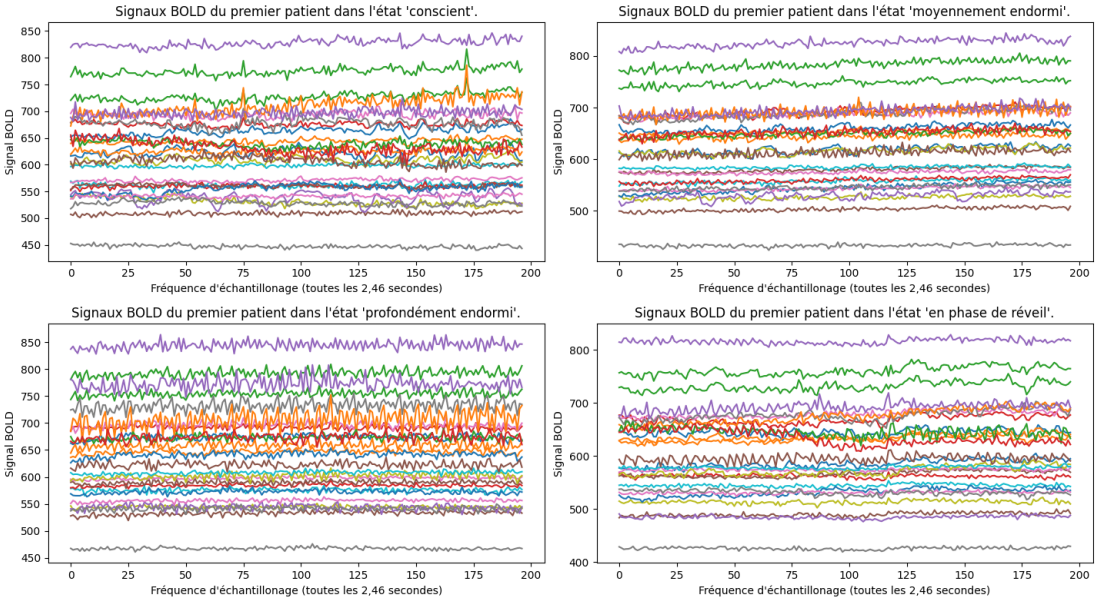

FIGURE 1.4 – Représentation des signaux BOLD issus d'une IRMf pour chacun des états de conscience du premier patient. Ces signaux sont ceux du type *reduced*.

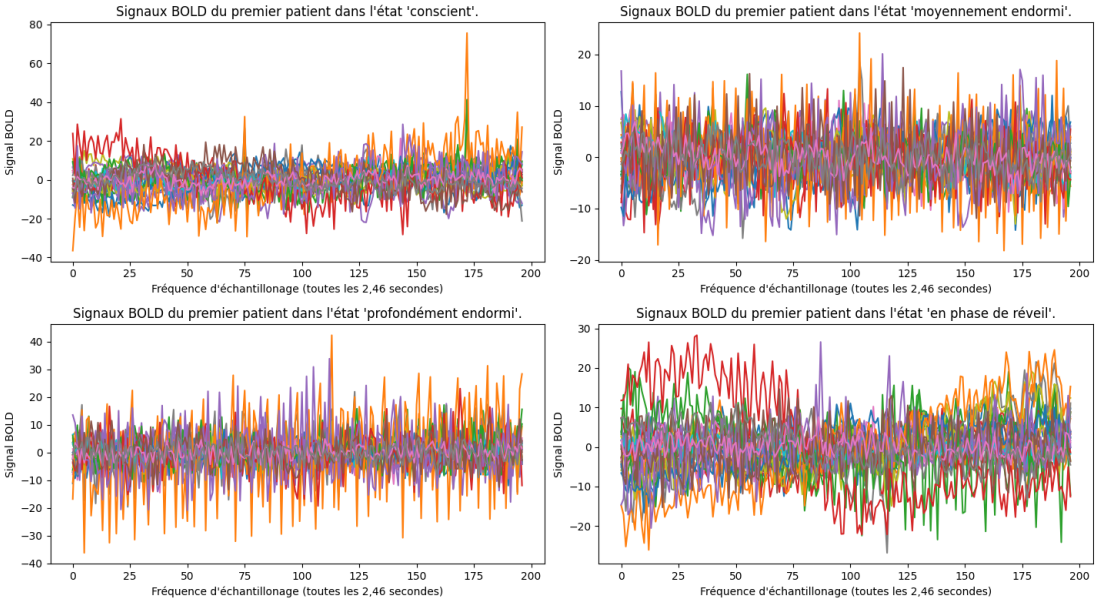

FIGURE 1.5 – Représentation des signaux BOLD issus d'une IRMf pour chacun des états de conscience du premier patient. Ces signaux sont ceux du type *reduced GS*.

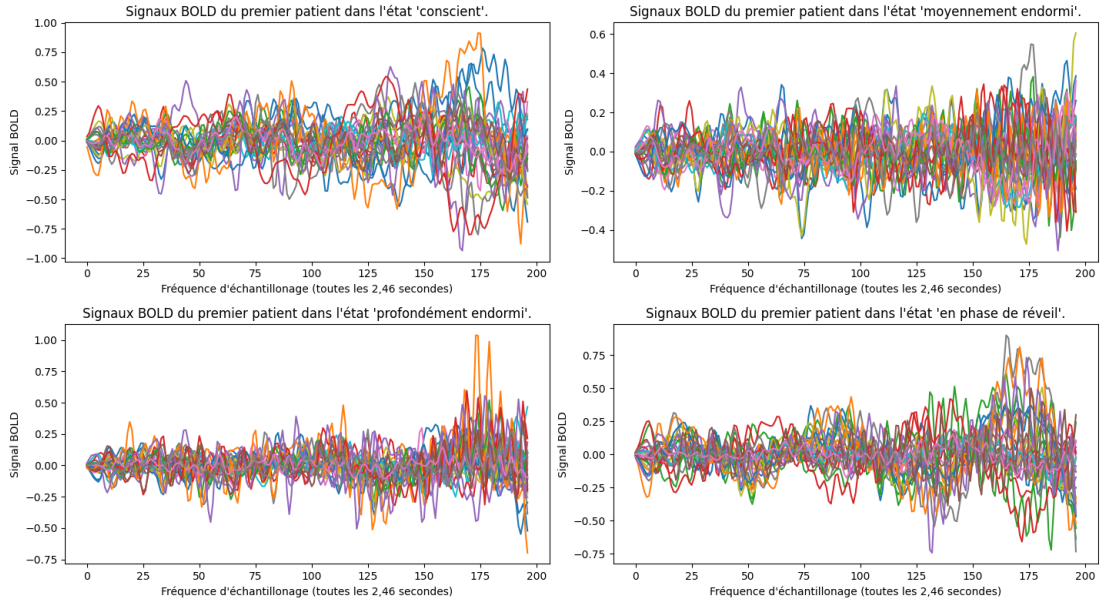

FIGURE 1.6 – Représentation des signaux BOLD issus d'une IRMf pour chacun des états de conscience du premier patient. Ces signaux sont ceux du type *reduced GS filtered*.

1.4 Conclusion

Dans la première partie de ce chapitre, nous avons expliqué l'enjeu crucial que représente l'évaluation de l'état de conscience d'une personne n'étant pas dans un état physiologique normal. En effet, le corps médical doit pouvoir déterminer de manière efficace et fiable cet état de conscience afin de pouvoir prendre les meilleures décisions concernant les soins à fournir au patient.

Ensuite, nous avons détaillé l'évolution des techniques de neuro-imagerie jusqu'à l'imagerie par résonance magnétique fonctionnelle qui est celle avec laquelle les données utilisées dans ce mémoire ont été obtenues. Ces données concernent le signal BOLD (*Blood Oxygenated Level Dependent*) qui varie lorsqu'une personne effectue une tâche psychomotrice car différentes zones de son cerveau sont alors activées.

Pour terminer, nous avons donné des précisions sur les données réelles que nous allons utiliser afin d'atteindre l'objectif de ce mémoire, c'est-à-dire détecter les différents états de conscience qui sont : "conscient", "moyennement endormi", "profondément endormi" et "en phase de réveil".

Dans les prochains chapitres, nous allons détailler la méthodologie utilisée pour détecter ces états de conscience chez les différents patients dont nous avons récolté les données. Plus précisément, cette méthodologie sera divisée en deux parties : l'extraction d'informations spectrales grâce à l'opérateur de Koopman et la classification de ces informations grâce à un algorithme de classification supervisée.

Chapitre 2

Approche spectrale pour les séries temporelles

Ce chapitre est dédié à l'opérateur de Koopman. Dans ce mémoire, cet opérateur sera utilisé pour extraire certaines informations des signaux BOLD présentés dans le chapitre précédent et ces informations serviront ensuite à détecter les états de conscience de différents patients. Dans un premier temps, nous introduisons l'opérateur de Koopman, un opérateur linéaire de dimension infinie, qui permet de décrire un système non-linéaire grâce à une décomposition appelée la décomposition en modes de Koopman. Ensuite, nous présentons deux méthodes, la décomposition en modes dynamiques exacte et la décomposition en modes dynamiques de Hankel, dont les algorithmes permettent d'approximer numériquement l'analyse spectrale de l'opérateur de Koopman. Pour terminer, nous effectuons une brève analyse visuelle du comportement de ces éléments spectraux afin de déterminer si elle est suffisante pour distinguer les différents états de conscience.

2.1 Opérateur de Koopman

Cette section reprend les principales informations concernant l'opérateur de Koopman, un opérateur dont le formalisme trouve son origine dans les travaux de Bernard Koopman en 1931 [23]. L'année suivante, John von Neumann et Bernard Koopman vont introduire la notion de spectre d'un système dynamique, c'est-à-dire le spectre de l'opérateur de Koopman associé. Ensuite, il faudra attendre plusieurs décennies après les travaux de von Neumann et Koopman pour que les propriétés spectrales de l'opérateur de Koopman soient analysées plus en détail et pour que les modes de Koopman soient étudiés [28]. Notons que dans cette section, nous n'aborderons que le cas discret car c'est celui que nous utiliserons dans la suite de ce mémoire. Cela signifie que nous n'allons considérer que des valeurs régulièrement espacées du temps, c'est-à-dire des valeurs t du temps qui satisfont l'égalité $t = \tau_0 + n \cdot \Delta t$, où τ_0 est un temps de référence, n est un naturel et Δt est l'intervalle de temps entre chaque valeur.

Nous commençons par énoncer quelques concepts liés à l'opérateur de Koopman. Pour ce faire, nous allons nous baser sur les références [5], [7], [27] et [34]. Considérons une application non-linéaire

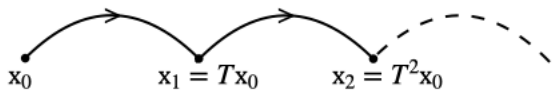

FIGURE 2.1 – Illustration de la trajectoire d'un système dynamique discret. Source : [27].

$$\begin{aligned} T: \quad X &\to X \\ \boldsymbol{x} &\mapsto T(\boldsymbol{x}) \ , \end{aligned}$$

où X est un espace donné de dimension finie et $\boldsymbol{x} \in X \subset \mathbb{R}^d$, $d \in \mathbb{N}_0$, est un vecteur d'état. Notons que pour le reste de cette section, les objets représentant des vecteurs seront notés en gras. La dynamique en temps discret décrite par cette application est donnée par la relation

$$\boldsymbol{x}_t \ = \ T(\boldsymbol{x}_{t-1}), \ \forall t \in \mathbb{N}_0.$$

Comme le domaine de l'application T est égal à son image, l'itération de T sur une condition initiale $\boldsymbol{x}_0 \in X$ donnée est bien définie. La trajectoire est illustrée à la Figure 2.1 et le t-ième élément de celle-ci est alors défini comme

$$\boldsymbol{x}_t \ = \ T(\boldsymbol{x}_{t-1}) \ = \ T \circ T(\boldsymbol{x}_{t-2}) \ = \ldots \ = \ T^{\circ t}(\boldsymbol{x}_0).$$

Nous introduisons ensuite un espace fonctionnel $\mathcal{F}$ contenant des observables

$$\begin{aligned} g: \quad X &\to \mathbb{C} \\ \boldsymbol{x} &\mapsto g(\boldsymbol{x}) \ , \end{aligned}$$

qui sont des fonctions à valeurs scalaires. Maintenant que toutes ces notions ont bien été mises en place, nous pouvons introduire la définition de l'opérateur de Koopman.

Définition 2.1.1. Soit $T : X \to X$ une application non-linéaire associée à une dynamique discrète $\boldsymbol{x}_t = T(\boldsymbol{x}_{t-1})$, $\forall t \in \mathbb{N}_0$, et soit $\mathcal{F}$ un espace fonctionnel. L'**opérateur de Koopman** (ou opérateur de composition), noté U, est défini par

$$\begin{aligned} U: \quad \mathcal{F} &\to \mathcal{F} \\ g &\mapsto Ug \ , \end{aligned}$$

tel que $\forall \boldsymbol{x} \in X$, $Ug(\boldsymbol{x}) \ \triangleq \ (g \circ T)(\boldsymbol{x})$.

De plus, voici une propriété cruciale de l'opérateur de Koopman.

Propriété 2.1.1. L'opérateur de Koopman est un opérateur linéaire. Dès lors, $\forall\ a, b \in \mathbb{R}$ et $\forall\ f, g \in \mathcal{F}$, nous avons que

$$U(af + bg) \ = \ a(Uf) + b(Ug).$$

Démonstration. Soient $a, b \in \mathbb{R}$ et $f, g \in \mathcal{F}$. En appliquant la définition de l'opérateur de Koopman et par linéarité de la composition de fonctions, nous obtenons

$$\begin{aligned} U(af+bg) &= (af+bg)\circ T \\ &= a(f\circ T)+b(g\circ T). \end{aligned}$$

Nous pouvons alors à nouveau appliquer la définition de l'opérateur de Koopman, de sorte que

$$a(f\circ T)+b(g\circ T) \;=\; a(Uf)+b(Ug).$$

Cette dernière égalité prouve la linéarité de l'opérateur de Koopman car nous obtenons le résultat attendu. □

L'opérateur de Koopman est donc un opérateur linéaire décrivant un système dynamique associé à une application non-linéaire T et il agit sur des observables. Autrement dit, cet opérateur permet d'étudier l'évolution du vecteur d'état $\boldsymbol{x}$ au travers d'une observable dans l'espace fonctionnel $\mathcal{F}$ plutôt que de l'étudier directement dans l'espace d'état X. L'inconvénient de cet opérateur est qu'il est de dimension infinie, mais il est toutefois possible de contourner ce problème en l'approximant par une matrice de dimension finie.

De plus, il est possible de simplifier la dynamique par une décomposition de l'opérateur de Koopman en valeurs propres et fonctions propres qui évoluent linéairement avec le flux de la dynamique. Cette décomposition en éléments propres est d'ailleurs l'une des motivations principales pour utiliser cet opérateur.

Définition 2.1.2. Une observable non-nulle ϕ_μ dans un espace fonctionnel $\mathcal{F}$ est une **fonction propre** de l'opérateur de Koopman U si il existe un nombre complexe μ appelé **valeur propre** de U associée à ϕ_μ tel que

$$U\phi_\mu \;=\; \mu\phi_\mu. \tag{2.1}$$

Dans ce cas, μ appartient au **spectre** de l'opérateur de Koopman U, noté $\sigma(U)$.

Remarquons que s'il vérifie la relation (2.1), le couple (ϕ_μ, μ) est appelé paire fonction propre - valeur propre de l'opérateur de Koopman U. Nous allons maintenant passer à une propriété intéressante concernant les fonctions propres et les valeurs propres de l'opérateur de Koopman.

Propriété 2.1.2. Soient μ_1, μ_2 deux valeurs propres complexes associées à $\phi_{\mu_1}, \phi_{\mu_2}$ deux observables qui sont deux fonctions propres de opérateur de Koopman U. Alors $\phi_{\mu_1}\cdot\phi_{\mu_2}$, pour autant que ce produit appartienne à l'espace $\mathcal{F}$, est également une fonction propre de U dont la valeur propre associée est $\mu_1\cdot\mu_2$.

Démonstration. Soient $\phi_{\mu_1}, \phi_{\mu_2} \in \mathcal{F}$ et $\mu_1, \mu_2 \in \mathbb{C}$ tels que

$$U\phi_{\mu_1} \;=\; \mu_1\phi_{\mu_1} \;\text{ et }\; U\phi_{\mu_2} \;=\; \mu_2\phi_{\mu_2}.$$

Alors par la définition de l'opérateur de Koopman et par la propriété de la composition de fonctions, il résulte que

$$\begin{aligned} U(\phi_{\mu_1}\cdot\phi_{\mu_2}) &= (\phi_{\mu_1}\cdot\phi_{\mu_2})\circ T \\ &= (\phi_{\mu_1}\circ T)\cdot(\phi_{\mu_2}\circ T) \\ &= U\phi_{\mu_1}\cdot U\phi_{\mu_2} \\ &= \mu_1\phi_{\mu_1}\cdot\mu_2\phi_{\mu_2} \\ &= (\mu_1\cdot\mu_2)(\phi_{\mu_1}\cdot\phi_{\mu_2}), \end{aligned}$$

ce qui prouve que $\mu_1 \cdot \mu_2$ est la valeur propre associée à la fonction propre $\phi_{\mu_1} \cdot \phi_{\mu_2}$ de U. $\square$

Notons que cette propriété implique que l'opérateur de Koopman possède une infinité de paires fonction propre - valeur propre.

De plus, pour une paire fonction propre - valeur propre (ϕ_μ, μ) de U, nous savons que

$$\phi_\mu(\boldsymbol{x}_t) \;=\; \phi_\mu(T(\boldsymbol{x}_{t-1})) \;=\; \phi_\mu \circ T(\boldsymbol{x}_{t-1}) \;=\; U\phi_\mu(\boldsymbol{x}_{t-1}) \;=\; \mu\phi_\mu(\boldsymbol{x}_{t-1}), \tag{2.2}$$

où $\boldsymbol{x}_t \in X$, $\forall t \in \mathbb{N}_0$. Cela implique que pour une condition initiale $\boldsymbol{x}_0 \in X$ donnée, les égalités

$$\phi_\mu(\boldsymbol{x}_t) \;=\; \mu\phi_\mu(\boldsymbol{x}_{t-1}) \;=\; \mu^2\phi_\mu(\boldsymbol{x}_{t-2}) \;=\; \cdots \;=\; \mu^t\phi_\mu(\boldsymbol{x}_0) \tag{2.3}$$

sont vérifiées.

Jusqu'à présent, nous avons considéré des mesures scalaires d'un système dynamique mais en pratique, les données mesurées sur un tel système ne proviennent pas souvent d'une seule observable. Par exemple, dans notre cas, les signaux BOLD sont enregistrés dans différentes zones du cerveau afin de déterminer l'état de conscience d'une personne. Ces mesures peuvent alors être organisées en un vecteur $\mathbf{g} : X \to \mathbb{R}^m$ défini par

$$\mathbf{g}(\boldsymbol{x}) \;=\; \begin{pmatrix} g_1(\boldsymbol{x}) \\ g_2(\boldsymbol{x}) \\ \vdots \\ g_m(\boldsymbol{x}) \end{pmatrix},$$

où chaque g_i est une observable, c'est-à-dire $g_i \in \mathcal{F},\ i = 1, \cdots, m$.

Si chacune des mesures individuelles de $\mathbf{g}$ est dans le sous-espace engendré par les fonctions propres $\phi_{\mu_j},\ j = 1, 2, \cdots$, de l'opérateur de Koopman U, les observables g_i peuvent se réécrire comme

$$g_i(\boldsymbol{x}) = \sum_{j=1}^{\infty} v_{ij}\phi_{\mu_j}(\boldsymbol{x}),$$

où les v_{ij} sont les coefficients complexes de la décomposition. Ainsi, le vecteur $\mathbf{g}$ des observables peut être développé de manière similaire, c'est-à-dire

$$\mathbf{g}(\boldsymbol{x}) = \sum_{j=1}^{\infty} \phi_{\mu_j}(\boldsymbol{x})\mathbf{v}_j, \tag{2.4}$$

avec $\mathbf{v}_j$ un vecteur de nombres complexes de dimension m. De plus, en utilisant la relation (2.3), l'évolution temporelle $\mathbf{g}(\boldsymbol{x}_t)$ peut être exprimée par

$$\mathbf{g}(\boldsymbol{x}_t) = \sum_{j=1}^{\infty} \mu_j^t\phi_{\mu_j}(\boldsymbol{x}_0)\mathbf{v}_j.$$

Cette expression est appelée la décomposition en modes de Koopman (*Koopman Mode Decomposition*, KMD), les $\mathbf{v}_j$ étant les modes de Koopman associés à la fonction propre ϕ_{μ_j} et au vecteur des observables $\mathbf{g}$. Par ailleurs, le triplet $(\mu_j, \phi_{\mu_j}, \mathbf{v}_j),\ j = 1, 2, \cdots$, est appelé le triplet de Koopman. Un résumé reprenant le processus appliqué pour obtenir la décomposition en modes

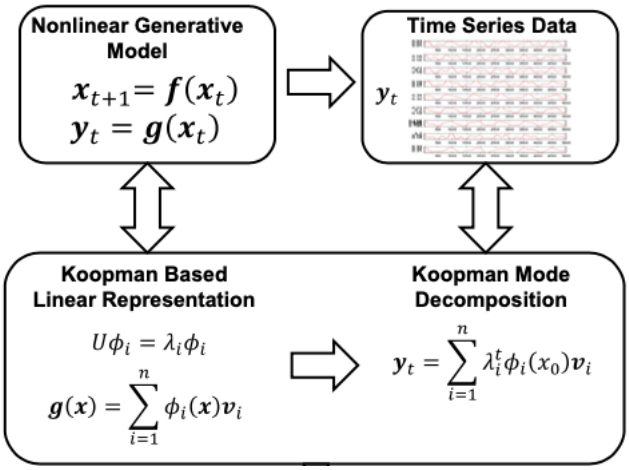

FIGURE 2.2 – Modèles de représentation des données issues de séries temporelles basés sur la théorie de l'opérateur de Koopman. Source : [34].

de Koopman est disponible à la Figure 2.2.

Maintenant que l'opérateur de Koopman a été introduit et que la décomposition en modes de Koopman a été présentée, nous allons développer une méthode numérique appelée la décomposition en modes dynamiques permettant d'approximer les propriétés spectrales de cet opérateur.

2.2 Décomposition en modes dynamiques

La décomposition en modes dynamiques (DMD) est un algorithme introduit en 2008 par Peter Schmid dans la communauté de la dynamique des fluides [33]. Les écoulements de fluides sont régis par des équations de dimension infinie, mais ils présentent souvent un comportement de faible dimension et nous pouvons identifier leur dynamique. Pour cela, les écoulements sont analysés à l'aide de techniques de décomposition modale dont fait partie la décomposition en modes dynamiques. Cette dernière permet de réduire efficacement la dimension des systèmes tout en fournissant une décomposition modale ainsi que l'évolution de ces modes dans le temps. Elle fournit donc des informations sur la dynamique d'un écoulement et elle est applicable même lorsque cette dynamique est non-linéaire. Pour toutes ces raisons, la décomposition en modes dynamiques est couramment utilisée dans la communauté des fluides depuis sa présentation. Peu après le développement de l'algorithme de décomposition en modes dynamiques original, Clarence W. Rowley, Igor Mezic et leurs collaborateurs ont établi un lien important entre la décomposition en modes dynamiques et la décomposition en modes de Koopman [32]. Pour rappel, l'opérateur de Koopman est un opérateur linéaire de dimension infinie dont les valeurs propres et les modes de Koopman obtenus lors de la décomposition en modes de Koopman décrivent l'évolution des observables d'un système dynamique. Nous allons donc utiliser la décomposition en modes dynamiques dans ce mémoire car elle peut être considérée comme une approximation numérique de l'analyse spectrale de Koopman.

Plusieurs algorithmes ont été proposés pour la décomposition en modes dynamiques mais dans ce mémoire, nous allons utiliser la décomposition en modes dynamiques exacte et la décom-

position en modes dynamiques de Hankel.

2.2.1 Décomposition en modes dynamiques exacte

Dans cette section, nous allons d'abord introduire la décomposition en modes dynamiques exacte et présenter son algorithme. Ensuite, nous allons justifier le choix de cet algorithme en démontrant le lien existant entre la décomposition en modes dynamiques exacte et la décomposition en modes de Koopman. Pour cela, nous nous baserons sur les références [5], [34] et [35].

L'algorithme de décomposition en modes dynamiques exacte, noté DMD exacte, est basé sur la décomposition en valeurs singulières. Alors que l'algorithme original de décomposition en modes dynamiques considérait une série temporelle séquentielle, la nouvelle définition présentée dans la référence [35] met l'accent sur les données collectées comme un ensemble de paires $\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^{m}$, où chaque $\boldsymbol{x}_k, \boldsymbol{y}_k$ est un vecteur réel de dimension n. La première étape consiste donc à réunir un ensemble de paires d'instantanés, c'est-à-dire de sauvegardes de l'état du système au fur et à mesure qu'il évolue dans le temps. Ces données peuvent ensuite être stockées dans deux matrices de dimension $n \times m$

$$X \triangleq \begin{bmatrix} | & & | \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_m \\ | & & | \end{bmatrix} \quad \text{et} \quad Y \triangleq \begin{bmatrix} | & & | \\ \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_m \\ | & & | \end{bmatrix}. \tag{2.5}$$

Définition 2.2.1. Pour un ensemble de données stockées de la même manière que dans l'équation (2.5), la **décomposition en modes dynamiques exacte** de la paire (X, Y) est donnée par la décomposition en valeurs propres et vecteurs propres de la matrice A définie telle que

$$A \triangleq YX^{+},$$

où X^{+} est la matrice pseudo-inverse de la matrice X. Autrement dit, les modes et les valeurs propres de la décomposition en modes dynamiques exacte sont les vecteurs propres et les valeurs propres de la matrice A.

Cependant, même si cette définition fournissant une approximation par un opérateur linéaire A est simple et brève, calculer la décomposition de A n'est pas le moyen le plus efficace d'effectuer la décomposition en modes dynamiques. C'est pourquoi un algorithme qui permet de calculer les modes et les valeurs propres de la décomposition en modes dynamiques sans représentation explicite ni manipulation directe de A a été proposé. En effet, suite à une décomposition en valeurs singulières de la matrice X, cet algorithme définit une matrice réduite $\tilde{A}$, appelée matrice de Koopman, qui a les mêmes valeurs propres non-nulles que la matrice A. Ainsi, il suffit de calculer la matrice réduite $\tilde{A}$, sans jamais travailler avec la matrice A qui peut être de très grande dimension. Cet algorithme est détaillé dans l'Algorithme 1.

Nous allons maintenant montrer qu'il existe un lien entre la définition de la décomposition en modes dynamiques exacte et la théorie de l'opérateur de Koopman. Sans ce lien, l'utilisation de l'algorithme DMD exacte pour analyser les dynamiques non linéaires serait inappropriée. Tout d'abord, nous allons reprendre l'équation (2.2) ainsi que l'opérateur de Koopman. Grâce à la relation (2.1), nous obtenons

$$U\boldsymbol{g}(\boldsymbol{z}) = \sum_{i=1}^{\infty} \mu_j \phi_{\mu_j}(\boldsymbol{z}) \boldsymbol{v}_j. \tag{2.6}$$

Algorithme 1 DMD exacte. Source : [35].

Entrée: Paires de données $\{(\boldsymbol{x}_k, \boldsymbol{y}_k)\}_{k=1}^m$, avec $\boldsymbol{x}_k, \boldsymbol{y}_k \in \mathbb{R}^n$.
Sortie: Modes et valeurs propres de la décomposition en modes dynamiques.

Étape 1 : Stocker les paires de données $\{(\boldsymbol{x}_1, \boldsymbol{y}_1), \cdots, (\boldsymbol{x}_m, \boldsymbol{y}_m)\}$ dans deux matrices X et Y telles que

$$X \triangleq \begin{bmatrix} | & & | \\ \boldsymbol{x}_1 & \cdots & \boldsymbol{x}_m \\ | & & | \end{bmatrix} \quad \text{et} \quad Y \triangleq \begin{bmatrix} | & & | \\ \boldsymbol{y}_1 & \cdots & \boldsymbol{y}_m \\ | & & | \end{bmatrix}.$$

Étape 2 : Calculer la décomposition en valeurs singulières (SVD) réduite de X, à savoir $X = U\Sigma V^*$.

Étape 3 : Définir la matrice $\tilde{A} \triangleq U^* Y V \Sigma^{-1}$.

Étape 4 : Calculer les valeurs propres et les vecteurs propres de $\tilde{A}$ en écrivant $\tilde{A}\omega = \lambda\omega$. Chaque valeur propre non nulle λ est une valeur propre de la DMD.

Étape 5 : Le mode de la DMD correspondant à λ est donné par

$$\varphi \triangleq \frac{1}{\lambda} Y V \Sigma^{-1} \omega.$$

Ensuite, nous allons introduire une définition et deux théorèmes qui nous seront utiles pour la suite.

Théorème 2.2.1. Chaque paire (λ, φ) obtenue par l'Algorithme 1 est une paire valeur propre - vecteur propre de A. De plus, l'algorithme identifie toutes les valeurs propres non-nulles de A [35].

Définition 2.2.2. Deux matrices X et Y de dimension $n \times m$ sont **linéairement cohérentes** si

$$Xc = 0 \;\Rightarrow\; Yc = 0,$$

pour un vecteur réel c de dimension m. Autrement dit, X et Y sont linéairement cohérentes si le noyau de Y contient le noyau de X.

Théorème 2.2.2. Définissons $A = YX^+$ comme dans la **Définition 2.2.1**. Alors $Y = AX$ si et seulement si X et Y sont linéairement cohérentes [35].

Enfin, nous allons considérer un ensemble d'états initiaux arbitraires $\{\boldsymbol{z}_1, \cdots, \boldsymbol{z}_m\}$ et, pour l'observable g et l'application non-linéaire T fixés précédemment, nous définissons les paires d'instantanés comme

$$\boldsymbol{x}_k = g(\boldsymbol{z}_k) \quad \text{et} \quad \boldsymbol{y}_k = g \circ T(\boldsymbol{z}_k) = U g(\boldsymbol{z}_k), \; k = 1, \cdots, m. \tag{2.7}$$

Nous pouvons alors stocker ces données dans des matrices X et Y comme définies en (2.5). Nous appliquons ensuite l'Algorithme 1 à ces données afin de trouver des valeurs propres et des vecteurs propres qui, grâce au **Théorème 2.2.1**, vérifient

$$A\varphi_j = \lambda_j \varphi_j, \tag{2.8}$$

où $A = YX^+$. Si A possède un ensemble complet de vecteurs propres et si chacune des colonnes de X se trouve dans le sous-espace engendré par ces vecteurs propres, nous pouvons développer chaque $\boldsymbol{x}_k$ comme

$$\boldsymbol{x}_k = \sum_{j=1}^{n} c_{jk}\varphi_j, \tag{2.9}$$

où les c_{jk} sont les coefficients constants de la décomposition. De plus, pour des données linéairement cohérentes, nous savons que $A\boldsymbol{x}_k = \boldsymbol{y}_k$ par le **Théorème 2.2.2**. Par conséquent et par les relations (2.8) et (2.9),

$$\begin{aligned} \boldsymbol{y}_k &= A\boldsymbol{x}_k \\ &= \sum_{j=1}^{n} c_{jk}A\varphi_j \\ &= \sum_{j=1}^{n} \lambda_j c_{jk}\varphi_j. \end{aligned}$$

Sachant que $\boldsymbol{y}_k = Ug(\boldsymbol{z}_k)$ par la relation (2.7), et en comparant le résultat avec l'équation (2.6), nous pouvons conclure que dans le cas de matrices de données linéairement cohérentes, les modes (resp. les valeurs propres) de la décomposition en modes dynamiques correspondent aux modes (resp. aux valeurs propres) de Koopman. En effet, les constantes c_{jk} sont données par $c_{jk} = \phi_{\mu_j}(\boldsymbol{z}_k)$, c'est-à-dire qu'elles correspondent à une fonction propre évaluée en un point et multipliée par le mode de Koopman associé. Cependant, en pratique, les matrices de données ne sont pas linéairement cohérentes car elles ne sont pas générées par un système linéaire. Par conséquent, les valeurs propres et les modes obtenus ne correspondent qu'à des approximations des valeurs propres et des modes de Koopman.

Nous allons maintenant présenter la deuxième méthode de décomposition en modes dynamiques appelée la décomposition en modes dynamiques de Hankel.

2.2.2 Décomposition en modes dynamiques de Hankel

Dans cette section, nous allons d'abord introduire une matrice particulière appelée la matrice de Hankel. Ensuite, nous présenterons la décomposition en modes dynamiques dite de Hankel ainsi que son algorithme. Pour cela, nous nous baserons sur les références [30] et [1].

La matrice de Hankel est une matrice qui permet de stocker les différents instantanés d'une série temporelle sous une certaine forme. En effet, pour construire cette matrice, nous devons utiliser une méthode appelée *Time-Delay Embedding* qui consiste à intégrer à retardement des morceaux d'une série temporelle. Ainsi, la matrice de Hankel $\mathcal{H}$ d'une série temporelle composée des instantanés $(\boldsymbol{x}_1, ..., \boldsymbol{x}_m)$, $\boldsymbol{x}_i \in \mathbb{R}^n$ pour $i = 1, ..., m$, est donnée par

$$\mathcal{H} = \begin{pmatrix} \boldsymbol{x}_1 & \boldsymbol{x}_2 & \dots & \boldsymbol{x}_{m_c} \\ \boldsymbol{x}_2 & \boldsymbol{x}_3 & \dots & \boldsymbol{x}_{m_c+1} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x}_{m_r} & \boldsymbol{x}_{m_r+1} & \dots & \boldsymbol{x}_m \end{pmatrix}, \tag{2.10}$$

où $m_c + m_r - 1 = m$, $m_c, m_r \in \mathbb{N}_0$. La matrice $\mathcal{H}$ possède donc $n \times m_r$ lignes et m_c colonnes.

Ensuite, une décomposition en modes dynamiques est appliquée à la matrice $\mathcal{H}$. Cette décomposition en modes dynamiques est appelée la décomposition en modes dynamiques de Hankel ou Hankel-DMD et elle est basée sur la matrice compagnon. En effet, dans cette méthode, la matrice de Koopman notée $\tilde{C}$ prend la forme d'une matrice compagnon. Le pseudo-code de l'algorithme Hankel-DMD est détaille dans l'Algorithme 2.

Algorithme 2 Hankel-DMD. Source : [1].

Entrée: La matrice $\mathcal{H}$ définie dans l'équation (2.10).
Sortie: Modes et valeurs propres de la décomposition en modes dynamiques.

Étape 1 : Stocker les $m_c - 1$ premières colonnes de $\mathcal{H}$ dans une matrice X telle que

$$X \triangleq \begin{bmatrix} | & & | \\ \mathcal{H}_1 & \cdots & \mathcal{H}_{m_c-1} \\ | & & | \end{bmatrix}.$$

Étape 2 : Calculer la matrice compagnon

$$\tilde{C} = \begin{pmatrix} 0 & 0 & \dots & 0 & \tilde{c}_0 \\ 1 & 0 & \dots & 0 & \tilde{c}_1 \\ 0 & 1 & \dots & 0 & \tilde{c}_2 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & \tilde{c}_{m_c-1} \end{pmatrix}$$

avec

$$(\tilde{c}_0, \tilde{c}_1, \tilde{c}_2, ..., \tilde{c}_{m_c-1}) = X^{+}\mathcal{H}_{m_c} \ ,$$

où X^{+} est la matrice pseudo-inverse de la matrice X et $\mathcal{H}_{m_c}$ est la dernière colonne de la matrice de Hankel $\mathcal{H}$.

Étape 3 : Calculer les valeurs propres et les vecteurs propres de $\tilde{C}$ en écrivant $\tilde{C}\omega = \lambda\omega$. Chaque valeur propre non nulle λ est une valeur propre de la DMD. Le mode $\tilde{v}$ de la DMD correspondant à λ est donné par

$$\tilde{v} = X\omega.$$

Tout comme pour l'algorithme DMD exacte, les vecteurs propres et les valeurs propres obtenus grâce à la décomposition de la matrice $\tilde{C}$ sont une approximation des valeurs propres et vecteurs propres de l'opérateur de Koopman (pour plus de détails, voir [1]).

2.3 Analyse d'une approximation des valeurs propres de Koopman

Dans cette section, nous allons observer le comportement des valeurs propres de Koopman d'une série temporelle, approximées par une méthode de décomposition en modes dynamiques.

Pour ce faire, la première méthode de décomposition en modes dynamiques, la DMD exacte, sera utilisée. Comme les matrices contenant les signaux BOLD sont de taille 28×197 ou 28×347,

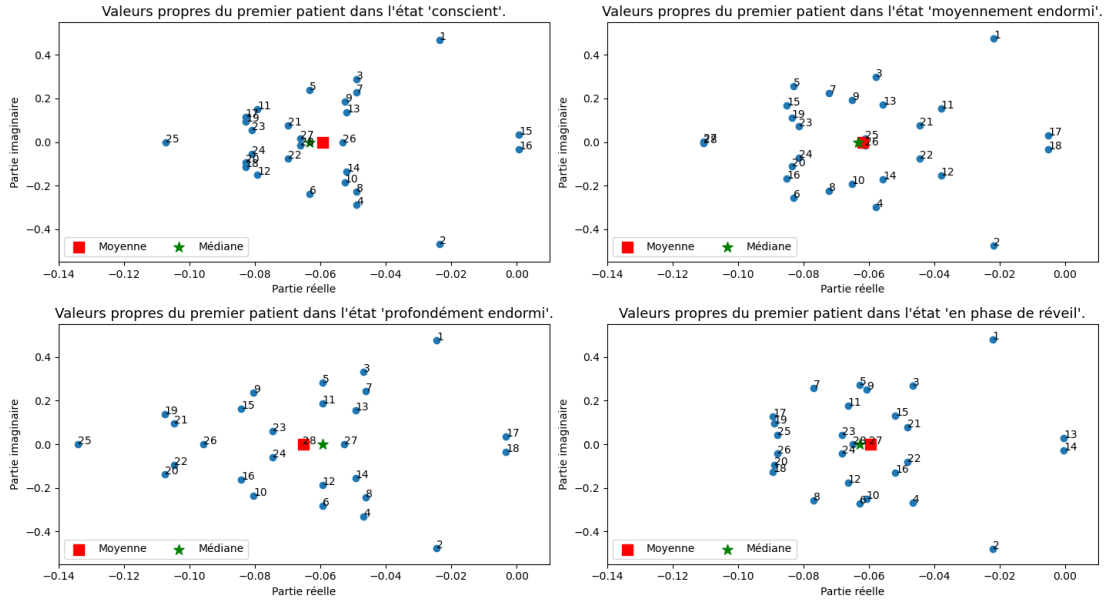

FIGURE 2.3 – Valeurs propres obtenues après avoir appliqué l'algorithme DMD exacte aux séries temporelles comprenant les signaux BOLD du premier patient. Ces signaux sont ceux du type *reduced GS filtered*.

nous obtenons 28 valeurs propres et 28 modes de Koopman appartenant à $\mathbb{C}^{28}$ à la sortie de l'algorithme DMD exacte. Dans cette analyse, nous allons nous concentrer sur les approximations des valeurs propres de Koopman. Durant la recherche, les valeurs propres de tous les états de conscience de tous les patients et pour tous les types de données ont été analysées, mais les conclusions étant similaires, nous n'allons présenter que celles des quatre états de conscience du premier patient pour le type de données *reduced GS filtered* dans ce mémoire. Ces valeurs propres sont affichées sur la Figure 2.3. Sur cette figure, nous pouvons d'abord observer que toutes les valeurs propres complexes obtenues grâce à l'algorithme DMD exacte ont une partie réelle comprise dans l'intervalle $[-0,14; 0,02]$ et une partie imaginaire comprise dans l'intervalle $[-0,55; 0,55]$. Nous avons affiché les valeurs propres des quatre états de conscience sur des graphiques ayant les mêmes axes pour pouvoir comparer leur comportement. Par conséquent, nous pouvons constater que les nuages de points des valeurs propres pour les quatre états de conscience ont des formes similaires. De plus, nous avons affiché la moyenne ainsi que la médiane de chaque nuage de points. Celles-ci sont a peu près placées au même endroit pour les quatre états de conscience, c'est-à-dire aux alentours du point dont les coordonnées sont $(0; -0,08)$.

Le constat est le même pour tous les autres patients quelque soit le type de données que nous considérons. Nous ne savons donc pas visuellement distinguer les états de conscience grâce aux informations spectrales fournies par les méthodes de décomposition en modes dynamiques. Par conséquent, nous devons employer d'autres méthodes permettant de déceler des différences non-visuelles entre les états de conscience. Il s'agit de méthodes de classification supervisée et elles seront introduites dans la chapitre suivant.

2.4 Conclusion

Dans la première partie de ce chapitre, nous avons introduit l'opérateur de Koopman et nous en avons donné quelques propriétés. Ensuite, nous avons présenté la décomposition en modes de Koopman qui est une décomposition spectrale de l'opérateur de Koopman.

Comme l'opérateur de Koopman est un opérateur linéaire mais de dimension infinie, nous avons ensuite détaillé deux algorithmes de décomposition en modes dynamiques qui permettent d'approximer numériquement l'analyse spectrale de l'opérateur de Koopman à partir d'une série temporelle. Ces deux algorithmes sont la décomposition en modes dynamiques exacte et la décomposition en modes dynamiques de Hankel.

Afin de déterminer si les éléments spectraux obtenus grâce aux méthodes de décomposition en modes dynamiques permettent de distinguer visuellement les différents états de conscience, nous avons analysé le comportement de ceux-ci et plus particulièrement celui des valeurs propres de Koopman approximées. La conclusion de cette analyse est que nous devons utiliser d'autres méthodes nous permettant de séparer efficacement les différents états de conscience.

Dans le prochain chapitre, la classification supervisée de données sera introduite. De plus, une méthode de classification supervisée appelée *Extremely Randomized Trees* sera présentée. Celle-ci permettra de classer les signaux BOLD sur bases des propriétés spectrales approximées.

Chapitre 3

Classification supervisée

Dans ce chapitre, la classification supervisée de données est évoquée et nous détaillons la procédure que nous utilisons pour classer les états de conscience. Pour cela, nous introduisons une méthode appelée *Extremely Randomized Trees* qui est la méthode principale utilisée pour obtenir les résultats exposés dans le Chapitre 4 et le Chapitre 5.

En *Machine Learning*, la classification de données fait partie des méthodes d'apprentissage supervisé. Comme leur nom l'indique, ces méthodes consistent à superviser l'apprentissage d'une machine en lui montrant des exemples d'une tâche qu'elle doit effectuer. Plus particulièrement, une machine s'entraîne sur des données labellisées, c'est-à-dire des données qui sont déjà étiquetées avec le bon label, pour pouvoir ensuite prédire le label d'une nouvelle donnée non étiquetée. Dans une méthode d'apprentissage supervisé il y a donc toujours deux phases : l'apprentissage et le test pour vérifier si la machine a bien appris la tâche qu'elle doit réaliser. Si nous considérons un jeu de données destiné à être utilisé pour un algorithme d'apprentissage supervisé, nous devons alors le diviser en deux ensembles : l'ensemble des données d'entraînement qui conservent leur label et l'ensemble des données de test pour lesquelles la machine va tenter de prédire le bon label. Le premier ensemble est très souvent plus grand que le deuxième et est utilisé pour la phase d'apprentissage tandis que le deuxième est utilisé pour la phase de test, c'est-à-dire en tant que vérification. Grâce à cette phase d'apprentissage, les méthodes d'apprentissage supervisé permettent, par exemple, de prédire une certaine valeur appelée *target* à partir de variables appelées *features* [10]. Dans notre problématique de classification des états de conscience, les *features* sont les informations spectrales obtenues grâce aux signaux BOLD dans les différentes zones du cerveau et les *targets* ne sont pas des valeurs, mais des classes représentant les différents états de conscience. Le but de la classification est alors de pouvoir prédire l'état de conscience d'un nouveau patient à partir des informations spectrales obtenues via l'application d'une décomposition en modes dynamiques sur ses signaux BOLD. Le principe des méthodes de classification supervisée est illustré à la Figure 3.1.

Afin d'obtenir des résultats fiables, une méthode de *cross-validation* appelée *K-folds* a été utilisée durant la classification supervisée. Cette méthode permet d'assurer que toutes les observations de l'ensemble de données original aient la chance d'apparaître dans l'ensemble d'entraînement et dans l'ensemble de test. Pour commencer, il faut diviser l'ensemble de données en K *folds*, c'est-à-dire en K échantillons. La phase d'entraînement de la classification supervisée est alors effectuée grâce à $K-1$ *folds* et la phase de test est réalisée avec le dernier *fold* restant, le K-ième. Le processus est ensuite répété jusqu'à ce que chaque *fold* serve au sein de l'ensemble

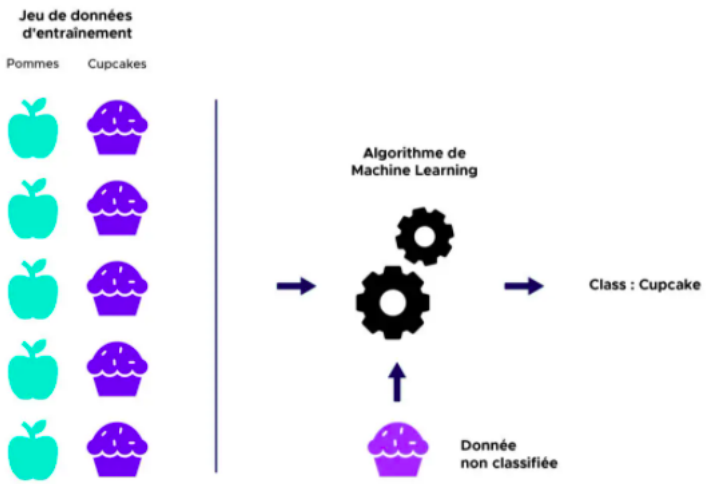

FIGURE 3.1 – Illustration du principe de la classification supervisée. Source : [38]

de test [10]. Cette méthode comporte donc un seul paramètre, le paramètre K, et le choix de ce paramètre est laissé libre. Dans le cadre de ce mémoire, il a été fixé à 18 afin d'utiliser les données de 17 patients dans l'ensemble d'entraînement et de réaliser la phase de test avec les données du patient restant. Ainsi, lors de la première itération, l'ensemble de test sera composé des données du premier patient et lors de la dernière itération, il sera composé des données du dernier patient.

Nous allons maintenant présenter le concept d'arbre de décision qui nous permettra de comprendre la méthode utilisée pour classer les états de conscience, appelée les *Extremely randomized trees*.

3.1 Arbre de décision

Cette section est dédiée à une introduction aux arbres de décision sur base du cours [10]. Un arbre de décision est un outil permettant de prendre des décisions sur base d'un ensemble de règles et qui prend la forme d'un arbre. Par exemple, cet ensemble de règles

- `Si (ordinateur s'allume) alors (utiliser ordinateur)`,
- `Si (ordinateur ne s'allume pas) et (ordinateur non branché) alors (brancher ordinateur)`,
- `Si (ordinateur ne s'allume pas) et (ordinateur branché) alors (appeler le service technique)`,

peut être représenté par l'arbre de décision présenté sur la Figure 3.2. De manière générale, tous les arbres de décision se lisent de haut en bas et ils sont composés de trois sortes de nœuds :

1. Le nœud racine : Ce nœud est tout au dessus de l'arbre de décision et c'est celui par lequel la lecture de l'arbre commence. Il n'a pas de nœud ascendant.
2. Les nœuds internes : Ces nœuds permettent de faire des choix, c'est-à-dire de décider le chemin à suivre dans l'arbre. Ils ont un nœud ascendant et au moins deux nœuds descendants qui peuvent être d'autres nœuds internes ou des feuilles.

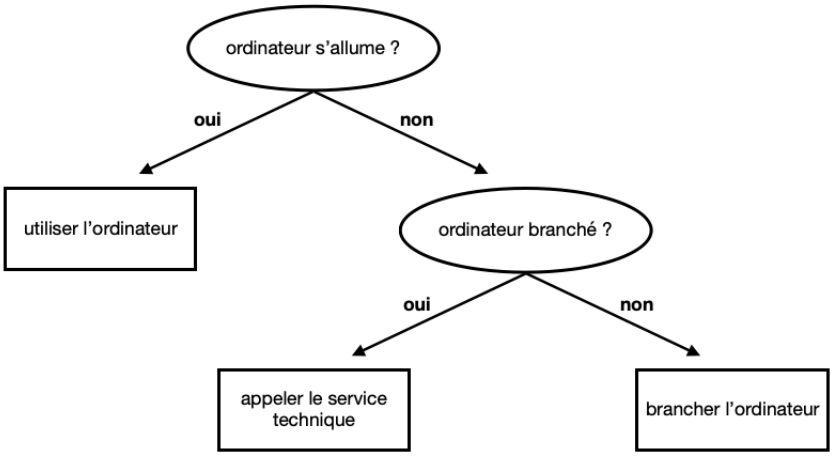

FIGURE 3.2 – Exemple d'arbre de décision. Source : [10]

3. Les feuilles : Ces nœuds sont tout en bas de l'arbre de décision et ils déterminent la décision finale à prendre. Ils ont un nœud ascendant mais pas de descendant.

Dans le cadre de la classification de données, les règles à suivre ne sont pas évidentes à déterminer. C'est pourquoi, les algorithmes d'apprentissage d'arbres de décision sont couramment utilisés car ils permettent de construire des arbres dont les nœuds sont déterminés sur base des *features* et *targets* d'un jeu de données. Dans un problème de classification, chaque élément $\boldsymbol{x}$ du jeu de données de départ est représenté par ses *features*, c'est-à-dire par un vecteur de variables $(x_1, x_2, ..., x_n)$. Le nœud racine et tous les nœuds internes correspondent alors à un test fait sur l'une des variables x_i. Pour déterminer quelle variable tester à un moment donné dans la construction d'un arbre, il faut choisir celle qui sépare le mieux les données par rapport à leur *target*. Cette séparation se fait sur l'ensemble des données de base si nous voulons construire le nœud racine ou sur les données issues du nœud précédent si nous voulons construire un nœud interne. La variable choisie sera alors celle qui maximise ce que l'on appelle le gain d'information et ce processus sera répété itérativement jusqu'à ce que les données soient toutes parfaitement séparées ou qu'un critère d'arrêt soit rencontré.

Le gain d'information de chaque *feature* est calculé grâce à une mesure qui quantifie le déséquilibre entre les classes entraîné par une séparation. L'une de ces mesures lorsque les *features* sont des données catégorielles est l'entropie de Shanon. Si nous notons D l'ensemble des données à séparer dans un nœud et $y_1, ..., y_m$ l'ensemble des *targets* possibles, nous pouvons calculer la probabilité p_j qu'un élément de D se retrouve dans y_j, avec $j = 1, ..., m$. L'entropie de Shanon de ce nœud vaut alors

$$H(D) = -\sum_{j=1}^{m} p_j . \log_2 \, p_j. \tag{3.1}$$

Ensuite, afin de calculer le gain d'information d'une *feature* x_i pour ce nœud, nous devons partitionner D en k groupes $D_1, ..., D_k$ pour les k valeurs distinctes que peut prendre cette *feature* x_i. Nous pouvons alors calculer la probabilité p_l qu'un élément de D appartienne à D_l et le gain d'information sera donné par la formule

N°	Prévision	Température	Humidité	Vent	Classe
1	Soleil	Chaud	Haute	Faux	N
2	Soleil	Chaud	Haute	Vrai	N
3	Couvert	Chaud	Haute	Faux	P
4	Pluie	Bon	Haute	Faux	P
5	Pluie	Frais	Normale	Faux	P
6	Pluie	Frais	Normale	Vrai	N
7	Couvert	Frais	Normale	Vrai	P
8	Soleil	Bon	Haute	Faux	N
9	Soleil	Frais	Normale	Faux	P
10	Pluie	Bon	Normale	Faux	P
11	Soleil	Bon	Normale	Vrai	P
12	Couvert	Bon	Haute	Vrai	P
13	Couvert	Chaud	Normale	Faux	P
14	Pluie	Bon	Haute	Vrai	N

TABLE 3.1 – Jeu de données. Source : [31]

$$\text{gain}(D, x_i) = H(D) - \sum_{l=1}^{k} p_l.H(D_l). \tag{3.2}$$

Pour mieux comprendre ce processus, prenons un exemple provenant de la source [31]. Dans cet exemple, un ensemble de samedis matin $(J_1, ..., J_{14})$ est classifié selon des données météo en deux catégories : "Adapté à une activité non spécifiée (P)" et "Pas adapté à cette activité (N)". Le jeu données disponible est détaillé sur le Tableau 3.1.

Dans ce cas, les *targets* des objets de jeu de données sont N ou P et ceux-ci sont représentés par des *features* catégorielles qui peuvent prendre plusieurs valeurs :

1. x_1 = Prévision (soleil, couvert, pluie),
2. x_2 = Température (chaud, bon, frais),
3. x_3 = Humidité (haute, normale),
4. x_4 = Vent (faux, vrai).

Si nous voulons construire l'arbre de décision qui pourra classifier ces données, nous devons commencer par choisir le nœud racine. Pour cela, nous allons calculer le gain d'information de chaque *feature* pour ce nœud et nous pouvons utiliser l'entropie de Shanon car les *features* sont catégorielles.

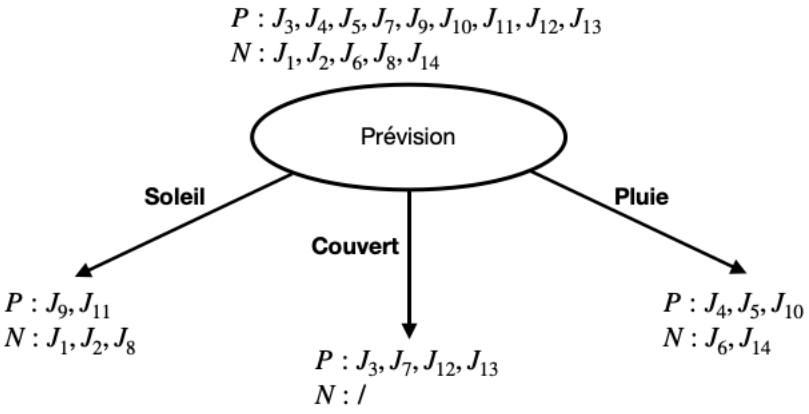

FIGURE 3.3 – Partition obtenue après avoir utilisé la *feature* "Prévision" comme nœud racine. Inspiré de : [31]

Au départ, le nœud racine contient tous les éléments $D = J_1, ..., J_{14}$. Si nous considérons que le premier critère de séparation se fait selon la *feature* x_1, nous obtenons un début d'arbre de décision illustré à la Figure 3.3.

En utilisant la formule (3.1), nous trouvons que l'entropie de ce nœud pour les *targets* P et N vaut

$$H(D) = -\frac{9}{14}.\log_2\frac{9}{14} - \frac{5}{14}.\log_2\frac{5}{14} = 0,940.$$

Ensuite, nous pouvons effectuer la division de ce nœud selon ses trois valeurs possible et nous obtenons la partition D_1, D_2, D_3 de l'ensemble D de départ. De la même manière que celle appliquée précédemment, nous pouvons calculer l'entropie des nœuds obtenus via la formule (3.1). Nous obtenons alors

$$\begin{aligned} H(D_1) &= -\frac{2}{5}.\log_2\frac{2}{5} - \frac{3}{5}.\log_2\frac{3}{5} = 0,971\ , \\ H(D_2) &= -\frac{4}{4}.\log_2\frac{4}{4} - \frac{0}{4}.\log_2\frac{0}{4} = 0\ , \\ H(D_3) &= -\frac{3}{5}.\log_2\frac{3}{5} - \frac{2}{5}.\log_2\frac{2}{5} = 0,971. \end{aligned}$$

Grâce à ces valeurs, nous pouvons maintenant calculer le gain d'information de la *feature* x_1 pour le nœud racine. En effet, en remplaçant les valeurs trouvées dans la formule (3.2), nous trouvons que le gain d'information vaut

$$\text{gain}(D, x_1) = 0,940 - \frac{5}{14}.0,971 - \frac{4}{14}.0 - \frac{5}{14}.0,971 = 0,246.$$

En procédant de façon similaire pour calculer les gains d'information des *features* x_2, x_3 et x_4, nous obtenons

$$\begin{aligned} \text{gain}(D, x_2) &= 0,029\ , \\ \text{gain}(D, x_3) &= 0,151\ , \\ \text{gain}(D, x_4) &= 0,048. \end{aligned}$$

Ainsi, selon cette méthode basée sur l'entropie de Shanon, le nœud racine de l'arbre de décision du jeu de données présenté au Tableau 3.1 doit séparer celui-ci selon la *feature* x_1, c'est-à-dire la *feature* "Prévision", car $\text{gain}(D, x_1) > \text{gain}(D, x_i),\ i \in \{2, 3, 4\}$.

Dans ce mémoire, nous allons uniquement travailler avec des *features* numériques mais le principe est le même que celui expliqué pour des *features* catégorielles. Pour une *feature* numérique x_i, le critère de coupure du nœud sera de la forme $x_i \leq c_i$, où c_i est un nombre sélectionné dans l'intervalle de valeurs que peut prendre la *feature* x_i.

Maintenant que les arbres de décisions ont été présentés, nous allons détailler la méthode des *Extremely randomized trees*.

3.2 Extremely randomized trees

Les *Extremely randomized trees* font partie des méthodes d'ensembles basées sur les arbres de décisions qui permettent de résoudre des problèmes de classification et de régression supervisée. Dans le cadre de ce mémoire, ils vont donc être utilisés pour classifier les différents états de conscience des patients car ils sont efficaces et donnent généralement de bons résultats dans le domaine de la classification. La méthode des *Extremely randomized trees* est dite d'ensemble car elle va combiner plusieurs algorithmes d'apprentissage qui sont, dans ce cas-ci, des arbres de décision.

Cette méthode a été présentée pour la première fois dans un article publié en 2006 et c'est sur cet article que nous nous baserons dans cette section [12]. L'algorithme des *Extremely randomized trees* (ou *Extra-trees*) est composé de plusieurs arbres de décision et il est donc similaire aux autres méthodes d'ensemble basées sur les arbres de décision à deux différences près. La première différence est que cet algorithme divise les nœuds en choisissant les points de coupe de manière totalement aléatoire au lieu de choisir le critère optimal à l'aide de l'entropie de Shanon par exemple. Cela permet d'augmenter la vitesse d'exécution de l'algorithme et de réduire les biais pouvant apparaître dans la production du résultat de la classification. En effet, dans les années 90, des études ont révélé que la plupart des méthodes basées sur les arbres de décision avaient une variance élevée entraînant une grande diversité des résultats lorsque différents ensembles d'entraînement sont utilisés. Cela est du au fait que le choix du point de coupure dans un nœud dépend très fortement de l'échantillon d'entraînement particulier utilisé. De plus, il a été démontré que cette variance élevée semble être responsable d'une grande partie des taux d'erreurs des méthodes basées sur les arbres de décision [12]. Pour réduire cette variance, le modèle *Extra-trees* va, pour chaque nœud, sélectionner son point de coupe de manière totalement aléatoire, c'est-à-dire indépendamment de la *target* de chaque élément qu'il doit classifier. Dans le cas d'une classification avec des *features* numériques, un choix aléatoire d'un certain nombre de *features* x_i ainsi que leur critère de coupure c_i est fait et parmi tous les fractionnements générés aléatoirement, celui qui donne le gain d'information le plus élevé est choisi pour diviser le n

œeud. Ce nombre de *features* sélectionnées est régulé par un paramètre de l'algorithme noté K. Dans le cas extrême, lorsque $K = 1$, la méthode choisit donc aléatoirement une seule *feature* et un seul point de coupe à chaque nœud et construit donc des arbres totalement indépendant de l'ensemble d'entraînement. La deuxième différence est que l'algorithme *Extra-trees* construit ses arbres de décision en utilisant toutes les données de l'échantillon d'entraînement, contrairement aux autres méthodes d'ensemble basées sur les arbres de décision qui ne considèrent que des échantillons *bootstrap*, c'est-à-dire une partie de l'ensemble d'entraînement. Bien que cette méthode appelée le *Bagging* permette également de réduire la variance d'un algorithme, les auteurs de l'article affirment que cette méthode provoque des perturbations dans la recherche du critère de fractionnement optimal des nœuds.

Une fois que les différents arbres de la méthode *Extremely randomized trees* ont été construits, la décision finale pour classifier les données est obtenue en tenant compte de la prédiction de chaque arbre. En effet, la *target* choisie est la classe sélectionnée par la majorité des arbres de décision individuels.

L'Algorithme 3 reprend le pseudo-code de l'algorithme *Extra-trees*. Celui-ci est divisé en trois procédures qui permettent de séparer les nœuds, construire un arbre de décision et combiner les différents arbres de décisions individuels pour construire la forêt d'arbres. Nous pouvons observer que trois paramètres sont importants pour construire cet algorithme. Il s'agit du paramètre K déjà abordé précédemment, du paramètre M qui détermine le nombre d'arbres construits lors de l'exécution de l'algorithme et du paramètre $n_{\min}$ qui donne la taille minimale de l'échantillon de données requise pour permettre la séparation d'un nœud. Le choix de ces paramètres a été discuté dans l'article [12] et nous allons reprendre les valeurs recommandées, c'est-à-dire

- $K = \sqrt{n}$, où n est le nombre de *features* disponibles,
- $M = 100$ et
- $n_{\min} = 2$.

En pratique, le classificateur utilisé dans ce mémoire est la classe `ExtraTreesClassifier` du module `sklearn.ensemble` dans lequel les paramètres K, M et $n_{\min}$ sont noté respectivement `max_features`, `n_estimators` et `min_samples_split`. Les valeurs par défaut de ces paramètres sont celles données dans l'article [12] donc nous n'allons pas les modifier. Cependant, d'autres paramètres peuvent prendre d'autres valeurs que celles attribuées par défaut par le modèle `ExtraTreesClassifier`. Un autre paramètre dont la valeur sera discutée dans le Chapitre 4 est le paramètre noté `max_depth` qui régule la profondeur maximale des arbres, c'est-à-dire le nombre de nœuds par lesquels les données passent en comptant le nœud racine et la feuille dans laquelle elles terminent. Nous allons considérer ce paramètre car en lui donnant une valeur particulière, nous pouvons parfois éviter le phénomène de surapprentissage des arbres de décision par rapport aux éléments de l'ensemble d'entraînement et ainsi aider le modèle à généraliser les règles de classification pour pouvoir mieux classifier les éléments de l'ensemble de test.

Nous allons maintenant présenter un modèle couramment utilisé dans les méthodes de classification des documents car il permet de générer des *features* à partir de mots. Nous allons également découvrir comment ce modèle peut s'appliquer à des séries temporelles si elles sont considérées comme des documents.

Algorithme 3 Extra-trees. Source : [12].

Entrée: Un ensemble d'entraînement $\mathcal{S}$
Sortie: Un ensemble d'arbres $\mathcal{T} = \{t_1, ..., t_M\}$
  procedure BUILD_AN_EXTRA_ENSEMBLE($\mathcal{S}$)
    for $i = 1, ..., M$ **do**
      Générer un arbre : t_i = BUILD_AN_EXTRA_TREE($\mathcal{S}$)
    end for
  end procedure

Entrée: Un ensemble d'entraînement $\mathcal{S}$
Sortie: Un arbre t
  procedure BUILD_AN_EXTRA_TREE($\mathcal{S}$)
    if $|\mathcal{S}| < n_{min}$ **or** toutes les *features* candidates sont constantes dans $\mathcal{S}$ **or** la *target* est constante dans $\mathcal{S}$ **then**
      return Une feuille labelisée par les fréquences de classe
    else
      1 : Sélectionner K *features* $\{x_1, ..., x_K\}$ sans remise parmi toutes les *features* non constantes disponibles dans $\mathcal{S}$
      2 : Générer K séparations $\{s_1, ..., s_K\}$ telles que s_i = PICK_A_RANDOM_SPLIT($\mathcal{S}, x_i$), $\forall i = 1, ..., K$
      3 : Sélectionner une séparation s^* telle que $\text{Score}(s^*, \mathcal{S}) = \max_{i=1,...,K} \text{Score}(s_i, \mathcal{S})$
      4 : Séparer $\mathcal{S}$ en deux sous-ensembles $\mathcal{S}_g$ et $\mathcal{S}_d$ selon la séparation s^*
      5 : Construire deux nouveaux arbres t_g = BUILD_AN_EXTRA_TREE($\mathcal{S}_g$) et t_d = BUILD_AN_EXTRA_TREE($\mathcal{S}_l$)
      6 : Créer un nœud avec la séparation s^* et y rattacher les deux nouveaux arbres t_g et t_d
      return L'arbre t
    end if
  end procedure

Entrée: Un ensemble d'entraînement $\mathcal{S}$ et une *feature* x
Sortie: Une séparation
  procedure PICK_A_RANDOM_SPLIT($\mathcal{S}, x$)
    1 : Calculer la valeur minimale et maximale de x dans $\mathcal{S}$, notées $x^{\mathcal{S}}_{\text{min}}$ et $x^{\mathcal{S}}_{\text{max}}$
    2 : Extraire un seuil c de manière uniforme dans $[x^{\mathcal{S}}_{\text{min}}; x^{\mathcal{S}}_{\text{max}}]$
    return La séparation $x < c$
  end procedure

3.3 Modèle du sac de mots

Le modèle du sac de mots (*Bag of Words*, BoW) est apparu dans la littérature pour la première fois en 1954 et à l'origine, il a été développé pour modéliser des documents textuels afin de pouvoir les analyser grâce à des techniques telles que des algorithmes de *Machine Learning*. En effet, ces algorithmes ne peuvent pas travailler directement avec du texte brut donc ce texte doit être converti en des données adaptées. Il s'agit de l'extraction ou le codage de *features*, les *features* étant, pour rappel, les variables qui agissent comme entrées d'un algorithme. Une méthode simple et populaire d'extraction de *features* de données textuelles est le modèle du sac de mots. Elle permet de convertir le texte en chiffres ou plus précisément en vecteurs de nombres. Dans ce modèle, un texte est représenté par l'occurrence de ses mots, sans tenir compte de l'ordre de ceux-ci. Le modèle se préoccupe donc uniquement de la présence de mots connus dans le document, et non de leur emplacement [4].

Nous allons illustrer le principe du modèle du sac de mots par un exemple simple, inspiré de la référence [4]. Premièrement, nous prenons un ensemble de phrases qui constituent un document, à savoir

1. Ophélie aime les sushis et Sacha aime les burritos.
2. Sacha aime aussi les sushis.

Deuxièmement, nous pouvons dresser une liste de tous les mots de vocabulaire, appelée *codebook*, qui vont alimenter notre modèle du sac de mots. Les mots de vocabulaire que nous allons retenir sont appelés *codewords* et ce sont les mots qui apparaissent au moins une fois dans le document choisi ci-dessus. Pour notre exemple, le *codebook* est composée des mots

"Ophélie", "aime", "les", "sushis", "et", "Sacha", "burritos", "aussi".

Nous avons donc une liste de huit mots pour un document contenant quatorze mots. La troisième et dernière étape du modèle du sac de mots consiste à compter le nombre de fois que les mots de la liste apparaissent dans chaque phrase du document. L'objectif de cette étape est de transformer les données textuelles en des vecteurs qui peuvent être utilisés en entrée d'un algorithme de *Machine Learning*. Comme nous savons que la liste de vocabulaire compte huit mots, nous pouvons utiliser un vecteur de taille fixe huit pour représenter chaque phrase du document. Ce vecteur est unique pour chaque phrase et l'occurrence des mots, c'est-à-dire le nombre de fois qu'ils apparaissent dans la phrase, constituent les composantes du vecteur. En utilisant l'ordre arbitraire des mots énumérés dans notre *codebook*, nous pouvons convertir les phrases en vecteurs numériques, de sorte que

1. "Ophélie aime les sushis et Sacha aime les burritos" = [1, 2, 2, 1, 1, 1, 1, 0].
2. "Sacha aime aussi les sushis" = [0, 1, 1, 1, 0, 1, 0, 1].

Notons que pour cette dernière étape, les composantes des vecteurs peuvent aussi représenter une normalisation de la fréquence d'un terme dans un texte ou tout simplement une pondération binaire de ce terme (1 si le mot est présent dans le texte et 0 s'il ne l'est pas).

Le modèle du sac de mots est très efficace pour capturer des informations concernant un document et c'est pourquoi il est utilisé dans de nombreux domaines. L'application la plus courante est évidemment l'analyse de textes comme le filtrage des e-mails afin d'éviter les spams. Pour

cela, la méthode du sac de mots compte les mots des e-mails qui sont un *codebook* pré-défini contenant par exemple des termes comme "acheter", "gagnant, "lot, "don", etc. Grâce à cela, un histogramme de *codewords* est créé pour chaque e-mail fournissant une représentation graphique de celui-ci. Cependant, le modèle du sac de mots a récemment été étendu à l'analyse d'images ou de vidéos. Dans ce cas, les *features* locales extraites des images ou des vidéos sont traitées comme des mots et ces images ou ces vidéos sont alors traitées comme des documents. Le vocabulaire utilisé est donc différent mais le principe reste le même [34].

De plus, nous pouvons encore étendre la représentation du modèle du sac de mots pour caractériser des séries temporelles de la même manière que pour des images ou des vidéos. C'est cette application que propose la référence [34], dans laquelle des segments locaux extraits des séries temporelles sont considérés comme des mots et chaque série temporelle est traitée comme un document. La procédure générale utilisée pour générer la représentation du modèle du sac de mots pour des séries temporelles est illustrée à la Figure 3.4 et elle comprend deux étapes qui sont les suivantes :

1. Génération du *codebook*, et
2. Représentation du modèle du sac de mots.

Pour la première étape, c'est-à-dire la génération du *codebook*, nous allons tout d'abord extraire un groupe de segments locaux à partir de chaque série temporelle des données d'entraînement. Ces segments locaux sont obtenus en faisant glisser une fenêtre d'une taille fixe prédéfinie le long des séries temporelles. Ensuite, la méthode numérique de décomposition en modes dynamiques est appliquée à chaque segment local afin de calculer une approximation les valeurs propres et les modes de Koopman. Ces informations spectrales représentent chaque segment temporel extrait des séries temporelles de départ et il nous reste alors à générer le *codebook* en partitionnant ces informations. Dans un cadre traditionnel du modèle du sac de mots, les éléments considérés pour générer le *codebook* sont dans un espace euclidien et peuvent donc être regroupés grâce à une méthode de *clustering*. Le problème est que dans notre cas, les informations spectrales obtenues se trouvent dans un espace non-euclidien. Afin de contourner ce problème, nous allons d'abord calculer la distance qui sépare chaque segment en utilisant une métrique appelée la première distance de Wasserstein.

La première distance de Wasserstein, aussi appelée *Earth mover's distance*, provient de la théorie du problème de transport optimal [37]. Cette distance s'exprime généralement entre deux densités de probabilité μ et ν et est définie telle que

$$W(\mu, \nu) = \inf_{\pi \in \Pi(\mu,\nu)} \int_{\mathcal{X}} d(x, y) d\pi(x, y),$$

où μ, ν appartiennent à l'espace $\mathcal{X}$ muni d'une métrique $d(\cdot, \cdot)$ et $\Pi(x, y)$ est l'ensemble des distributions de probabilité conjointes π sur $\mathcal{X}$. Dans le cadre de ce mémoire, cette distance va être utilisée pour calculer la distance entre deux segments locaux. Plus précisément, la première distance de Wasserstein va permettre de déterminer la distance entre deux vecteurs contenant les approximations des valeurs propres de deux segments locaux. Ainsi, la distance entre chaque segment local pourra être calculée et stockée dans une matrice D.

Une fois que la matrice D reprenant les premières distances de Wasserstein entre chaque paire de segments locaux a été construite, nous pouvons passer à la dernière étape de la génération du *codebook*. Celle-ci consiste à partitionner les segments locaux grâce à un algorithme de *clustering*.

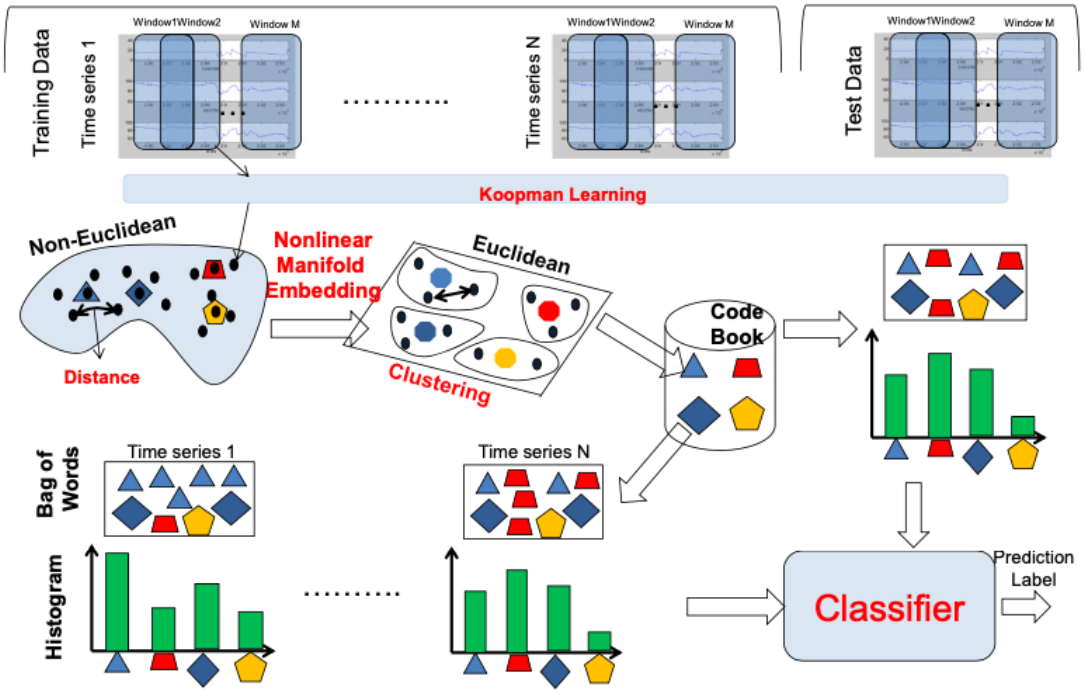

FIGURE 3.4 – Procédure basée sur la théorie de l'opérateur de Koopman pour générer automatiquement des *features* adaptées à la classification de séries temporelles. La partie gauche de la figure illustre la génération d'un *codebook* à partir des données d'entraînement. Ensuite, les *codewords* sont utilisés afin de créer un histogramme pour chaque série temporelle d'entraînement. Ces histogrammes correspondent aux *features* des séries temporelles utilisées pour entraîner le classificateur. La partie de droite, quant à elle, illustre la génération d'un histogramme pour la série temporelle de test en utilisant le *codebook* précédemment défini afin de déterminer sa *target*. Source : [34]

Dans ce mémoire, nous allons choisir un algorithme de *clustering* appelé k-médoïdes car il permet de partitionner les éléments de la matrice D sans devoir les plonger dans un espace euclidien. Les explications suivantes sont tirées des références [10] et [26]. Les méthodes de *clustering* font partie des méthodes d'apprentissage non-supervisé qui, contrairement aux méthodes d'apprentissage supervisé, considèrent des données qui n'ont pas de label. Il n'y a donc pas de phase d'apprentissage et la machine doit en apprendre par elle-même sur les données qui lui sont fournies. Dans le cas du *clustering*, la méthode détecte des similarités entre les données afin de les diviser en différents groupes appelés *clusters*. La méthode de *clustering* k-médoïdes est une variante d'une autre méthode de *clustering* appelée *k-means* et c'est cette dernière que nous allons commencer par présenter. L'algorithme *k-means* nécessite de faire le choix du nombre k de *clusters* souhaités à l'avance pour qu'il soit initialisé avec k points choisis au hasard dans l'espace ou dans les données fournies à l'algorithme. Ces k points seront temporairement désignés comme les centres des *clusters* et lors de la première itération, l'algorithme va assigner le reste des données à l'un de ces k centres, aussi appelés centroïdes ou prototypes, de manière à ce que la distance entre ces points et leur centre soit minimale. Lorsque chaque donnée est assignée à un centre, k *clusters* sont formés et le centroïde de chaque *cluster* formé peut être mis à jour en le remplaçant par le centre de gravité du *cluster*. Ces deux étapes sont répétées jusqu'à ce que les centroïdes se stabilisent, c'est-à-dire jusqu'à ce qu'ils ne bougent plus durant plusieurs itérations, ou jusqu'à ce qu'un nombre d'itérations fixé à l'avance soit atteint. L'algorithme k-médoïdes fonctionne de manière analogue, mais il choisit les prototypes, c'est-à-dire les points qui vont représenter les *clusters*, parmi les données réelles appartenant aux *clusters* au lieu de prendre le centre de gravité de ceux-ci. L'avantage de l'algorithme k-médoïdes est que, contrairement à l'algorithme *k-means* considérant des moyennes facilement influençables par des valeurs extrêmes, il n'est pas sensible aux valeurs aberrantes et se dit donc plus robuste.

Dans le modèle du sac de mots, l'algorithme k-médoïdes est appliqué sur la matrice D des premières distances de Wasserstein dont chaque ligne représente un segment local extrait des séries temporelles de départ. Par exemple, la première ligne de la matrice D reprend les distances qui séparent le premier segment local de tous les autres et constitue donc un vecteur d'informations concernant le premier segment local. Nous pouvons alors voir chaque ligne de D comme un vecteur de *features* correspondant à une donnée que l'algorithme k-médoïdes prendra en compte et placera dans l'un des k *clusters*. Au terme de l'exécution de l'algorithme, k prototypes seront désignés parmi les données et ces k prototypes formeront le *codebook* de taille k de la méthode du sac de mots, chaque prototype étant un *codeword*.

La deuxième étape consiste à représenter le modèle du sac de mots. Pour ce faire, il faut représenter chaque série temporelle de départ comme un histogramme du *codebook*, c'est-à-dire calculer l'occurrence à laquelle les k *codewords* apparaissent dans celle-ci. Pour commencer, nous allons regrouper tous les segments locaux extraits d'une même série temporelle et observer à quel *cluster*, et donc à quel prototype, ils ont été assigné par l'algorithme k-médoïdes. Une fois cette étape franchie, nous pouvons déterminer l'occurrence de chaque prototype, c'est-à-dire *codeword*, dans les séries temporelles et représenter celles-ci par des histogrammes de *codewords*. La mesure d'occurrence utilisée est la méthode de pondération TF-IDF (*Term Frequency - Inverse Document Frequency*). Pour expliquer ce concept, nous nous basons sur la référence [22]. La méthode TF-IDF permet de quantifier la pertinence d'un mot dans un document parmi une collection de documents. Cette méthode va alors repérer la présence de mots anormalement fréquents dans certains documents par rapport aux autres documents de la collection. Pour déterminer l'importance d'un mot i dans un document j de la collection, la méthode TF-IDF combine deux termes. Le premier est TF_{ij}, la fréquence du terme i dans le document j, dont la

valeur est donnée par la formule

$$\mathrm{TF}_{ij} = \frac{\text{Nombre de fois que } i \text{ apparaît dans } j}{\text{Nombre total de mots dans } j}.$$

Le deuxième terme considéré par la méthode TF-IDF est IDF_i, la fréquence inverse des documents par rapport au mot i. Ce terme calcule donc l'importance de i dans la collection de documents grâce à la formule

$$\mathrm{IDF}_i = \log\left(\frac{\text{Nombre total de documents dans la collection}}{\text{Nombre de documents contenant } i}\right).$$

La valeur de TF-IDF_{ij} du mot i et du document j est alors donnée par la formule

$$\text{TF-IDF}_{ij} = \mathrm{TF}_{ij} \times \mathrm{IDF}_i.$$

Au terme de ce processus, chaque fenêtre temporelle de départ sera représentée par des histogrammes de *codewords* grâce à la méthode TF-IDF. Ces histogrammes seront considérées comme les *features* des séries temporelles lors des différentes classifications que nous allons effectuer dans le Chapitre 5.

3.4 Mesures de la qualité d'une classification

Dans cette section, nous allons présenter les différentes métriques utilisées pour mesurer la qualité d'une classification. Ces métriques permettent de comparer les performances de deux classificateurs différents ou d'analyser le comportement d'un même classificateur en modifiant différents paramètres de celui-ci. Les métriques choisies sont la matrice de confusion, l'*accuracy*, le *F1-score* et le *log loss*. Les explications suivantes sont inspirées des références [3] et [14].

Bien que dans le cadre de ce mémoire, nous serons amené à faire de la classification avec plus de deux classes, appelée classification multiclasses, nous allons d'abord prendre l'exemple d'un problème de classification avec deux classes, appelé problème de classification binaire. Lorsque la majorité de la société était touchée par la maladie du COVID-19, il était important de savoir si une personne était négative ou positive au COVID-19. Pour cela, des tests étaient effectué et quatre types de résultats pouvaient se produire :

1. **Vrais positifs (TP) :** Cas où une personne malade est classée comme positive au COVID-19,
2. **Faux positifs (FP) :** Cas où une personne saine est classée comme positive au COVID-19,
3. **Vrais négatifs (TN) :** Cas où une personne saine est classée comme négative au COVID-19,
4. **Faux négatifs (FN) :** Cas où une personne malade est classée comme négative au COVID-19.

Toutes ces informations sont reprises dans la matrice de confusion qui est un tableau croisé enregistrant le nombre d'occurrence de chaque prédiction par rapport à la réalité. Généralement, les colonnes représentent les résultats de la classification prédite et les lignes celles de la classification réelle. Les classes sont énumérées dans le même ordre pour les lignes et les colonnes

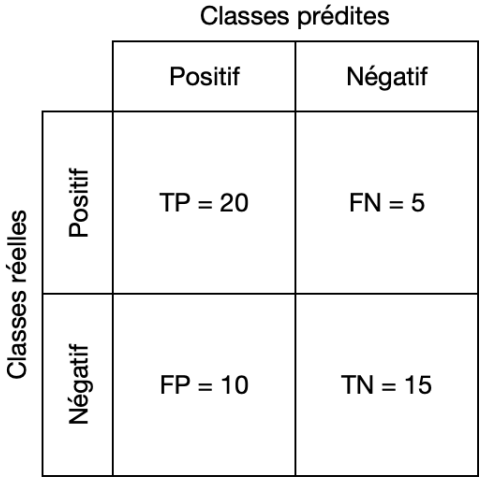

FIGURE 3.5 – Matrice de confusion pour un problème de classification binaire. Source : [14]

et par conséquent, les éléments correctement classés sont situés sur la diagonale principale de la matrice de confusion.

Si nous reprenons l'exemple concernant la maladie du COVID-19 et que nous testons 50 personnes dont la moitié est réellement atteinte de la maladie, nous pourrions obtenir la matrice de confusion affichée à la Figure 3.5. Dans ce cas, 20 personnes malades et 10 personnes saines sont testées positives (TP et FP), et 5 personnes malades et 15 personnes saines sont testées négatives (FN et TN). Cela signifie que 35 personnes (TP+TN) ont été correctement classifiées et que 15 personnes (FP+FN) ne l'ont pas été.

De nombreuses métriques sont basées sur la matrice de confusion et l'*accuracy* en fait partie. Ce terme n'est pas traduit par "précision" dans ce mémoire car la précision sera une autre métrique à part entière et cela permet de ne pas introduire de confusion entre les deux métriques. L'*accuracy* est l'une des mesures les plus populaires en classification et elle est directement calculée à partir de la matrice de confusion. Cette métrique permet de calculer la proportion d'éléments correctement classifiés par le modèle. En d'autres termes, si nous prenons un élément au hasard dans l'ensemble de test et que nous prédisons sa classe au moyen d'un classificateur, l'*accuracy* donne la probabilité que cette prédiction soit correcte. La formule de l'*accuracy* considère tous les éléments correctement classifiés au numérateur et la somme de toutes les entrées de la matrice de confusion au dénominateur. Elle est donc donnée par

$$\text{accuracy} = \frac{TP + TN}{TP + FP + TN + FN}.$$

Pour l'exemple du COVID-19 et grâce à la matrice de confusion affichée sur la Figure 3.5, nous trouvons que l'*accuracy* de ce problème de classification vaut

$$\text{accuracy} = \frac{20 + 15}{20 + 10 + 15 + 5} = 0,7.$$

Il y a donc 70% de chances qu'une personne testée pour la maladie du COVID-19 soit bien

classifiée.

Une autre métrique utilisée dans ce mémoire et basée sur la matrice de confusion est le *F1-score*. Cette métrique mesure la capacité d'un modèle à bien prédire les individus positifs en faisant intervenir deux autres métriques : la précision (ou valeur prédictive positive) et le rappel (ou sensibilité). La précision est la proportion d'individus correctement classés comme positifs parmi tous les individus classés comme positifs par le modèle. La formule de la précision est donc obtenue en divisant les vrais positifs par la somme des vrais positifs et des faux positifs, c'est-à-dire

$$\text{Précision} = \frac{TP}{TP + FP}.$$

L'optimisation de la précision permet donc de diminuer le nombre de faux positifs prédits par le modèle de classification.

Le rappel, quant à lui, correspond à la proportion d'individus réellement positifs qui est correctement classée. La formule du rappel s'obtient donc en divisant les vrais positifs par la somme des vrais positifs et des faux négatifs, c'est-à-dire

$$\text{Rappel} = \frac{TP}{TP + FN}.$$

L'optimisation du rappel permet, cette fois, de diminuer le nombre de faux négatifs prédits par le modèle de classification.

Comme le *F1-score* combine la précision et le rappel, il permet de résumer ces deux valeurs en une seule métrique. La formule du *F1-score* est une moyenne harmonique de la précision et du rappel et elle est donnée par

$$\text{F1-score} = \frac{2}{\frac{1}{\text{Précision}} + \frac{1}{\text{Rappel}}} = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}.$$

Le *F1-score* est donc un compromis entre la précision et le rappel. Par conséquent, lorsque la précision diminue par exemple, le terme $\frac{1}{\text{Précision}}$ augmente et le *F1-score* baisse également. La précision et le rappel prenant des valeurs entre 0 et 1, le *F1-score* atteint sa valeur optimale à 1 et son pire score à 0. Lorsque le *F1-score* prend la valeur de 1, cela signifie que le classificateur ne prédit aucun faux positif et aucun faux négatif.

Dans le cas de l'exemple du COVID-19 et grâce à la matrice de confusion présente à la Figure 3.5, nous pouvons calculer la précision, le rappel et le *F1-score* tels que

$$\begin{aligned}
\text{Précision} &= \frac{20}{20 + 10} = 0,67\ , \\
\text{Rappel} &= \frac{20}{20 + 5} = 0,8 \text{ et} \\
\text{F1-score} &= 2 \times \frac{0,67 \times 0,8}{0,67 + 0,8} = 0,73.
\end{aligned}$$

La dernière métrique que nous allons présenter n'est pas basée sur la matrice de confusion. Il s'agit du *log loss* (ou *cross entropy*) qui est la métrique de classification la plus importante

basée sur les probabilités et l'une des métriques à un seul chiffre les plus robustes. Contrairement à l'*accuracy* ou au *F1-score*, le but est de minimiser le *log loss* pour améliorer la qualité d'un classificateur car c'est une fonction d'erreur. Comme cette métrique se base sur les probabilités qu'un classificateur calcule afin d'attribuer une classe à un élément, elle prend en compte l'incertitude du modèle. Plus particulièrement, si pour une observation particulière, le classificateur attribue une très faible probabilité à la classe correcte, la contribution correspondante au *log loss* sera très importante. Dans le cas d'une classification binaire, pour une *target* $y \in \{0, 1\}$ et pour une estimation de probabilité $p = P(y = 1)$, la formule du *log loss* est

$$\mathrm{L}_{\log}(y, p) = -\log P(y|p) = -(y \log(p) + (1 - y) \log(1 - p)).$$

Il est difficile d'interpréter la valeur d'un *log loss* mais cette métrique est assez efficace pour comparer la performance globale d'un classificateur à celle d'un autre et c'est dans cette optique qu'elle va être utilisée dans ce mémoire.

Les quatre métriques présentées sont définies par défaut pour des problèmes de classification binaire. Pour les étendre aux problèmes de classification multiclasses, nous allons utiliser l'approche *One-vs-Rest*, aussi appelée *One-vs-All*, qui consiste à décomposer le problème initial en une série de différents problèmes binaires. Dans le cadre de ce mémoire, nous devons classifier différents patients en quatre classes : "conscient", "moyennement endormi", "profondément endormi" et "en phase de réveil". Nous pouvons alors convertir ce problème de classification à quatre classes en quatre problèmes binaires :

1. Tâche 1 : conscient vs. [moyennement endormi, profondément endormi, en phase de réveil], c'est-à-dire conscient vs. pas conscient,
2. Tâche 2 : moyennement endormi vs. [conscient, profondément endormi, en phase de réveil], c'est-à-dire moyennement endormi vs. pas moyennement endormi,
3. Tâche 3 : profondément endormi vs. [conscient, moyennement endormi, en phase de réveil], c'est-à-dire profondément endormi vs. pas profondément endormi,
4. Tâche 4 : en phase de réveil vs. [conscient, moyennement endormi, profondément endormi], c'est-à-dire en phase de réveil vs. pas en phase de réveil.

La matrice de confusion sera une matrice avec quatre lignes et quatre colonnes correspondant aux quatre états de conscience cités ci-dessus et un exemple de cette matrice peut être observé à la Figure 3.6. Grâce à cette matrice, nous pouvons observer que le classificateur a classifié comme "profondément endormi" sept patients qui étaient en réalité moyennement endormis. Cette erreur est conséquente et cela se ressent dans le calcul de l'*accuracy* qui, dans ce cas, vaut

$$\text{accuracy} = \frac{6 + 2 + 9 + 8}{6 + 5 + 4 + 3 + 5 + 2 + 7 + 4 + 3 + 5 + 9 + 1 + 3 + 4 + 3 + 8} = \frac{25}{72} = 0,35.$$

Cela signifie que si nous prenons l'état d'un patient au hasard et que nous le classifions, la probabilité que cette classification soit correcte est de 0,35.

Pour mesurer la qualité des classificateurs multiclasses, le *F1-score* est couramment utilisé lorsque les classes sont déséquilibrées, c'est-à-dire lorsqu'il y a une grande différence du nombre d'éléments dans chaque classe. Cependant, il est également possible de l'utiliser afin de détecter les classes pour lesquelles le classificateur est le plus efficace. En effet, nous pouvons calculer le *F1-score* de chaque classe grâce à la méthode *One-vs-Rest* en faisant comme s'il y avait des

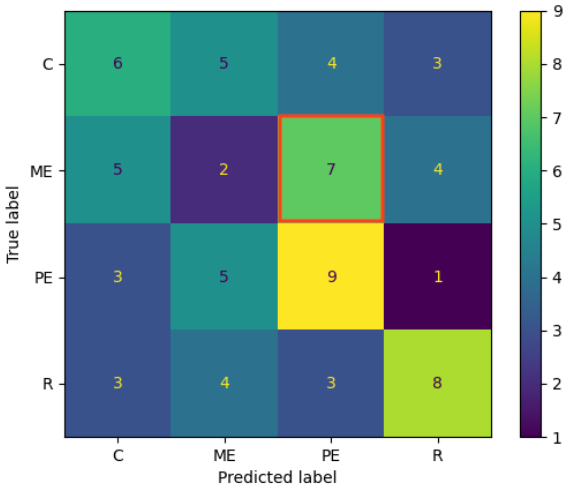

FIGURE 3.6 – Exemple de matrice de confusion pour le problème de classification des états de conscience. "C" désigne l'état "conscient", "ME" désigne l'état "moyennement endormi", PE désigne l'état "profondément endormi" et "R" désigne l'état "en phase de réveil". La valeur encadrée en rouge montre que le classificateur a classifié comme "profondément endormi" sept patients qui étaient en réalité moyennement endormis.

classificateurs distincts pour chaque classe. Si nous voulons calculer le *F1-score* de la classe "conscient" par exemple, nous allons considérer l'état "conscient" comme positif et tous les autres états comme négatifs. En reprenant la matrice de confusion affichée sur la Figure 3.6 et en calculant la précision, le rappel et le *F1-score* correspondant à la classe "conscient" (C), nous trouvons alors

$$\begin{aligned}
\text{Précision}(\text{Classe} = C) &= \frac{6}{6+5+3+3} = \frac{6}{17} = 0,35\ , \\
\text{Rappel}(\text{Classe} = C) &= \frac{6}{6+5+4+3} = \frac{6}{18} = 0,33\ , \\
\text{F1-score}(\text{Classe} = C) &= 2 \times \frac{0,35 \times 0,33}{0,35+0,33} = 0,34.
\end{aligned}$$

En effectuant le même calcul du *F1-score* pour les trois autres classes "moyennement endormi" (ME), "profondément endormi" (PE) et "en phase de réveil" (R), nous obtenons

$$\begin{aligned}
\text{F1-score}(\text{Classe} = ME) &= 0,12\ , \\
\text{F1-score}(\text{Classe} = PE) &= 0,44\ , \\
\text{F1-score}(\text{Classe} = R) &= 0,47.
\end{aligned}$$

Ce résultat prouve que même si le classificateur a correctement classé l'état de conscience "profondément endormi" neuf fois, ce n'est pas pour cet état de conscience qu'il est le plus efficace car il a également classé comme "profondément endormi" sept patients qui étaient en réalité dans l'état "moyennement endormi", entrainant une baisse de la précision pour la classe "profondément endormi". En terme de *F1-score*, nous pouvons donc conclure que l'état de conscience que

le classificateur prédit le mieux est l'état "en phase de réveil".

Si nous considérons la métrique *log loss* pour évaluer la qualité d'un classificateur multiclasses, nous devons à nouveau utiliser la méthode *One-vs-Rest* et considérer une classe après l'autre. Une probabilité de 1 sera alors attribuée à la classe correcte et les autres classes se verront attribuer une probabilité de 0.

En pratique, dans ce mémoire, pour mesurer la qualité d'un classificateur implémenté en langage `Python` grâce aux quatre métriques présentées dans cette section, nous allons utiliser les fonctions `sklearn.metrics.confusion_matrix`, `sklearn.mectrics.accuracy_score`, `sklearn.mectrics.f1_score` avec le paramètre `average=None` afin d'obtenir un *F1-score* pour chaque classe et `sklearn.mectrics.log_loss`.

3.5 Conclusion

Dans ce chapitre, nous avons d'abord introduit le concept d'apprentissage supervisé en *Machine Learning*. Ces méthodes permettent de superviser l'apprentissage d'une machine en lui donnant des exemples de tâches à effectuer. Le but de ce mémoire étant de classifier les états de conscience de certains patients, nous allons implémenter un classificateur qui s'entraînera sur un ensemble d'entraînement et dont nous pourrons vérifier la qualité sur un ensemble de test.

Ensuite, la méthode de classification que nous allons considérer dans ce mémoire a été détaillée et il s'agit de celle des *Extremely randomized trees*. C'est une méthode d'ensemble basée sur les arbres de décision qui introduit de l'aléatoire dans le choix du critère de séparation des données dans ses nœuds. Cela permet d'augmenter la vitesse d'exécution de l'algorithme et de minimiser la variance des résultats lorsque différents ensembles d'entraînement sont utilisés.

De plus, nous avons développé une méthode couramment utilisée dans la classification de documents, appelée le modèle du sac de mots. En faisant intervenir une méthode de décomposition en modes dynamiques, nous avons découvert comment ce modèle pouvait s'appliquer à des données temporelles pour extraire des *features* de celles-ci, dans le but de les classifier.

Pour terminer, nous avons présenté quatre métriques qui vont nous permettre de mesurer la qualité d'un classificateur. Il s'agit de la matrice de confusion, de l'*accuracy*, du *F1-score* et du *log loss* qui sont par défaut des métriques définies pour des problèmes de classification binaires, mais que nous pouvons étendre à des problèmes de classification multiclasses. Ces métriques sont destinées à comparer les résultats de plusieurs classifications ou détecter pour quel état de conscience une classification est la plus efficace.

Dans le prochain chapitre, nous allons présenter les résultats obtenus sur les valeurs réelles fournies par l'équipe du groupe *Coma Science Group* de l'Université de Liège. En effet, nous allons utiliser la méthode des *Extremely randomized trees* pour classer les états de conscience de différents patients pour lesquels nous disposons de données concernant les signaux BOLD dans différentes zones de leur cerveau. Les *features* utilisées pour effectuer les classifications de ce chapitre seront les informations spectrales extraites des séries temporelles via l'algorithme DMD exacte.

Chapitre 4

Classification des états de conscience

Ce chapitre est dédié à la présentation des résultats obtenus par les différentes classifications faites avec les signaux BOLD récoltés par le groupe *Coma Science Group* de l'Université de Liège.

Avant de passer à l'étape de la classification des états de conscience, nous devons d'abord sélectionner les données médicales que nous allons utiliser. En *Machine Learning*, la phase d'entraînement est primordiale car c'est elle qui va apprendre à la machine la tâche à effectuer, à savoir classifier les états de conscience. Il faut donc lui fournir le plus d'exemples possible, c'est-à-dire un ensemble d'entraînement de taille importante. Comme il n'y a que les signaux BOLD de 17 patients repris dans le premier type de données, le type *reduced*, ce dernier va être laissé de côté. Nous allons donc nous concentrer sur les deux autres types de données, les types *reduced GS* (RGS) et *reduced GS filtered* (RGSF), car il contiennent tous les deux les signaux BOLD de 18 patients.

Ensuite, nous devons sélectionner un modèle de classification. Le classificateur principal que nous allons considérer dans ce chapitre est le modèle *Extra-trees* et pour l'appliquer aux signaux BOLD des données retenues, nous allons utiliser la classe `ExtraTreesClassifier` du module `sklearn.ensemble` de `Python` (version 3.7.3). Nous avons choisi ce modèle car, comme énoncé dans le Chapitre 3, il est efficace et produit généralement de bons résultats pour la résolution de problèmes de classification.

Dans le but d'obtenir des résultats fiables, la méthode *K-folds* permettant de faire de la *cross-validation* sera appliquée lors des classifications effectuées dans ce chapitre. Celle-ci a été construite afin que les données d'un patient ne se retrouvent pas en même temps dans l'ensemble d'entraînement et dans l'ensemble de test. En effet, elle consiste à prendre les signaux BOLD des quatre états de conscience de 17 patients (sur 18) dans l'ensemble d'entraînement et à effectuer le test sur les quatre états de conscience du patient restant. Ce choix a été opéré car dans la réalité, si l'état de conscience d'un patient doit être prédit, nous ne disposerons généralement pas de données labellisées le concernant et nous ne pourrons pas entraîner un modèle d'apprentissage supervisé avec ces données. Ce processus est répété afin que chaque patient passe une fois dans l'ensemble de test. De plus, nous allons effectuer chaque classification dix fois et analyser les résultats moyens obtenus. En effet, nous aurons une idée plus réaliste de la qualité d'une classification en analysant ses résultats moyens après dix répétitions plutôt qu'avec les résultats d'une seule classification isolée.

Pour terminer, à la suite des différentes classifications, les résultats sont analysés grâce aux quatre métriques présentées dans le chapitre précédent. Il s'agit de l'*accuracy*, de la matrice de confusion, du *F1-score* et du *log loss*. Il faut noter que nous n'allons pas présenter les matrices de confusion moyennes pour deux raisons. Premièrement, si les moyennes sont calculées, les cases de la matrice de confusion moyenne ne contiendront plus des nombres entiers et il sera difficile d'interpréter les résultats. De plus, en présentant des matrices de confusion correspondant à une seule classification, nous pourrons vérifier si les performances de cette classification sont stables ou pas, en comparant les résultats de deux matrices de confusion avec le *F1-score* moyen obtenus après les dix répétitions.

Dans ce chapitre, nous allons d'abord présenter trois classifications effectuées avec les données spectrales directement récoltées via l'algorithme DMD exacte. Ensuite, nous allons analyser l'effet que peut avoir le paramètre `max_depth`, un paramètre qui régule la profondeur maximale des arbres de décision, sur la performance du modèle de classification `ExtraTreesClassifier`. Pour terminer, d'autres classificateurs seront testés afin de comparer leur performance à celle de la méthode des *Extra-trees*. Seuls les résultats les plus pertinents produits lors de la recherche seront présentés dans ce chapitre. Ceux-ci ont été obtenus via des codes implémentés en `Python` (version 3.7.3) et ces codes sont disponibles via l'URL donnée dans l'Annexe B.

4.1 Premières classifications

Dans cette section, nous allons présenter les trois premières classifications retenues lors de la recherche. Ces classifications sont effectuées à l'aide des informations spectrales récoltées après avoir passé les données de type RGSF et RGS en entrée de l'algorithme DMD exacte. Pour rappel, il s'agit dune approximation des 28 valeurs propres de l'opérateur de Koopman ainsi que des 28 modes de l'opérateur de Koopman correspondants. C'est dans ces données spectrales que les *features* sont sélectionnées afin d'alimenter le classificateur permettant de prédire les états de conscience des patients.

Classification sur base de tous les éléments spectraux

Pour effectuer la première expérience, nous avons considéré toutes les données spectrales obtenues par l'application de l'algorithme DMD exacte aux matrices de signaux BOLD. Il s'agit donc des 28 valeurs propres et de toutes les composantes des 28 modes de Koopman appartenant à $\in \mathbb{C}^{28}$, ce qui donne un total de $28 + 28 \times 28 = 812$ *features*. Les valeurs propres et les composantes des modes de Koopman étant des nombres complexes, nous avons pris le module de ces valeurs pour alimenter l'algorithme *Extra-trees* qui ne considère que des valeurs réelles.

Lors des dix répétitions de la classification, les différentes valeurs des métriques mesurant de la qualité du classificateur ont été retenues dans le Tableau A.1, le Tableau A.2 et le Tableau A.3 de l'Annexe A. Le Tableau 4.1 reprend les moyennes de ces métriques calculées à la suite des dix itérations.

Le type de données le plus traité, c'est-à-dire le type RGSF, obtient une *accuracy* moyenne de 0,35. Cela signifie que si nous prenons au hasard un état de conscience d'un patient, il a 35% de chances d'être correctement classifié par le classificateur. Ce score est légèrement plus élevé pour le deuxième type de données, le type RGS, qui obtient une *accuracy* moyenne de 0,37. Ces résultats ne sont pas exceptionnels mais ils démontrent que la méthode de classification construite a un peu plus de chances qu'une classification aléatoire de classifier une donnée correctement. En

Type de données	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
RGSF	0,35	0,16	0,27	0,33	0,32	1,33
RGS	0,37	0,20	0,21	0,37	0,36	1,34

TABLE 4.1 – Moyenne des métriques mesurant la qualité de la classification considérant tous les éléments spectraux obtenus via l'algorithme DMD exacte.

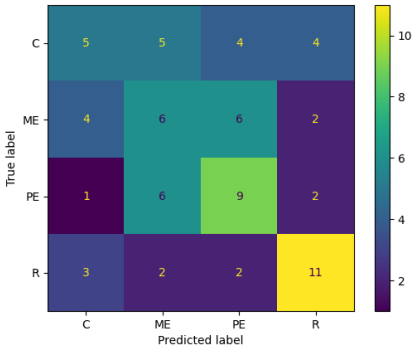

FIGURE 4.1 – Matrice de confusion pour l'itération n°4 de la première classification pour le type de données RGSF.

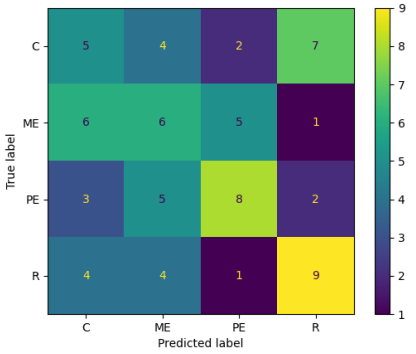

FIGURE 4.2 – Matrice de confusion pour l'itération n°9 de la première classification pour le type de données RGSF.

effet, puisque nous sommes dans un cas de classification avec quatre classes, une classification aléatoire classifie correctement une donnée prise au hasard avec une probabilité de 0,25. Même si ces résultats sont faibles, nous pourrions conclure que le classificateur est plus efficace sur les données de type RGS, mais le *log loss* donne une information contraire. En effet, cette métrique prend la valeur moyenne de 1,33 pour les données de type RGSF et de 1,34 pour les données de type RGS. Cela signifie que pour ce dernier type de données, le classificateur accorde une plus faible probabilité à la classe correcte.

Les matrices de confusion des deux itérations ayant obtenu les plus hauts scores d'*accuracy* pour le type de données RGSF sont affichées sur la Figure 4.1 et la Figure 4.2. Il s'agit des itérations n°4 et n°9 pour lesquelles l'*accuracy* vaut respectivement 0,43 et 0,39. Nous pouvons observer dans ces deux matrices que la classe la plus correctement détectée est l'état de conscience "en phase de réveil" et cela se confirme par le *F1-score* de cette classe qui vaut 0,51 lors de la quatrième itération et 0,38 lors de la neuvième itération. Cependant, en moyenne, le *F1-score* est légèrement meilleur pour détecter l'état de conscience "profondément endormi" car il vaut 0,33 contre 0,32 pour l'état de conscience "en phase de réveil". Nous pouvons donc conclure que cette classification est plus efficace pour détecter l'état de conscience correspondant à la classe "profondément endormi".

Concernant le type de données RGS, nous avons également affiché les deux matrices de confusion des deux itérations ayant obtenu les plus hauts scores d'*accuracy* sur la Figure 4.3 et sur

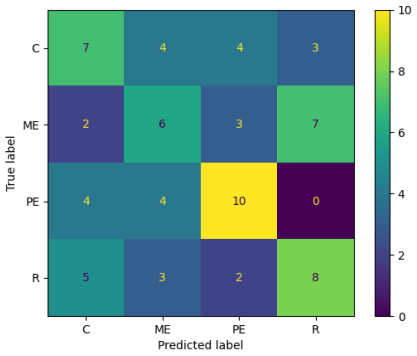

FIGURE 4.3 – Matrice de confusion pour l'itération n°2 de la première classification pour le type de données RGS.

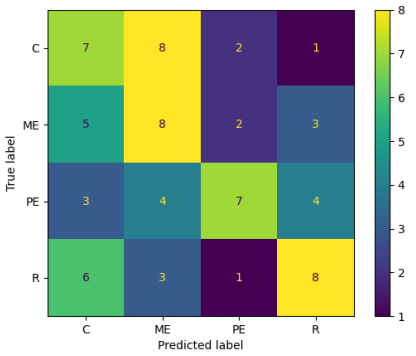

FIGURE 4.4 – Matrice de confusion pour l'itération n°7 de la première classification pour le type de données RGS.

la Figure 4.4. Sur ces matrices, nous pouvons observer que, contrairement au type de données RGSF, l'état de conscience le mieux détecté n'est pas toujours le même. Cependant, en considérant les *F1-score* moyens pour ce type de données, nous voyons qu'une tendance se dégage car les *F1-score* maximaux sont également obtenus pour les états de conscience "en phase de réveil" et "profondément endormi".

En conclusion, nous ne pouvons pas affirmer que cette classification est meilleure pour les données de type RGSF ou RGS car l'*accuracy* et le *log loss* sont très similaires pour les deux types de données. Cependant, grâce aux matrices de confusion et au *F1-score* moyen, nous pouvons conclure que ce modèle est plus efficace pour détecter les états de conscience "en phase de réveil" et "profondément endormi". Les scores d'*accuracy* moyens ne dépassant pas les 40%, nous allons essayer d'améliorer ceux-ci en sélectionnant les *features* d'une manière différente lors de la prochaine expérience.

Classification sur base d'une sélection des éléments spectraux

Lors de la première classification, 812 *features* avaient été considérées et ce nombre important de *features* peut conduire à un surapprentissage du modèle par rapport aux éléments de l'ensemble d'entraînement. Dans ce cas, le modèle n'arrive pas à généraliser les règles de classification et ne parvient pas à prédire correctement les classes des éléments dans l'ensemble de test. Nous allons donc uniquement sélectionner certaines *features* pour effectuer une deuxième classification. En particulier, nous allons nous intéresser aux composantes des modes dominants de Koopman. Les modes dominants sont ceux associés aux valeurs propres ayant les plus grandes parties réelles. En effet, nous savons que les états d'un système non-linéaire peuvent s'écrire comme

$$x_t = \sum_{k=1}^{K} b_k v_k \exp(\lambda_k t),$$

où K est le nombre de valeurs propres, v_k sont les modes de Koopman associés aux valeurs propres en temps continu λ_k et les coefficients b_k sont les coordonnées de la condition initiale [27]. Cette formule indique que les modes de Koopman ont de l'importance dans la détermination

Type de données	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
RGSF	0,23	0,19	0,13	0,32	0,09	1,43
RGS	0,33	0,25	0,19	0,43	0,15	1,34

TABLE 4.2 – Moyenne des métriques mesurant la qualité de la classification considérant une sélection des éléments spectraux obtenus via l'algorithme DMD exacte.

d'un futur état du système à n'importe quel temps t, mais que cette importance dépend du taux de (dé)croissance λ_k de l'exponentielle. Les modes qui ont le plus d'influence sont donc ceux associés aux valeurs propres ayant les plus grandes parties réelles.

Pour cette expérience, les *features* seront composées des modules des 28 valeurs propres obtenues après avoir fourni les signaux BOLD en entrée de l'algorithme DMD exacte, ainsi que des modules de toutes les composantes des trois modes de Koopman dominants. Au total, il y a aura donc $28+3\times28 = 112$ *features* pour chaque élément dans les ensembles d'entraînement et de test.

Les métriques mesurant la qualité du modèle *Extra-trees* pour cette sélection de *features* ont à nouveau été retenues durant les dix classifications et les tableaux récapitulatifs les reprenant sont le Tableau A.4, le Tableau A.5 et le Tableau A.6 de l'ANNEXE A. Les moyennes de ces métriques sont reprises dans le Tableau 4.2

Les *accuracy* moyennes valent 0,23 et 0,33 pour les types de données RGSF et RGS respectivement. Par rapport à la première expérience, nous pouvons donc observer une baisse de la qualité de la classification car l'*accuracy* moyenne a diminué pour les deux types de données. Le constat est le même pour le *log loss* du type de données RGSF qui atteint ici la valeur moyenne de 1,43. Le *log loss* moyen du type de données RGS, quant à lui, est identique à celui de la première classification et vaut 1,34. En termes d'incertitude, nous ne pouvons donc pas dire que cette classification est moins bonne que la précédente, même si l'*accuracy* moyenne est légèrement plus faible.

Les matrices de confusion associées aux itérations (n°2 et n°7 resp.) ayant eu les deux scores d'*accuracy* les plus élevés (0,31 et 0,29 resp.) pour le type de données RGSF sont disponibles sur la Figure 4.5 et la Figure 4.6. Si nous observons ces matrices, nous voyons que l'état de conscience "profondément endormi" semble être mieux prédit que les autres même si pour l'itération n°2, l'état de conscience "conscient" est tout aussi bien détecté. Cependant, si nous considérons le *F1-score* moyens des états de conscience correspondants, nous pouvons conclure que le classificateur est en général plus efficace pour l'état de conscience "profondément endormi" car son *F1-score* moyen vaut 0,32 contre 0,19 pour l'état de conscience "conscient". Cela montre l'importance de considérer le *F1-score* moyen pour compléter les informations données par les matrices de confusion et d'effectuer dix répétitions de la classification pour ne pas se laisser tromper par des résultats extrêmes issus d'une seule classification.

Sur la Figure 4.7 et Figure 4.8 sont affichées les matrices de confusion des deux itérations ayant obtenus à la fois les scores d'*accuracy* les plus grands et les scores de *log loss* les plus faibles pour le type de données RGS. Cette fois ci, une classe se démarque légèrement par rapport aux

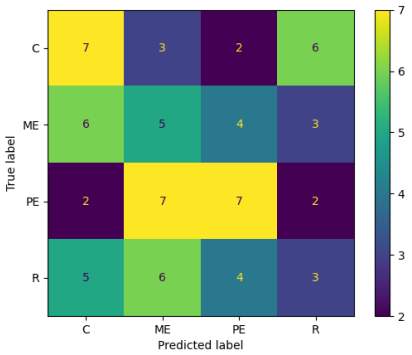

FIGURE 4.5 – Matrice de confusion pour l'itération n°2 de la deuxième classification pour le type de données RGSF.

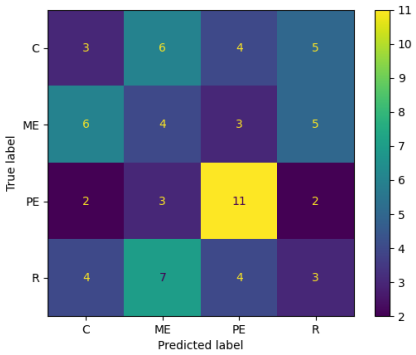

FIGURE 4.6 – Matrice de confusion pour l'itération n°7 de la deuxième classification pour le type de données RGSF.

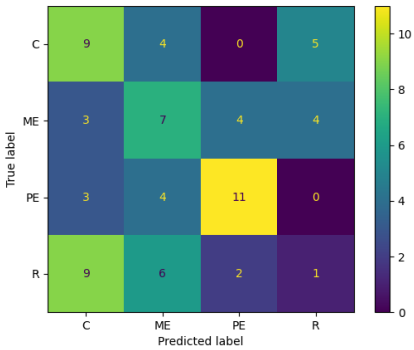

FIGURE 4.7 – Matrice de confusion pour l'itération n°5 de la deuxième classification pour le type de données RGS.

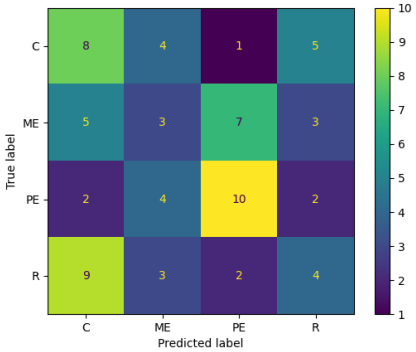

FIGURE 4.8 – Matrice de confusion pour l'itération n°7 de la deuxième classification pour le type de données RGS.

Type de données	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
RGSF	0,30	0,24	0,15	0,45	0,10	1,41
RGS	0,44	0,30	0,23	0,61	0,30	1,28

TABLE 4.3 – Moyenne des métriques mesurant la qualité de la classification considérant une moyenne des éléments spectraux fournis par l'algorithme DMD exacte.

autres et il s'agit de la classe "profondément endormi". L'état de conscience "conscient" semble également être bien prédit, mais nous pouvons voir sur les deux matrices de confusion que le classificateur à tendance à classifier des patients comme conscients alors qu'ils sont en réalité en phase de réveil, ce qui impacte négativement la précision de la classe "conscient". Cela est confirmé par le *F1-score* moyen de l'état de conscience "profondément endormi" affiché dans le Tableau 4.2 car nous voyons qu'il est de loin le plus élevé. En effet, il vaut 0,43 contre 0,25 pour l'état de conscience "conscient".

En conclusion, pour cette sélection de *features*, l'algorithme *Extra-trees* est plus efficace pour le type de données RGS car son *accuracy* moyenne est plus haute et son *log loss* moyen est plus bas que ceux du type de données RGSF. De plus, grâce aux matrices de confusion et au *F1-score* moyen des différentes classes, nous pouvons conclure que l'état de conscience "profondément endormi" est celui pour lequel le classificateur est le plus efficace.

Classification sur base d'une moyenne des éléments spectraux

La sélection de *features* choisie pour la troisième expérience a été inspirée du mémoire [8] et est le fruit de plusieurs essais qui ne seront pas détaillés dans ce mémoire. Elle consiste à prendre la moyenne des parties réelles des 28 valeurs propres obtenues à la sortie de l'algorithme DMD exacte ainsi que les modules des composantes du mode de Koopman dominant. Au total, cela représente donc $1 + 28 \times 1 = 29$ *features* par état de conscience et par patient qui vont permettre à l'algorithme *Extra-trees* de s'entraîner, mais aussi d'être testé.

Les valeurs des métriques mesurant la qualité de chacune des dix classifications effectuées l'une après l'autre ont été retenues et elles sont disponibles dans le Tableau A.7, le Tableau A.8 et le Tableau A.9 de l'ANNEXE A. Les moyennes de ces métriques calculées après les dix répétitions de la classification sont reprises dans le Tableau 4.3

Cette classification est présentée dans ce mémoire car c'est celle qui a permis d'atteindre la plus haute *accuracy* moyenne (0,44) et le plus faible *log loss* moyen (1,28) lors des premières expériences, ces valeurs ayant été obtenues avec les données de type RGS. Ces résultats démontrent que le classificateur a un peu moins d'une chance sur deux de classifier correctement une donnée prise au hasard. L'*accuracy* moyenne pour le type de données RGSF est de 0,30 tandis que le *log loss* moyen est de 1,41. Pour ce type de données, nous avons donc réussi à améliorer les valeurs de ces métriques par rapport à la deuxième expérience, mais pas par rapport à la première.

La Figure 4.9 et la Figure 4.10 représentent les matrices de confusion de la septième et de la sixième itération qui sont les itérations ayant obtenu le score d'*accuracy* le plus élevé, c'est-à-dire 0,33, pour le type de données RGSF. Pour le type de données RGS, ce sont les matrices

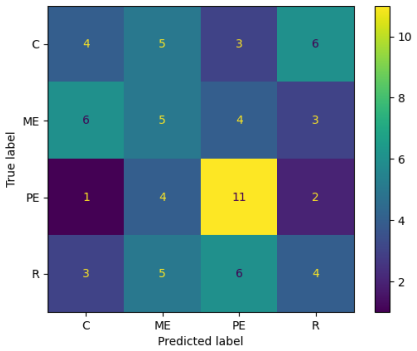

FIGURE 4.9 – Matrice de confusion pour l'itération n°7 de la troisième classification pour le type de données RGSF.

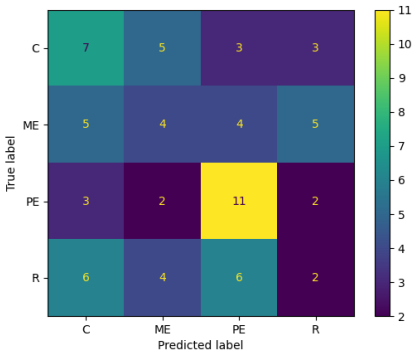

FIGURE 4.10 – Matrice de confusion pour l'itération n°6 de la troisième classification pour le type de données RGSF.

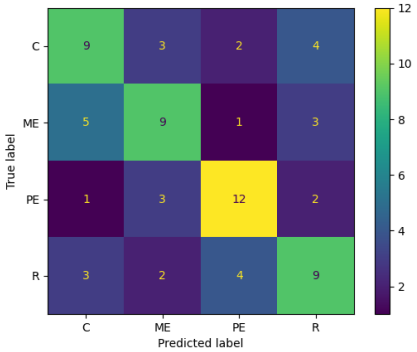

FIGURE 4.11 – Matrice de confusion pour l'itération n°3 de la troisième classification pour le type de données RGS.

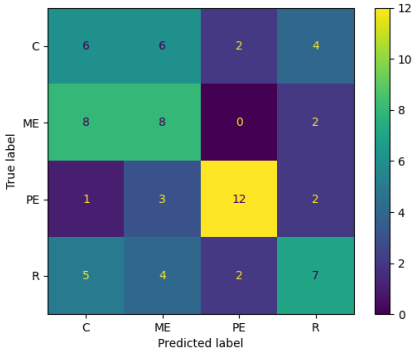

FIGURE 4.12 – Matrice de confusion pour l'itération n°4 de la troisième classification pour le type de données RGS.

de confusion des itérations n°3 et n°4 qui sont affichées sur la Figure 4.11 et la Figure 4.12 car lors de ces itérations, l'*accuracy* est maximale (0,54 et 0,46 resp.) pour cette expérience. Nous pouvons observer une similarité entre les quatre matrices de confusion car dans les quatre cas, l'état de conscience le mieux détecté est l'état "pronfondément endormi". Cela se confirme par le *F1-score* moyen de cet état qui vaut 0,45 pour le type de données RGSF, 0,61 pour le type de données RGS et qui est maximal par rapport aux *F1-score* moyens des autres états de conscience.

Pour conclure, cette troisième classification est la plus efficace pour les données de type RGS car elle nous a fourni l'*accuracy* moyenne la plus élevée, mais aussi le *log loss* moyen le plus faible des trois expériences effectuées jusqu'à présent. Malheureusement, cette sélection de *features* n'est pas optimale pour le type de données RGSF car le modèle utilisant ces *features* est moins efficace que le premier présenté dans cette section. Cependant, pour les deux types de données, nous pouvons admettre que cette classification détecte mieux l'état de conscience "profondément endormi" que les autres états grâce aux matrices de confusion et aux *F1-score* moyens de cette classe.

Conclusion des trois premières classifications

Les valeurs moyennes des métriques mesurant la qualité des différentes classifications effectuées dans cette section sont reprises dans le Tableau 4.1, Tableau 4.2 et Tableau 4.3. Dans ces tableaux, nous pouvons remarquer que la classification la plus efficace pour le type de données RGSF est la première classification dont le modèle est alimenté par toutes les données spectrales obtenues en sortie de l'algorithme DMD exacte, c'est-à-dire par 812 *features*. Pour cette classification, l'*accuracy* moyenne sur dix répétitions vaut 0,33 et le *log loss* moyen vaut 1,33. Cependant, de meilleures valeurs moyennes d'*accuracy* et de *log loss* sont obtenues grâce à la troisième classification sur le type de données RGS. En effet, dans ce cas, l'*accuracy* moyenne obtenue après dix répétitions de cette classification est maximale par rapport aux autres résultats de cette section et vaut 0,44 tandis que le *log loss* moyen obtenu est minimal et vaut 1,28. Lors de cette classification, 29 *features* ont été produites avec les éléments spectraux disponibles à la sortie de l'algorithme DMD exacte. Nous pouvons également observer que la classe qui semble être le mieux détectée lors de la majorité des classifications correspond à l'état de conscience "profondément endormi". Après avoir effectué de nombreuses classifications avec différentes sélections de *features*, nous avons constaté que celles comprenant la partie réelles des approximations des valeurs propres de Koopman fournissaient de meilleurs résultats de manière générale. Nous pouvons donc conclure que les parties réelles de ces valeurs propres sont suffisamment distinctes que pour permettre au modèle `ExtraTreesClassifier` de séparer les états de conscience, même si cette différence n'était pas visible à l'œil nu.

Pour la meilleure des trois classifications effectuées jusqu'à présent, un élément pris au hasard a 44% de chances d'être correctement classifié. Nous allons donc tenter d'améliorer ce résultat en faisant varier les valeurs de l'un des paramètres du classificateur *Extra-trees* puis en testant d'autres modèles de classification. Étant donné que le meilleur résultat observé dans cette section est obtenu lors de la dernière expérience, c'est-à-dire lorsque la classification considère une moyenne des éléments spectraux, nous allons uniquement considérer cette sélection de *features* pour le type de données RGS dans la suite de ce chapitre.

Type de données	*accuracy* moyenne	*F1-score* moyen				*log loss* moyen
		C	ME	PE	R	
RGS	0,46	0,33	0,24	0,57	0,37	1,29

TABLE 4.4 – Moyenne des métriques mesurant la qualité de la classification utilisant la classe `GridSearchCV` sur le paramètre `max_depth` selon les valeurs reprises dans le vecteur $(3, 4, 5, 8, 10, 20, 30)$.

4.2 Variation du paramètre régulant la profondeur des arbres du modèle *Extra-trees*

Pour appliquer l'algorithme *Extra-trees* sur les signaux BOLD, nous considérons la classe `ExtraTreesClassifier` du module `sklearn.ensemble` de `Python` (version 3.7.3). Dans la section 3.2, nous avons fixé les trois paramètres discutés dans l'article [12] aux valeurs recommandées par celui-ci, c'est-à-dire `n_estimators` $= 100$, `min_samples_split` $= 2$ et `max_features` $= \sqrt{N}$, où N est le nombre de *features* disponibles. La première modification que nous allons donc apporter au modèle en vue d'améliorer ses performances est la variation du quatrième paramètre du classificateur abordé dans cette section, le paramètre `max_depth`. Pour rappel, ce paramètre permet de réguler la profondeur, c'est-à-dire le nombre de nœuds, des arbres de décision qui composent la forêt d'arbres des *Extremely Randomized Trees*.

La valeur optimale du paramètre `max_depth` pour la classification des états de conscience ne peut pas être simplement devinée ou prise au hasard. Nous allons donc faire une recherche exhaustive sur des valeurs spécifiées pour ce paramètre grâce à la classe `GridSearchCV` du module `sklearn.model_selection` de `Python`. Le constructeur de cette classe prend en paramètre un modèle de classification ainsi que des vecteurs de valeurs choisies pour différents paramètres. Ensuite, durant la classification, elle permet de tester toutes les combinaisons de valeurs des paramètres possibles afin de trouver celles qui rendent le classificateur le plus performant.

Afin d'essayer d'améliorer les résultats obtenus lors de la section précédente, nous allons effectuer une nouvelle expérience durant laquelle une recherche exhaustive sur le paramètre `max_depth` du classificateur *Extra-trees* selon les valeurs reprises dans le vecteur (3,4,5,8,10,20,30) sera faite. Pour rappel, la classification est appliquée au type de données RGS et les *features* sélectionnées pour les données de l'ensemble d'entraînement et de test sont celles spécifiées lors de la troisième classification de la section précédente.

Lors de l'expérience, les valeurs prises par les différentes métriques mesurant la qualité de la classification ont été retenues et elles sont disponibles dans le Tableau A.10 en ANNEXE A. Les moyennes de ces métriques calculées à la suite des dix itérations sont reprises dans le Tableau 4.4. Grâce à cette technique de recherche, l'*accuracy* atteint une valeur moyenne de 0,46, ce qui est supérieur à tous les scores moyens d'*accuracy* produits jusqu'à présent. Au contraire, le *log loss* a légèrement augmenté comparé au plus petit score atteint dans la section précédente parce que sa valeur moyenne est de 1,29, indiquant que le modèle est légèrement plus incertain. De plus, l'exécution du code prend beaucoup plus de temps car la classe `GridSearchCV` teste toutes les valeurs proposées pour le paramètre `max_depth` à chaque itération de l'expérience, c'est-à-dire à chaque classification. Nous allons donc essayer de trouver une valeur fixe pour ce paramètre en essayant de minimiser une possible diminution de performance du classificateur.

Valeur de `max_depth`	3	4	5	8	10	20	30
Fréquences d'apparition	0,26	0,22	0,14	0,10	0,07	0,11	0,09

TABLE 4.5 – Fréquences d'apparition des valeurs du paramètre `max_depth` selon les valeurs reprises dans le vecteur $(3, 4, 5, 8, 10, 20, 30)$. Ces fréquences sont obtenues avec le modèle `ExtraTreesClassifier` utilisant la classe `GridSearchCV` lors des répétitions de la classification.

Valeur de `max_depth`	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
3	0,44	0,26	0,26	0,63	0,33	1,30
4	0,46	0,31	0,30	0,60	0,32	1,28

TABLE 4.6 – moyennes des métriques mesurant la qualité des classifications effectuées en fixant le paramètre `max_depth` à la valeur 3 ou 4, pour les données de type RGS.

Les fréquences d'apparition de chaque valeur proposée pour le paramètre `max_depth` sont affichées dans le Tableau 4.5. Dans ce tableau, nous pouvons observer que les valeurs les plus fréquemment choisies par la classe `GridSearchCV` sont les valeurs 3 et 4 car elles ont une fréquence d'apparition de 0,26 et 0,22 respectivement, et elles sont plus élevées que celles des autres valeurs. Après quelques classifications effectuées en fixant le paramètre `max_depth` à 3 ou à 4, nous pouvons conclure que la valeur 4 est la plus appropriée des deux. En effet, le Tableau 4.6 reprend les valeurs moyennes de l'*accuracy* et du *log loss* des classifications faites avec le paramètre `max_depth` fixé. Sur ce tableau, nous pouvons observer que les valeurs moyennes des deux métriques sont meilleures pour la valeur 4. L'*accuracy* moyenne obtenue pour cette valeur est identique à celle résultant de la recherche exhaustive sur le paramètre `max_depth` faite précédemment car elle vaut également 0,46. Le *log loss* moyen, quant à lui, a légèrement diminué pour atteindre la valeur de 1,28. Nous pouvons donc conclure que la valeur 4 pour le paramètre `max_depth` est optimale parmi celles proposées dans le vecteur $(3, 4, 5, 8, 10, 20, 30)$ car le classificateur muni de ce paramètre obtient la même *accuracy* moyenne, un *log loss* moyen plus faible et son exécution est plus rapide qu'avec la classe `GridSearchCV`. De plus, si nous comparons ces résultats avec ceux obtenus lors de la troisième classification de la section 4.1, nous constatons que la valeur du *log loss* moyen est identique (1,28), mais que celle de l'*accuracy* moyenne a augmenté car elle vaut maintenant 0,46. Pour toutes ces raisons, nous allons retenir la valeur de 4 pour le paramètre `max_depth` du modèle `ExtraTreesClassifier` lors de la plupart des classifications que nous allons effectuer par la suite.

Pour information, les valeurs des différentes métriques mesurant la qualité des dix classifications lorsque `max_depth` est fixé à 4 sont reprises dans le Tableau A.11 en ANNEXE A. La Figure 4.13 et la Figure 4.14 reprennent les matrices de confusion des deux itérations ayant obtenus les plus hauts scores d'*accuracy* qui sont de 0,53 et 0,49 respectivement. Dans ces matrices, nous voyons que la classe la mieux prédite est celle correspondant à l'état de conscience "profondément endormi" et cela se confirme par le *F1-score* moyen de cette classe qui vaut 0,60. Cette observation appuie le constat déjà fait précédemment : l'état de conscience "profondément endormi" est mieux classifié que les autres états de conscience.

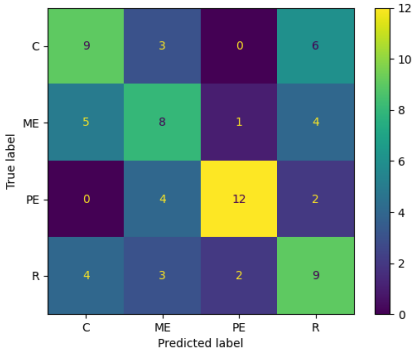

FIGURE 4.13 – Matrice de confusion pour l'itération n°4 de la classification prenant pour paramètre `max_depth` = 4 dans le modèle `ExtraTreesClassifier`.

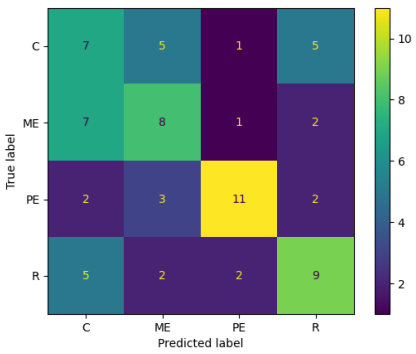

FIGURE 4.14 – Matrice de confusion pour l'itération n°8 de la classification prenant pour paramètre `max_depth` = 4 dans le modèle `ExtraTreesClassifier`.

Grâce aux recherches menées sur le paramètre `max_depth` régulant la profondeur des arbres de décision du classificateur `ExtraTreesClassifier`, nous avons réussi à améliorer légèrement la classification des états de conscience. Cependant, nous ne savons pas si le fait de fixer ce paramètre sera idéal pour toutes les prochaines classifications effectuées dans ce mémoire. C'est pourquoi, lors de chaque classification future, nous testerons si le paramètre `max_depth` fixé à 4 influence positivement ou non les résultats de celle-ci. Ces tests ne seront pas présentés dans ce mémoire, mais il sera indiqué à chaque début de section si le modèle `ExtraTreeClassifier` considère le paramètre `max_depth` fixé à 4 ou non.

Dans la prochaine section, nous allons comparer les performances d'autres modèles de classification avec celle du classificateur *Extra-trees* lorsqu'ils sont employés dans le but de classifier des états de conscience. Pour ce faire, d'autres classificateurs seront testés et les résultats produits seront comparés à ceux obtenus pour le modèle `ExtraTreesClassifier` dont le paramètre `max_depth` est fixé à 4.

4.3 Autres modèles de classification

Dans cette section, nous allons tester d'autres classificateurs afin de comparer leur performance avec celle du classificateur *Extra-trees*. En effet, le choix d'un modèle adapté pour résoudre un problème d'apprentissage supervisé est complexe. Il faut trouver le classificateur qui soit à la fois adapté aux données et au problème à résoudre. Pour savoir quel autre classificateur considérer, nous allons nous baser sur un organigramme qui a été conçu pour donner aux utilisateurs un guide approximatif des classificateurs à tester selon les données disponibles. Celui-ci est illustré à la Figure 4.15.

Au vu de la problématique exposée dans ce mémoire, nous allons nous concentrer sur les modèles de classification de l'organigramme. De plus, si nous suivons les instructions de cet organigramme concernant les données dont nous disposons, nous tombons également dans la catégorie de la classification. Étant donné que nous avons moins de 100 000 données, nous allons d'abord tester un modèle de classification appelé *Linear Support Vector Classification*. Le but de

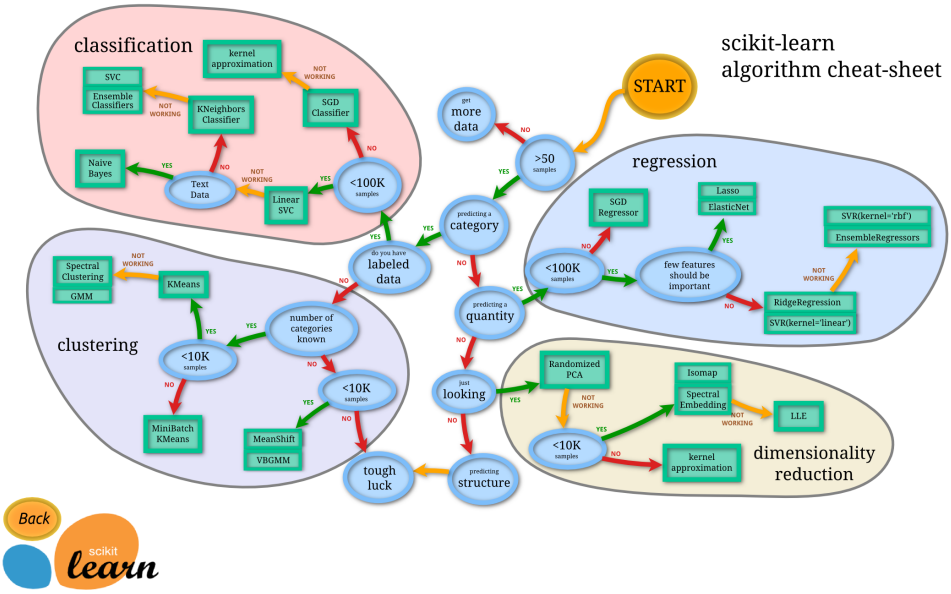

FIGURE 4.15 – Organigramme permettant de choisir un modèle d'apprentissage adapté aux données. Source : [17]

cette méthode de classification est de trouver un hyperplan qui sépare les données appartenant à différentes classes. Une donnée de test sera donc classée en fonction de sa place par rapport à l'hyperplan. Si nous sommes en deux dimensions, cet hyperplan sera une droite. Ensuite, l'organigramme nous guide vers la méthode des *K Neighbors Classifier*. Contrairement au *Linear Support Vector Classification*, ce modèle de classification ne cherche pas à établir un modèle général. Il va classer une donnée de test en fonction de la classe des K plus proches voisins de cette donnée de test, c'est-à-dire des K données qui sont les plus proches d'elle dans le plan. La classe attribuée à la donnée de test sera celle qui aura le plus de représentants parmi ses K plus proches voisins. Pour terminer, l'organigramme propose de tester la méthode appelée *Support Vector Classification* ainsi que des méthodes d'ensemble dont fait partie le modèle *Extra-trees*. Le *Support Vector Classification* est similaire au *Linear Support Vector Classification* car ces deux modèles cherchent à séparer les données grâce à des hyperplans mais la méthode de *Support Vector Classification* intervient lorsque les données sont linéairement non-séparable, c'est-à-dire qu'elles ne sont pas directement séparables par un hyperplan. Cette méthode va alors plonger les données dans un espace de dimension supérieure afin de les rendre linéairement séparables puis va trouver l'hyperplan sépare le mieux ces données. Concernant les méthodes d'ensemble, nous allons d'abord tester les *Random Forests*. Tout comme les *Extra-trees*, ce classificateur fait partie des méthodes d'ensemble basées sur les arbres de décision. Il y a deux différences majeures entre ces modèles. Premièrement, les *Random Forests* considèrent uniquement des échantillons *bootstraps*, c'est-à-dire des parties de l'ensemble d'entraînement, pour construire les arbres de décision permettant d'effectuer la classification. Deuxièmement, dans les nœuds des arbres de décision, le modèle des *Random Forests* va choisir le critère optimal pour séparer les données. Il va donc prendre en compte les *targets* des éléments à classifier. Enfin, nous allons tester une dernière méthode d'ensemble appelée *AdaBoost* ou *Adaptive Boosting*. Le principe de cette méthode repose sur de l'apprentissage séquentiel, c'est-à-dire qu'elle va construire un premier modèle de

Classificateur	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
Linear SVC	0,46	0,31	0,30	0,60	0,29	1,33
K Neighbors Classifier	0,33	0,36	0,20	0,40	0,06	6,71
SVC	0,38	0,29	0,21	0,43	0,20	1,41
Random Forests	0,41	0,20	0,24	0,66	0,30	1,25
AdaBoost	0,39	0,22	0,32	0,37	0,33	1,38

TABLE 4.7 – Moyenne des métriques mesurant la qualité de la classification en fonction du modèle de classification considéré, pour les données de type RGS.

classification sur l'ensemble des données d'entraînement pour ensuite construire un deuxième modèle rectifiant les erreurs commises par le premier. Ce processus est itéré jusqu'à ce qu'une condition d'arrêt soit rencontrée.

Afin de comparer la performance de ces classificateurs avec celle du modèle *Extra-trees*, nous avons sélectionné les *features* des données de la même manière pour toutes les expériences, c'est-à-dire selon la troisième classification de la section 4.1. Les valeurs moyennes des métriques évaluant la performance après dix itérations des différentes classifications sont reprises dans le Tableau 4.7. Dans ce tableau, nous pouvons observer que le classificateur ayant obtenu la plus haute *accuracy* moyenne mais aussi le plus faible *log loss* moyen est le modèle *Linear Support Vector Classification*. En effet, ces métriques atteignent les valeurs de 0,46 et 1,33 respectivement. Cela signifie que ce classificateur a un peu moins d'une chance sur deux de classifier correctement une donnée prise au hasard et qu'il est moins incertain que les autres modèles pour distinguer les données. Le Tableau 4.7 nous fournit une information supplémentaire. En effet, étant donné que les résultats obtenus avec le modèle *Linear Support Vector Classification* sont meilleurs que ceux obtenus avec le classificateur *Support Vector Classification*, nous pouvons conclure que les signaux BOLD, lorsque les *features* sont sélectionnées de la façon choisie dans cette section, sont plutôt linéairement séparables. Si nous comparons ces résultats avec ceux du meilleur modèle de classification testé jusqu'à présent, nous pouvons constater que les scores moyens d'*accuracy* sont identiques. Nous pourrions alors imaginer changer de modèle de classification pour le reste de ce mémoire mais si nous regardons le *log loss* moyen, nous constatons qu'il est plus élevé que celui obtenu lors de la classification effectuée dans la section 4.2, en considérant le modèle `ExtraTreesClassifier` et lorsque le paramètre `max_depth` est fixé à 4.

En conclusion, plusieurs modèles de classification ont été testés afin de comparer leur performance avec celle du classificateur *Extra-trees* retenu jusqu'à présent. Le modèle ayant obtenu les meilleurs résultats est le *Linear Support Vector Classification*, mais après avoir considéré son *log loss* moyen, nous avons constaté que ce modèle est plus incertain que celui des *Extra-trees*.

4.4 Conclusion

Dans ce chapitre, différentes méthodes de classification ont été expérimentées afin de prédire les états de conscience des patients. Les données utilisées à cette fin sont des séries temporelles reprenant des signaux BOLD récoltés par le groupe *Coma Science Group* de l'Université de Liège. Lors des trois premières classifications, l'algorithme DMD exacte était préalablement appliqué aux séries temporelles et les *features* utilisées pour la classification des données correspondaient à une sélection de certaines informations spectrales fournies par l'algorithme. Le classificateur utilisé pour ces tests était l'algorithme *Extra-trees*. Ensuite, le paramètre `max_depth` du classificateur `ExtraTreesClassifier` a été pris en compte et plusieurs tests ont été effectués pour déterminer son importance et sa valeur optimale. Pour terminer, la performance de ce classificateur a été comparée avec celles d'autres modèles de classification. Tout au long du chapitre, les métriques *accuracy*, *log loss* et *F1-score* ont été utilisées pour mesurer la qualité de chaque classification. Nous pouvons conclure que les meilleurs résultats sont obtenus lorsque les *features* sont sélectionnées de la façon expliquée dans la troisième classification de la section 4.1, c'est-à-dire lorsque la moyenne des parties réelles des valeurs propres de Koopman ainsi que les modules des composantes du mode dominant de Koopman sont considérés (rappelons que les éléments spectraux réellement utilisés sont des approximations fournies par l'algorithme de décomposition en modes dynamiques exacte). Pour obtenir ces résultats, nous avons utilisé le modèle de classification `ExtraTreesClassifier` avec le paramètre `max_depth` fixé à 4. Pour cette classification, **l'*accuracy* moyenne vaut 0,46 et le *log loss* moyen vaut 1,28. L'état de conscience le mieux prédit est celui correspondant à la classe "profondément endormi" avec un *F1-score* moyen de 0,60.**

Dans le prochain chapitre, nous proposons des pistes pour améliorer ces résultats grâce à d'autres manipulations que nous pouvons appliquer sur la sélection de *features*. Nous n'allons pas uniquement sélectionner d'autres *features* parmi les informations spectrales fournies par l'algorithme de décomposition en modes dynamiques exacte, mais également proposer de nouvelles méthodes permettant de générer les *features* des séries temporelles comportant les signaux BOLD. Pour terminer, nous allons évaluer la performance d'une classification qui combine plusieurs méthodes.

Chapitre 5

Amélioration de la classification

Ce chapitre est dédié à l'amélioration des meilleurs résultats obtenus dans le chapitre précédent. Pour ce faire, plusieurs techniques seront utilisées et celles-ci sont

- L'augmentation des données,
- Le modèle du sac de mots,
- La méthode Hankel-DMD,
- La combinaison de méthodes de classification.

Dans ce chapitre, nous allons générer les *features* des séries temporelles contenant les signaux BOLD de différentes façons, mais les classifications seront toutes effectuées avec le modèle de classification ayant produit les meilleurs résultats dans le chapitre précédent, c'est-à-dire le modèle `ExtraTreesClassifier`. Pour certaines classifications, nous fixerons son paramètre `max_depth` à 4 mais ce ne sera pas le cas dans toutes les classifications. En effet, il arrivera que la performance du modèle augmente si nous ne fixons pas ce paramètre au préalable, mais que nous laissons le classificateur le fixer par lui-même. Ce choix sera spécifié dans chaque section. De plus, une *cross-validation* identique à celle expliquée dans le Chapitre 4 sera appliquée et chaque classification sera itérée dix fois pour obtenir des résultats de performance moyens. La performance des modèles de classification sera à nouveau évaluée selon la matrice de confusion et les métriques *acurracy*, *F1-score* et *log loss*. Tout comme pour le chapitre précédent, nous ne présenterons pas les matrices de confusion moyennes mais celles correspondant à une seule classification.

Afin de ne pas surcharger le chapitre, seuls les résultats les plus pertinents produits lors de la recherche seront présentés. Ceux-ci ont été obtenus via des codes implémentés en `Python` (version 3.7.3) et ces codes sont disponibles via l'URL donnée dans l'Annexe B.

5.1 Augmentation des données

Dans cette section, nous allons augmenter le nombre de données disponibles pour alimenter le modèle de classification *Extra-trees*. En effet, nous devons sélectionner l'ensemble d'entraînement et l'ensemble de test parmi les séries temporelles correspondant aux quatre états de conscience de chacun des 18 patients, ce qui donne au total $18 \times 4 = 72$ données. Cependant, pour qu'une classification en *Machine Learning* soit efficace, il faut idéalement plusieurs centaines de données. Nous allons donc nous baser sur l'article [20] pour appliquer une méthode d'augmentation de

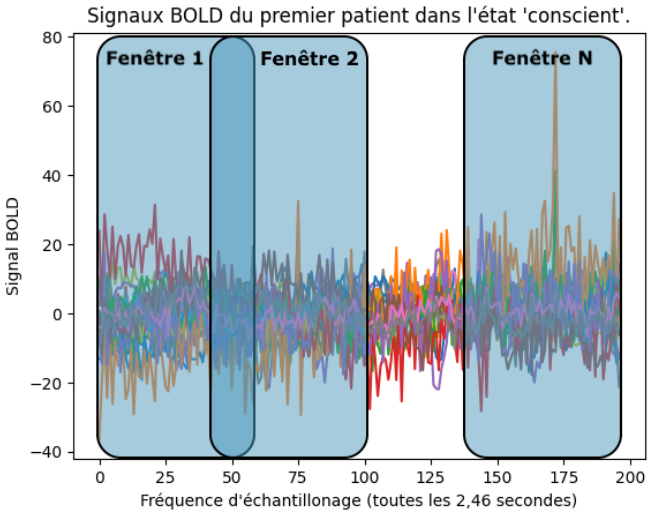

FIGURE 5.1 – Illustration de la méthode du *Window Slicing* sur les signaux BOLD de l'état "conscient" du premier patient, pour le type de données RGS. Inspiré de : [34]

données.

L'augmentation de données consiste à ajouter des copies modifiées ou non des données. Les données fournies par le groupe *Coma Science Group* étant des séries temporelles, nous allons leur appliquer une méthode agissant sur l'axe du temps et appelée le *Window Slicing*. Cette méthode consiste à découper la série temporelle d'origine en N différentes sous-séries de taille choisie W et ce découpage se fait à des moments t différents. En d'autres termes, N fenêtres de taille W de la série temporelle de départ sont extraites à des moments t et ces fenêtres sont considérées comme de nouvelles données ayant la même *target* que la série temporelle entière. Ce processus est illustré sur la Figure 5.1.

Pour appliquer cette technique aux signaux BOLD, nous devons choisir plusieurs paramètres. Tout d'abord, la taille des fenêtres W doit être fixée en respectant la contrainte $W > 28$ pour que les fenêtres possèdent plus de points temporels (colonnes) que de régions du cerveau dans lesquelles les signaux BOLD ont été récoltés (lignes) afin que ces fenêtres soient acceptées en entrée de l'algorithme DMD. Ensuite, nous allons choisir les paramètres t et N de sorte que tous les états de la série temporelle se retrouvent au moins une fois dans une fenêtre. Pour ce faire, nous allons construire un vecteur contenant des index indiquant les moments auxquels extraire les fenêtres. Le premier élément de ce vecteur sera 0, le dernier sera $m - W$, où m est le nombre d'états total contenu dans la série temporelle de départ, et les autres $N - 2$ index seront choisis dans l'intervalle $]0; m - W[$ selon une distribution uniforme grâce à la fonction `randint` de `Python`. Le nombre N de fenêtres sélectionnées doit donc être suffisamment grand pour que les fenêtres se chevauchent et couvrent l'entiereté de la série temporelle de départ. Plusieurs combinaisons de paramètres (N, W) ont été testées dans la recherche liée à ce mémoire et seuls les résultats de la combinaison optimale vont être présentés.

Dans ce mémoire, les valeurs retenues pour les paramètres N et W sont 8 et 140 respective-

Classification	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
Première classification	0,43	0,30	0,38	0,66	0,21	1,27
Deuxième classification	0,44	0,30	0,30	0,65	0,19	1,20

TABLE 5.1 – Moyenne des métriques mesurant la qualité des classifications considérant 576 données en entrée grâce à une technique d'augmentation des données appelée *Window Slicing*, pour les données de type RGS. Lors de la première classification, les sous-séries sont considérées comme indépendantes les unes des autres et lors de la deuxième classification, les séries temporelles de départ sont reformées. Les *targets* attribuées à celles-ci sont les classes prédites le plus grand nombre de fois par le classificateur pour les sous-séries extraites de ces séries temporelles.

ment. Nous avons donc $18 \times 4 \times 8 = 576$ données au lieu de $18 \times 4 = 72$ pour alimenter notre classification. La séparation de celles-ci en ensembles d'entraînement et de test est similaire à celle utilisée jusqu'à présent. En effet, nous appliquons toujours une *cross-validation* dans laquelle toutes les données concernant un même patient sont utilisées dans l'ensemble de test et les données restantes servent d'entraînement pour le classificateur. Cette fois, nous avons donc $1 \times 4 \times 8 = 32$ données dans l'ensemble de test et $17 \times 4 \times 8 = 544$ données dans l'ensemble d'entraînement. La *cross-validation* est à nouveau répétée afin que la phase de test s'effectue une fois sur chaque patient. Le modèle de classification utilisé est toujours le modèle `ExtraTreesClassifier` et pour cette section, nous allons lui attribuer le paramètre `max_depth` fixé à 4.

Tout comme pour les autres expériences, les valeurs des métriques mesurant la qualité des dix classification ont été retenues dans le Tableau A.12 de l'ANNEXE A, et les moyennes de ces métriques sont reprises dans la première ligne du Tableau 5.1. Dans ce tableau, nous pouvons voir que l'*accuracy* moyenne obtenue vaut 0,43 et que le *log loss* moyen obtenu vaut 1,27. Par rapport aux meilleurs résultats produits dans le chapitre précédent, seule la valeur du *log loss* a donc diminué, mais de très peu. Cela peut s'expliquer par le fait que le modèle se base sur plus de données pour effectuer la classification et que, par conséquent, il est moins incertain. Malgré tout, cette technique d'augmentation des données n'a donc pas amélioré de façon importante la performance du classificateur. Nous allons donc expérimenter une seconde classification durant laquelle les sous-séries temporelles ne sont pas traitées individuellement.

Dans l'expérience précédente, l'algorithme *Extra-trees* prenait en entrée 576 données puis ressortait 576 *targets* et elles étaient toutes considérées comme indépendantes les unes des autres. Cette fois, nous allons reformer les séries temporelles de base à la sortie de l'algorithme. La classe qui sera alors attribuée à une série temporelle sera celle que le classificateur aura prédit le plus grand nombre de fois pour les fenêtres extraites de cette série temporelle. Les valeurs des métriques mesurant la performance des dix classifications sont reprises dans le Tableau A.13 dans l'ANNEXE A et leur moyenne sont affichées dans la deuxième ligne du Tableau 5.1. Par rapport à la méthode la plus performante obtenue dans le Chapitre 4 , nous pouvons à nouveau observer que le *log loss* moyen est amélioré par cette technique mais pas l'*accuracy* moyenne. Ce modèle est donc moins incertain, mais il ne classifie pas mieux les données. Par contre, les valeurs des deux métriques ont été améliorées par rapport à l'expérience dans laquelle les sous-séries étaient

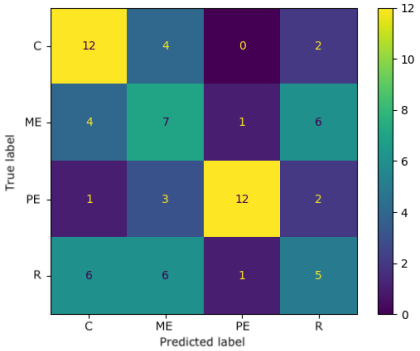

FIGURE 5.2 – Matrice de confusion pour l'itération n°9 de la classification utilisant une méthode d'augmentation des données appelée *Window Slicing*. Lors de cette classification, chaque série temporelle de départ est reformée et la classe attribuée à celle-ci correspond à la majorité des classes des sous-séries extraites.

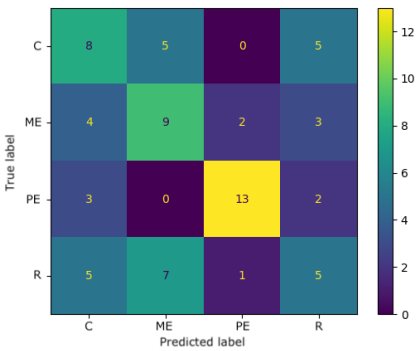

FIGURE 5.3 – Matrice de confusion pour l'itération n°6 de la classification utilisant une méthode d'augmentation des données appelée *Window Slicing*. Lors de cette classification, chaque série temporelle de départ est reformée et la classe attribuée à celle-ci correspond à la majorité des classes des sous-séries extraites.

considérées individuellement. Afin de savoir sur quelle classe cette nouvelle manière de classifier les données est la plus efficace, nous allons analyser les matrices de confusion des deux itérations ayant obtenues les plus hauts scores d'*accuracy*. Celles-ci sont affichées sur la Figure 5.2 et Figure 5.3. Nous pouvons y constater que la classe la mieux classifiée est celle correspondant à l'état "profondément endormi" et cela se confirme avec son *F1-score* moyen qui est plus élevé que celui des autres classes. Nous pouvons donc conclure que même lorsque nous dupliquons les données, l'état "profondément endormi" reste celui qui est le plus facile à distinguer pour toutes les méthodes de classification testées jusqu'à présent.

Dans la prochaine section, nous essayons d'améliorer la classification des états de conscience en utilisant une technique permettant de générer les *features* d'une façon moins conventionnelle. Celle-ci est appelée le modèle du sac de mots et pour l'appliquer, les approximations des valeurs propres et vecteurs propres obtenus via l'algorithme de décomposition en modes dynamiques exacte subissent d'autres transformations avant d'être considérées comme des *features* alimentant un classificateur.

5.2 Modèle du sac de mots

À l'origine, le modèle du sac de mots fournit une nouvelle manière de générer les *features* de données textuelles afin de les classifier. Cependant, grâce à la combinaison avec des méthodes de décomposition en modes dynamiques appliquées à des séries temporelles que propose la référence [34], nous pouvons appliquer cette méthode du sac de mots aux signaux BOLD des patients, dans le but de classifier leurs états de conscience. Nous allons donc tester cette manière de générer les *features* sur les deux types de données fournies par le groupe *Coma Science Group*, c'est-à-dire le type de données RGSF et RGS. Les classifications de cette section seront effectuées avec le même modèle que les précédentes, c'est à dire le classificateur `ExtraTreeClassifier` avec le paramètre

`max_depth` fixé à 4.

Les explications concernant l'application de la méthode du sac de mots à des séries temporelles sont disponibles dans la section 3.3. Pour générer le *codebook* de la méthode du sac de mots, il faut d'abord extraire des segments locaux à partir des séries temporelles de départ. Pour cela, nous allons appliquer le même processus que celui choisi pour l'augmentation de données, c'est-à-dire un *Window Slicing* avec $N = 8$ le nombres de fenêtres temporelles extraites, $W = 140$ la taille de celles-ci et t un vecteur de moments auxquels elles sont prélevées. Tout comme pour l'augmentation de données, le premier élément du vecteur sera 0, le dernier sera $m - W$, où m est le nombre total d'états contenus dans les séries temporelles, et les $N - 2$ valeurs de t seront sélectionnés de manière uniforme entre $]0, m - W[$. Ensuite, des informations spectrales sont déduites des segments temporels grâce à l'algorithme DMD exacte. Ces informations vont représenter les segments locaux, comme des mots pourraient représenter un texte, et elles permettront d'appliquer un *clustering* sur ceux-ci pour générer les *codewords* de la méthode du sac de mots. Ces informations spectrales sont donc données en entrée de l'algorithme k-médoïdes qui fournit les *codewords*, ceux-ci étant les centres des *clusters* générés par l'algorithme, qui permettent de former le *codebook*. Nous devons donc commencer par déterminer le nombre de *clusters* maximisant la performance du *clustering* et/ou de la classification des états de conscience. En effet, ce nombre de *clusters* déterminera le nombre de *codewords* dans le *codebook*, c'est-à-dire le nombre de *features* alimentant la classification. La valeur optimisant le *clustering* n'est peut-être donc pas celle maximisant la performance de la classification des états de conscience. Pour déterminer le nombre de *clusters* idéal pour le *clustering*, nous allons utiliser la méthode du coude qui optimise la distance intra-classe. Cette distance aussi appelée WCSS (*Within-Cluster Sum-of-Squares*) est la somme sur tous les *clusters* des distances (euclidiennes ou non) au carré entre chaque centroïde d'un *cluster* et les différentes observations inclues dans ce *cluster*, c'est-à-dire

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{x_i \in C_k} d(x_i, z_k)^2,$$

avec K le nombre total de *clusters*, C_k un *cluster*, x_i une observation de C_k et z_k le centroïde de C_k, $k = 1, ..., K$. Cette distance est la fonction objectif de l'algorithme k-médoïdes.

Si un seul *cluster* est généré, la méthode de *clustering* ne produira qu'un seul centroïde et la plupart des données se trouveront loin de celui-ci, entraînant une distance WCSS assez élevée. Au contraire, si le nombre de *clusters* est égal au nombre de données, chaque centroïde sera une donnée et la distance WCSS sera nulle. Ces deux cas extrêmes sont évidemment à éviter si nous voulons que le *clustering* soit efficace pour partitionner les observations. Une autre raison d'éviter ces cas extrêmes est que, pour rappel, le nombre de *clusters* générés par l'algorithme k-médoïdes est égal au nombre de *features* considérées par la classification. Une classification avec une seule *feature* n'est pas envisageable et une classification avec autant de *features* que de données risque de causer un surapprentissage du classificateur par rapport aux données d'entraînement. Pour rappel, ce surapprentissage est à bannir car dans ce cas, le modèle n'arrive pas à généraliser les règles de classification et ne parvient pas à prédire correctement les *targets* des éléments dans l'ensemble de test. Il faut donc trouver un juste milieu et c'est ce que permet la méthode du coude. Pour l'appliquer, il faut afficher la distance WCSS en fonction du nombre k de *clusters* générés par la méthode de *clustering* et essayer de détecter visuellement le moment où la courbe s'infléchit, c'est-à-dire lorsqu'elle forme un coude. Un exemple de cette courbe est affiché sur la Figure 5.4. Pour cet exemple, la méthode du coude nous indique que le nombre de *clusters* optimal est 3.

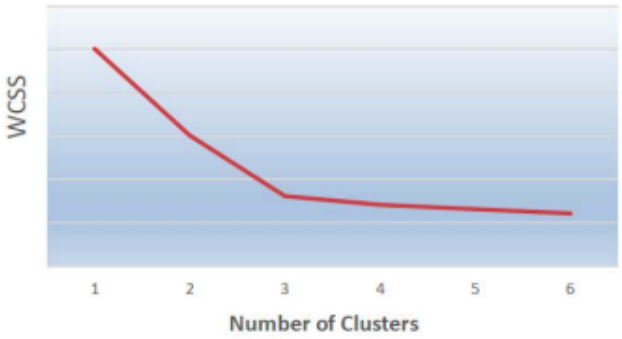

FIGURE 5.4 – Exemple d'un graphique représentant la distance WCSS en fonction du nombre de *clusters*. La méthode du coude nous indique que le nombre de *clusters* optimal est 3. Source : [6]

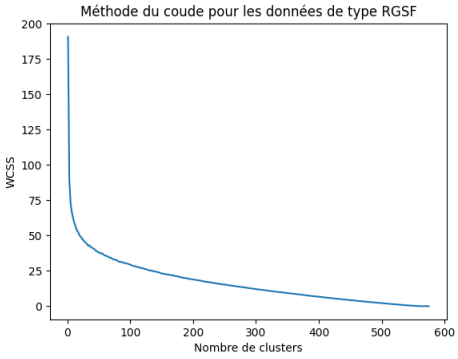

FIGURE 5.5 – Distance WCSS en fonction du nombre de *clusters* $k \in [1, 576]$ par pas de 2, pour le type de données RGSF. La méthode du coude indique que le nombre de *clusters* optimal de l'algorithme k-médoïdes se trouve entre 1 et 100.

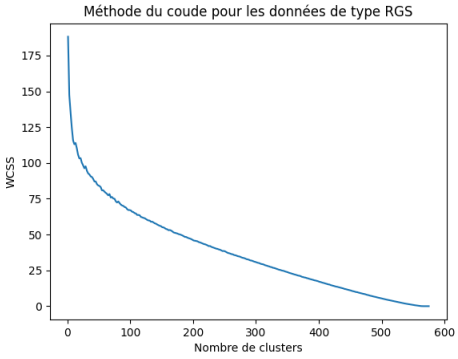

FIGURE 5.6 – Distance WCSS en fonction du nombre de *clusters* $k \in [1, 576]$ par pas de 2, pour le type de données RGS. La méthode du coude indique que le nombre de *clusters* optimal de l'algorithme k-médoïdes se trouve entre 1 et 100.

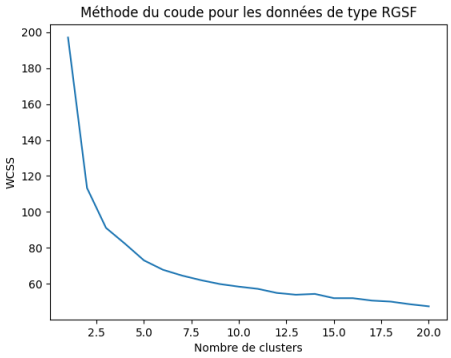

FIGURE 5.7 – Distance WCSS en fonction du nombre de *clusters* $k \in \{1, ...20\}$, pour le type de données RGSF. La méthode du coude indique que le nombre de *clusters* optimal de l'algorithme k-médoïdes est 3 ou 5.

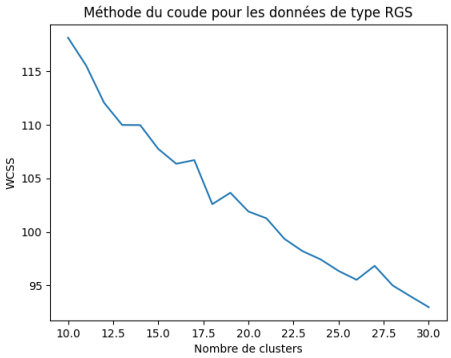

FIGURE 5.8 – Distance WCSS en fonction du nombre de *clusters* $k \in \{10, ...30\}$, pour le type de données RGS. Dans ce cas, la méthode du coude est difficilement applicable pour trouver le nombre de *clusters* optimal de l'algorithme k-médoïdes.

Sur la Figure 5.5 et la Figure 5.6, la distance WCSS est affichée en fonction du nombre k de *clusters* générés par l'algorithme k-médoïdes lors de la génération du *codebook*, pour les données de type RGSF et RGS respectivement. Les valeurs de k testées parcourent l'intervalle $[1, 576]$ par pas de deux. En effet, lors de la première étape de la méthode du sac de mots, huit fenêtres sont extraites de chacune des 72 séries temporelles de départ et par conséquent, l'algorithme k-médoïdes doit partitionner $8 \times 72 = 576$ éléments. Sur ces deux figures, la méthode du coude indique que le nombre optimal de *clusters* se trouve entre 0 et 100. En réduisant petit à petit l'intervalle de valeurs prises par k et en recalculant la distance WCSS pour chacune de ces valeurs afin de déterminer lesquelles constituent les coudes des deux courbes, nous sommes arrivés à l'intervalle $[1, 20]$ pour les données de type RGSF et $[10, 30]$ pour celles de type RGS. Les distances WCSS pour ces intervalles sont affichées sur la Figure 5.7 et la Figure 5.8. Sur la première figure, la méthode du coude indique clairement que le nombre de *clusters* optimal est 3 ou 5 pour l'algorithme k-médoïdes. Nous allons donc effectuer deux classifications, l'une considérant $k = 3$ et l'autre $k = 5$. Pour $k = 3$, l'*accuracy* moyenne calculée sur dix répétitions de la classification vaut 0,22 et le *log loss* moyen vaut 1,60 tandis que pour $k = 5$, les valeurs obtenues sont 0,20 et 1,49 respectivement. Cela est du au fait que le nombre de *clusters* est égal au nombre de *features* disponibles pour chaque série temporelle. Il n'est donc pas étonnant que le classificateur ne sache pas efficacement distinguer les différents états de conscience en ayant à sa disposition seulement trois ou cinq informations sur chaque série temporelle représentant ces états. Ces résultats étant bien inférieurs à ceux obtenus lors du chapitre précédent, nous n'allons pas nous en contenter. Concernant la courbe affichée sur la Figure 5.8 pour les données de type RGS, nous constatons que la méthode du coude y est difficilement applicable car cette courbe ne prend pas la forme d'un coude. Nous concluons que le nombre de *clusters* optimisant la performance de l'algorithme k-médoïdes n'est pas idéal pour la classification des états de conscience.

C'est pourquoi, d'autres valeurs de k ont été testées dans le but d'améliorer les résultats de la classification des états de conscience, que ce soit pour le type de données RGSF ou RGS. Plus particulièrement, il s'agit des valeurs 10, 20, 30, 50, 100, 150 et 250, et elles sont inspirées

Nombre de *clusters*	***accuracy* moyenne**		***log loss* moyen**	
	RGSF	**RGS**	**RGSF**	**RGS**
10	0,21	0,33	1,45	1,31
20	0,28	0,36	1,42	1,33
30	0,21	0,42	1,44	1,35
50	0,21	0,42	1,45	1,34
75	0,28	0,34	1,44	1,35
100	0,22	0,44	1,42	1,31
150	0,29	0,24	1,37	1,41
250	0,17	0,35	1,44	1,36

TABLE 5.2 – Tableau récapitulatif reprenant l'*accuracy* moyenne et le *log loss* moyen en fonction du nombre de *clusters* considérés par l'algorithme k-médoïdes pour les type de données RGSF et RGS.

Type de données	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
RGSF	0,29	0,23	0,15	0,27	0,25	1,37
RGS	0,44	0,32	0,42	0,43	0,27	1,31

TABLE 5.3 – Moyenne des métriques mesurant la qualité des classifications utilisant le modèle du sac de mots pour générer les *features*. Le nombre de *clusters* générés par l'algorithme k-médoïdes est $k = 150$ et $k = 100$ pour les données de type RGSF et RGS respectivement.

de la référence [34]. Les scores d'*accuracy* moyenne de *log loss* moyen obtenus obtenus après dix itérations de la classification pour les données de type RGSF et RGS sont affichés sur le Tableau 5.2. Pour les données de type RGS, le nombre de *clusters* maximisant la performance de la classification des états de conscience en terme d'*accuracy* et de *log loss* est $k = 100$. Pour cette valeur, l'*accuracy* indique qu'il y a 44% de chances qu'une donnée prise au hasard soit bien classifiée et le *log loss* vaut 1,33. Concernant les données de type RGSF, la classification atteint ses meilleurs résultats pour $k = 150$, ce qui est très éloigné du nombre de *clusters* optimal pour la méthode k-médoïdes. Cela peut à nouveau s'expliquer par le fait qu'un classificateur parvient mieux à classifier les données avec 150 *features* qu'avec trois ou cinq *features*. Pour ce type de données et pour cette valeur de k, l'*accuracy* moyenne vaut 0,29 et le *log loss* moyen vaut 1,37.

Les valeurs des métriques *accuracy*, *F1-score* et *log loss* de ces deux meilleurs résultats ont été retenues durant les 10 répétitions des deux classifications, l'une considérant les données de type RGSF et l'autre celles de type RGS. Elles sont reprises dans le Tableau A.14, le Tableau A.15 et le Tableau A.16 de l'ANNEXE A et la moyenne de ces métriques sont affichées dans le Tableau 5.3. Les matrices de confusion des classifications ayant obtenus les deux plus hauts scores d'*accuracy* lors des dix répétitions sont affichées sur la Figure 5.9 et la Figure 5.10 pour les données de

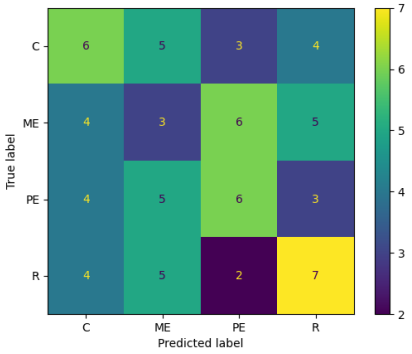

FIGURE 5.9 – Matrice de confusion pour l'itération n°3 de la classification dont les *features* sont générées par le modèle du sac de mots pour les données RGSF. Le nombre de *clusters* générés par l'algorithme k-médoïdes est $k = 150$.

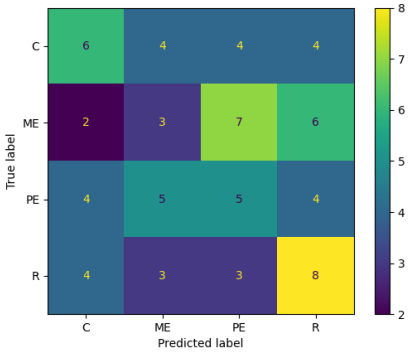

FIGURE 5.10 – Matrice de confusion pour l'itération n°1 de la classification dont les *features* sont générées par le modèle du sac de mots pour les données RGSF. Le nombre de *clusters* générés par l'algorithme k-médoïdes est $k = 150$.

type RGSF, et celles pour les données RGS sont exposées sur la Figure 5.11 et la Figure 5.12. En se basant uniquement sur les deux premières matrices de confusion, nous pourrions affirmer que cette fois, c'est l'état "en phase de réveil" qui est le mieux classifié par le modèle. Pourtant, si nous regardons les *F1-score* moyens des quatre classes, nous pouvons remarquer c'est l'état de conscience "profondément endormi" qui obtient un score légèrement plus haut que celui des autres classes. Cela est du au fait que le classificateur attribue à la classe "en phase de réveil" beaucoup de données qui ne correspondent pas réellement à cet état. Ces mauvaises prédictions diminuent alors le score de précision de la classe "en phase de réveil", entraînant également une baisse de son *F1-score*. Concernant les deux matrices de confusion correspondant aux données de type RGS, nous pouvons constater qu'en général, ce sont les états "moyennement endormi" et "profondément endormi" qui sont le mieux classifiés. En effet, leur *F1-score* moyen est plus élevé que ceux des deux autres classes. Sur la Figure 5.11, nous pouvons également remarquer que le modèle a été plus performant pour la classe "conscient". En effet, son *F1-score* pour cette itération est de 0,45 et il dépasse ceux des classes "moyennement endormi" et "profondément endormi", mais cette itération était une exception car le *F1-score* moyen de cette classe est de 0,32. Cet exemple montre bien l'intérêt d'analyser les valeurs moyennes des métriques mesurant la qualité d'une classification après plusieurs répétitions de la classification et non pas uniquement après une classification isolée.

En conclusion, il est intéressant de remarquer que malgré la manière de générer les *features* à l'aide de la méthode du sac de mots qui est une manière bien distincte de celles testées jusqu'à présent, les classifications sont à nouveau plus performantes pour l'état de conscience "profondément endormi", que ce soit pour les données de type RGSF ou RGS. Cependant, peu importe le type de données considéré, les scores moyens d'*accuracy* et de *log loss* obtenus en utilisant cette méthode sont moins bons que ceux résultants des classifications analysées dans le chapitre précédent. Par conséquent, nous n'allons pas la retenir dans la suite de ce mémoire.

Dans la section suivante, nous allons tester une autre manière de générer les *features* des

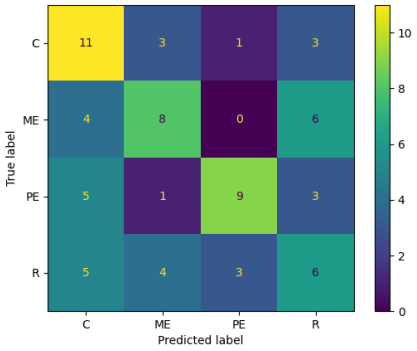

FIGURE 5.11 – Matrice de confusion pour l'itération n°3 de la classification dont les *features* sont générées par le modèle du sac de mots pour les données RGS. Le nombre de *clusters* générés par l'algorithme k-médoïdes est $k = 100$.

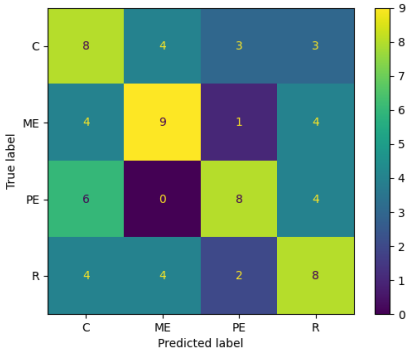

FIGURE 5.12 – Matrice de confusion pour l'itération n°6 de la classification dont les *features* sont générées par le modèle du sac de mots pour les données RGS. Le nombre de *clusters* générés par l'algorithme k-médoïdes est $k = 100$.

séries temporelles pour tenter d'améliorer les résultats obtenus jusqu'à présent. Cette nouvelle piste est basée sur la matrice dite de Hankel et la décomposition en modes dynamiques appelée Hankel-DMD.

5.3 Hankel-DMD

Le but de cette section est d'analyser la performance de la classification lorsque nous générons les *features* à l'aide de l'algorithme de décomposition en modes dynamiques de Hankel défini dans la sous-section 2.2.2.

Afin d'appliquer l'algorithme de décomposition en modes dynamiques de Hankel aux séries temporelles reprenant les signaux BOLD, nous devons d'abord organiser celles-ci en matrice de type Hankel. Pour cela, nous devons déterminer des valeurs de m_c et m_r qui détermineront le nombre de lignes et de colonnes de chaque matrice de Hankel. Pour rappel, ces paramètres sont liés par la relation $m_c + m_r - 1 = m$, où m est le nombre connu d'états total composant les séries temporelles. C'est pourquoi, nous allons uniquement faire varier le paramètre m_c et déterminer automatiquement la valeur de m_r grâce à la relation $m_r = m + 1 - m_c$. Ensuite, nous allons appliquer l'algorithme de décomposition en modes dynamiques de Hankel aux différentes matrices de Hankel afin d'obtenir les valeurs propres et les vecteurs propres de la matrice de Koopman calculée dans cet algorithme. Plus particulièrement, nous allons conserver les valeurs propres de cet opérateur et les utiliser en tant que *features* des séries temporelles lors de la classification. Par conséquent, le nombre de *features* sera égal au nombre de valeurs propres de la matrice de Koopman obtenue via l'algorithme Hankel-DMD, c'est-à-dire $m_c - 1$. Nous pouvons donc nous attendre à ce que le choix de la valeur du paramètre m_c influence fortement la performance de la classification.

Concernant le classificateur *Extra-trees*, la décision de ne pas fixer le paramètre `max_depth` à 4 a été prise car cette fois, après plusieurs tests préliminaires, nous nous sommes rendus compte

Valeur de m_c	***accuracy***		***log loss***	
	RGSF	**RGS**	**RGSF**	**RGS**
5	0,30	0,39	4,44	2,27
8	0,42	0,32	1,35	1,57
10	0,52	0,27	1,14	1,48
15	0,39	0,39	1,33	1,39
20	0,31	0,37	1,36	1,33
30	0,33	0,43	1,36	1,29
50	0,38	0,34	1,37	1,32

TABLE 5.4 – Tableau récapitulatif reprenant l'*accuracy* moyenne et le *log loss* moyen en fonction de la valeur du paramètre m_c pour les données de type RGSF et RGS. Ces scores ont été obtenus après dix répétitions de l'algorithme `ExtraTreesClassifier` pour lequel nous n'avons pas fixé le paramètre `max_depth`.

Type de données	***accuracy* moyenne**	***F1-score* moyen**				***log loss* moyen**
		C	**ME**	**PE**	**R**	
RGSF	0,52	0,43	0,36	0,50	0,46	1,14

TABLE 5.5 – Moyenne des métriques mesurant la qualité des classifications utilisant le méthode Hankel-DMD pour générer les *features*. Pour ces classifications, nous avons considéré $m_c = 10$.

que les résultats obtenus étaient meilleurs lorsque nous ne fixions pas nous-même la valeur de ce paramètre au préalable. Par contre, tout comme pour la section précédente, nous allons prendre en compte les deux types de données RGSF et RGS lors des classifications car une nouvelle manière de générer des *features* est expérimentée et nous ne savons pas encore pour quel type de données celle-ci sera la plus efficace.

Plusieurs valeurs de m_c ont été testée durant la recherche et le Tableau 5.4 reprend les scores moyens d'*accuracy* et de *log loss* obtenus en fonction de certaines valeurs de m_c. Dans ce tableau, nous pouvons observer que les scores obtenus ne dépassent pas ceux produits dans les section précédentes exceptés lorsque $m_c = 10$ avec les données de type RGSF. En effet, dans ce cas, l'*accuracy* dépasse les 50% et le *log loss* diminue jusqu'à 1,14 et atteint une valeur minimale par rapport aux *log loss* moyens des autres classifications effectuées jusqu'à présent. Cette classification étant bien meilleure pour les données de type RGSF que pour les données de type RGS, nous allons analyser uniquement le meilleur résultat issu du premier type de données. Les métriques mesurant la qualité de cette classification ont été retenues durant les dix itérations et leurs valeurs sont affichées dans le Tableau A.17 en ANNEXE A. Le tableau reprenant la moyenne de ces métriques est le Tableau 5.5. Dans celui-ci, nous voyons que les *F1-score* moyens des classes ne présentent pas d'aussi grandes différences que ceux obtenus auparavant, mais que leur valeur n'est pas aussi élevée que celle produite pour la "profondément endormi" lors de certaines classifications effectuées dans le chapitre précédent. Cela signifie que le classificateur classifie tous les

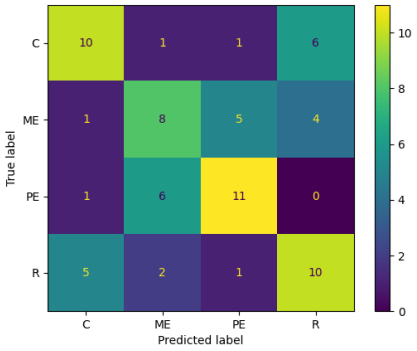

FIGURE 5.13 – Matrice de confusion pour l'itération n°9 de la classification dont les *features* sont générées à l'aide de la méthode Hankel-DMD pour les données RGSF.

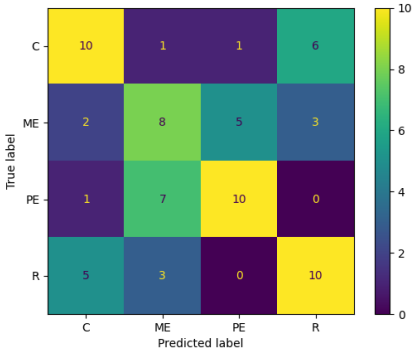

FIGURE 5.14 – Matrice de confusion pour l'itération n°10 de la classification dont les *features* sont générées à l'aide de la méthode Hankel-DMD pour les données RGSF.

états de conscience de manière satisfaisante mais pas aussi bien que l'état de conscience "profondément endormi" dans le chapitre précédent. Les deux matrices de confusion des itérations ayant obtenu les deux plus hauts scores d'*accuracy* expliquent ce phénomène. Elle sont affichées à la Figure 5.13 et la Figure 5.14 et sur celles-ci nous pouvons constater que le modèle *Extra-trees* ne parvient pas bien à distinguer les classes "conscient" et "en phase de réveil" ainsi que les classes "moyennement endormi" et "profondément endormi". En effet, dans la Figure 5.13 par exemple, nous remarquons que le classificateur a classifié six patients comme "moyennement endormi" alors qu'ils étaient en réalité dans l'état de conscience "profondément endormi". Par contre si nous groupons ces classes et que nous prenons d'un côté "conscient ou en phase de réveil" et de l'autre côté "moyennement ou profondément endormi", nous pouvons espérer obtenir de bons résultats au vu de ces matrices de confusion.

En conclusion, dans cette section, nous avons testé une autre manière de générer les *features* des séries temporelles à l'aide de la matrice de Hankel et de l'algorithme de décomposition en modes dynamiques de Hankel. Grâce à cette génération de *features* et au modèle de classification `ExtraTreesClassifier` sans fixer le paramètre `max_depth` au préalable, nous avons réussi à améliorer les scores moyens d'*accuracy* et de *log loss* pour les données de type RGSF. De plus, les matrices de confusion fournissent une information très intéressante : cette classification semble bien séparer les états de conscience lorsqu'ils sont groupés deux par deux de sorte que le premier groupe soit composé des classes "conscient" et "en phase de réveil", et que le deuxième groupe reprenne les classes "moyennement endormi" et "profondément endormi". Cependant, elle semble ne pas parvenir à distinguer correctement les deux états de conscience d'un même groupe.

Dans la section suivante, nous allons combiner deux modèles de classification testés jusqu'à présent pour en créer un nouveau, que nous espérons plus adapté aux données particulières que nous traitons dans ce mémoire. En particulier, la méthode Hankel-DMD sera utilisée pour séparer les données en deux classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi", puis la troisième classification de la section 4.1 sera utilisée pour séparer les classes "moyennement endormi" et "profondément endormi".

5.4 Combinaison de méthodes de classification

Cette section est dédiée à l'élaboration d'une nouvelle méthode de classification construite à partir de deux autres : la méthode Hankel-DMD et la troisième classification de la section 4.1. Cette démarche est effectuée dans l'espoir de produire de meilleurs résultats que ceux obtenus précédemment. En effet, la meilleure *accuracy* résultant des classifications précédentes vaut 0,52 et cela signifie qu'une donnée prise au hasard a 52% de chances d'être bien classifiée. Ce score a été obtenu dans la section 5.3 pour les données de type RGSF et le *log loss* pour cette même cette même classification vaut 1,14.

La première classification effectuée dans cette section est celle générant les *features* à l'aide de la méthode Hankel-DMD pour les données de type RGSF. Celle-ci a été sélectionnée grâce aux résultats obtenus dans la section 5.3 et plus particulièrement grâce aux matrices de confusion affichées sur la Figure 5.13 et la Figure 5.14. En effet, sur ces matrices de confusion, nous pouvons observer que le modèle de classification semble bien séparer les états de conscience s'ils sont groupés deux par deux de sortes à former deux classes plus grandes qui sont "conscient ou en phase de réveil" et "moyennement ou profondément endormi". Nous avons donc réalisé une classification pour les données de type RGSF avec ces deux classes de la même manière que celle effectuée dans la section 5.3. Pour cette classification, les *features* des séries temporelles reprenant les signaux BOLD sont donc générées à l'aide de la méthode Hankel-DMD, et le classificateur utilisé est le modèle `ExtraTreesClassifier` dans lequel nous n'avons pas fixé nous-même le paramètre `max_depth` au préalable.

Les valeurs des métriques mesurant la qualité de cette première classification ont été retenues lors des dix répétitions de celle-ci et elles sont reprises dans le Tableau A.18 de l'Annexe A et les valeurs moyennes sont dans le Tableau 5.6. Dans ce tableau, nous pouvons d'abord observer que l'*accuracy* moyenne dépasse les 80%. Cela confirme bien l'intuition eue en analysant les matrices de confusions affichées à la Figure 5.13 et la Figure 5.14. Le score du *log loss* est également amélioré par rapport aux classifications précédentes car il vaut en moyenne 0,45. Cela est logique car il s'agit d'une classification à deux classes et le modèle de classification a donc beaucoup moins de chance d'accorder une faible probabilité à la bonne classe. La valeur moyenne de ces deux scores nous permettent de conclure que le classificateur testé sur les deux classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi" est performant. En effet, il permet de correctement classifier une donnée prise au hasard dans 81% des cas. Les matrices de confusion correspondant aux deux itérations ayant généré les plus hauts scores d'*accuracy* (0,86 et 0,83 resp.) et les scores de *log loss* les plus faibles (0,44 et 0,44 resp.) sont présentées sur la Figure 5.15 et Figure 5.16. Les nombres dans les différentes cases de ces matrices sont plus élevés que ceux présents dans les autres matrices de confusion car les classes contiennent chacune deux états de conscience et donc $2 \times 18 = 36$ données. Cependant, proportionnellement, nous constatons que la plupart des états de conscience ont été correctement prédits. Ces matrices de confusion ainsi que les *F1-score* moyens pour les deux classes confirment la bonne performance de la classification.

La deuxième classification doit maintenant permettre de séparer les états de conscience "conscient" et "en phase de réveil" ainsi que ceux "moyennement endormi" et "profondément endormi". Dans les sections précédentes ainsi que dans le chapitre précédent, nous avons observé que la plupart des classificateurs parvenaient plus facilement à distinguer la classe "profondément endormi" des autres classes. Nous avons donc repris les résultats obtenus lors des différentes classifications effectuées précédemment et nous avons sélectionné un modèle de classification ayant

Classification	***accuracy* moyenne**	***F1-score* moyen**		***log loss* moyen**
		C+R	**ME+PE**	
R+C vs. ME+PE	0,81	0,80	0,79	0,45

TABLE 5.6 – Moyenne des métriques mesurant la qualité de la première classification intervenant dans cette section. Cette classification sépare les données "conscient ou en phase de réveil" et "moyennement ou profondément endormi" grâce à la méthode Hankel-DMD, pour les données de type RGSF.

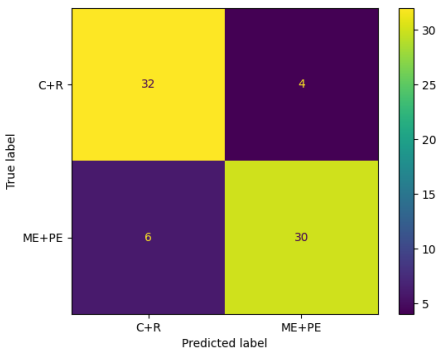

FIGURE 5.15 – Matrice de confusion pour l'itération n°9 de la classification considérant les deux classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi". Pour cette classification, les *features* sont générées à l'aide de la méthode Hankel-DMD pour les données RGSF. Le classificateur est le modèle `ExtraTreesClassifier` sans le paramètre `max_depth` fixé préalablement.

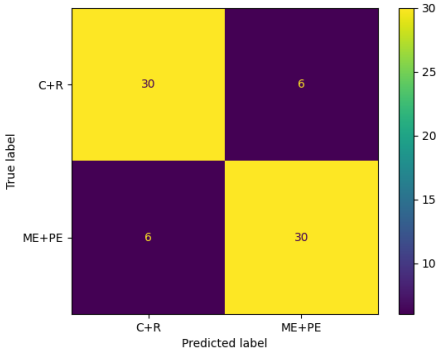

FIGURE 5.16 – Matrice de confusion pour l'itération n°1 de la classification considérant les deux classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi". Pour cette classification, les *features* sont générées à l'aide de la méthode Hankel-DMD pour les données RGSF.Le classificateur est le modèle `ExtraTreesClassifier` sans le paramètre `max_depth` fixé préalablement.

Classification	***accuracy* moyenne**	***F1-score* moyen**		***log loss* moyen**
		ME	**PE**	
ME vs. PE	0,71	0,66	0,59	0,62

TABLE 5.7 – Moyenne des métriques mesurant la qualité de la deuxième classification intervenant dans cette section. Celle-ci sépare les classes "moyennement endormi" et "profondément endormi" grâce à la troisième méthode de la section 4.1, pour les données de type RGS.

une haute *accuracy* moyenne, un faible *log loss* moyen, un *F1-score* moyen élevé pour la classe "profondément endormi" et pouvant être combiné sans trop de difficultés avec un autre modèle de classification. En considérant ces critères, le choix s'est porté sur la deuxième classification présentée dans la section 4.2 pour les données de type RGS car l'*accuracy* moyenne vaut 0,46, le *log loss* moyen vaut 1,28 et le *F1-score* moyen de la classe "profondément endormi" est égal à 0,60. Pour rappel, la génération de *features* pour cette classification se fait en sélectionnant la moyenne des parties réelles des valeurs propres de l'opérateur de Koopman ainsi que les modules des composantes du mode dominant de Koopman. Ces éléments spectraux sont des approximations obtenues via l'application de l'algorithme DMD exacte aux séries temporelles contenant les signaux BOLD. Le modèle de classification, quant à lui, est le modèle `ExtraTreesClassifier` avec le paramètre `max_depth` fixé à 4.

Pour quantifier l'efficacité de cette classification sur les classes "moyennement endormi" et "profondément endormi", nous avons effectué dix répétitions de celle-ci et les métriques mesurant sa qualité sont reprises dans le Tableau A.19 de l'ANNEXE A. Quant aux moyennes de ces métriques, elles sont affichées dans le Tableau 5.7. Les résultats sont un peu moins bons que ceux obtenus lors de la classification précédente, mais ils permettent tout de même de constater une séparation correcte des classes "moyennement endormi" et "profondément endormi". En effet, l'*accuracy* moyenne vaut 0,71 et le *log loss* moyen vaut 0,62. En analysant les *F1-score* moyens des deux classes ainsi que les matrices de confusion correspondant aux deux itérations ayant les plus hautes *accuracy*, affichées sur la Figure 5.17 et la Figure 5.18, nous observons que le classificateur est plus performant pour la classe "moyennement endormi" que pour la classe "profondément endormi". Cette constatation est surprenante au vu des résultats obtenus précédemment pour l'état de conscience "profondément endormi" et montre la tendance du modèle à classifier plus de données dans l'état "moyennement endormi" sans trop impacter la précision de cette dernière classe.

Concernant la séparation des classes "conscient" et "en phase de réveil", aucun classificateur assez efficace n'a été trouvé lors de la recherche. Nous allons donc laisser ces deux états de conscience dans une même classe lors de la combinaison des deux méthodes de classification. Médicalement parlant, ces deux états de conscience ne semblent pas les plus capitaux à distinguer car si un patient est conscient, la détection de son état de conscience ne doit généralement pas être faite. Par conséquent, si un patient inconscient est classé dans la classe "conscient ou en phase de réveil" suite à la classification présentée dans cette section, cela signifiera dans la plupart des cas qu'il est dans l'état "en phase de réveil" avec une probabilité dépendant de la performance de cette classification.

Nous allons maintenant combiner les deux modèles de classification présentés ci-dessus. Étant donné que les états "conscient" et "en phase de réveil" sont regroupés dans une même classe, une

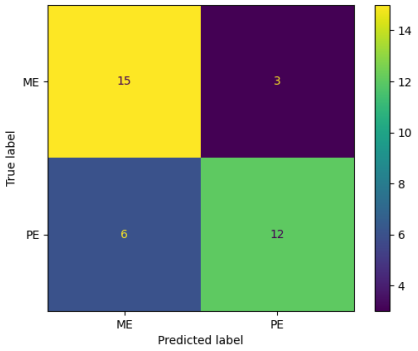

FIGURE 5.17 – Matrice de confusion pour l'itération n°7 de la classification considérant les deux classes "moyennement endormi" et "profondément endormi". Pour cette classification, les *features* sont générées de la même manière que celle utilisée pour la troisième classification de la section 4.1 pour les données RGS.Le classificateur est le modèle `ExtraTreesClassifier` avec le paramètre `max_depth` fixé à 4.

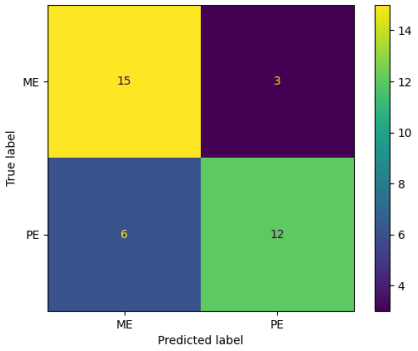

FIGURE 5.18 – Matrice de confusion pour l'itération n°1 de la classification considérant les deux classes "moyennement endormi" et "profondément endormi". Pour cette classification, les *features* sont générées de la même manière que celle utilisée pour la troisième classification de la section 4.1 pour les données RGS.Le classificateur est le modèle `ExtraTreesClassifier` avec le paramètre `max_depth` fixé à 4.

classification à trois classes est proposée afin de distinguer les états "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Premièrement, une classification séparant les classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi" est effectuée selon la première méthode de classification exposée dans cette section. Pour rappel, celle-ci génère les *features* à l'aide de la méthode Hankel-DMD pour les données de type RGSF puis utilise le modèle `ExtraTreesClassifier` pour lequel nous ne fixons pas nous-même le paramètre `max_depth` au préalable. À ce stade, la *target* des données attribuées à la classe "conscient ou en phase de réveil" n'est plus modifiée. Ensuite, les données assignées à la classe "moyennement ou profondément endormi" sont reclassifiées. Cette fois, la classification s'effectue selon la deuxième classification de la section 4.2. Lors de celle-ci, la génération des *features* est réalisée en sélectionnant certaines informations spectrales fournies par l'algorithme DMD exacte, de la même manière que celle de la troisième classification de la section 4.1. Le modèle de classification appliqué sur ces *features* est à nouveau le modèle `ExtraTreesClassifier`, mais cette fois, avec le paramètre `max_depth` fixé à 4. Au terme de cette classification, les données correspondant aux états "moyennement endormi" et "profondément endormi" sont séparées. Pour terminer, en reprenant les données mises de côté après la première classification, nous obtenons une classification des trois classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi".

Dix répétitions de cette double classification ont été effectuées et les valeurs des métriques *accuracy* et *F1-score* ont été retenues puis retranscrites dans le Tableau A.20 de l'ANNEXE A. Les moyennes de ces métriques sont reprises dans le Tableau 5.8. Le *log loss* n'a pas pu être déterminé car la combinaison de méthodes de classification construite dans cette section rend ce calcul compliqué. Dans ce tableau, nous constatons que cette combinaison de méthodes a une

Classification	***accuracy* moyenne**	***F1-score* moyen**		
		C+R	**ME**	**PE**
R+C vs. ME vs. PE	0,72	0,84	0,51	0,67

TABLE 5.8 – Moyenne des métriques mesurant la qualité de la combinaison des deux classifications précédentes. Cette classification sépare les classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi", pour les données de type RGSF puis RGS.

accuracy moyenne de 0,72. Sur base de l'*accuracy*, nous pourrions conclure que cette classification est la plus performante de toutes celles considérées dans ce mémoire. En effet, le résultat semble meilleur pour cette classification avec trois classes mais nous ne pouvons pas réellement comparer les résultats de deux classifications ne séparant pas le même nombre de classes. Dans le Tableau A.20, nous constatons que lors de trois itérations sur les dix, l'*accuracy* de la combinaison de classification atteint voire dépasse la valeur de 0,75 signifiant que, pour ces itérations, une donnée prise au hasard a plus de 75% de chances d'être correctement classifiée. Les deux matrices de confusion correspondant aux itérations n°4 et n°7 sont exposées à la Figure 5.19 et la Figure 5.20, et l'*accuracy* obtenue lors de ces itérations est égale à 0,78 et 0,75 respectivement. Sur ces matrices, nous ne pouvons pas observer une mise en évidence claire de la diagonale même si la classification est efficace car la première classe contient deux fois plus de données que les autres suite aux choix opérés précédemment. Par conséquent, les classes sont déséquilibrées et cela explique cette particularité visuelle due à la légende. Le *F1-score* est une mesure appropriée pour les problèmes de classification sur des ensembles de données déséquilibrés. C'est pourquoi, malgré ce déséquilibre, il permet de comparer la performance du classificateur sur chacune des classes. Dans le Tableau 5.8, nous constatons que la classe "conscient ou en phase de réveil" est celle que le classificateur parvient à distinguer le mieux. Cela n'est pas étonnant car l'*accuracy* moyenne obtenue pour la classification séparant les classes "conscient ou en phase de réveil" et "moyennement ou profondément endormi" atteint les 80% tandis que celle obtenue pour la classification des états "moyennement endormi" et "profondément endormi" est plus faible. Ensuite, la classe "profondément endormi" obtient un *F1-score* moyen de 0,66 et est mieux distinguée par le classificateur que la classe "moyennement endormi", malgré le résultat opposé obtenu lors de la classification de ces deux états uniquement. Ces comportements ne sont pas anormaux car la distribution des données de test dans les deux classifications ne sont pas identiques.

En conclusion, dans cette section, nous avons construit un classificateur combinant deux modèles de classification pour séparer les trois classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Le premier modèle sépare les données en deux classes qui sont "conscient ou en phase de réveil" et "moyennement ou profondément endormi". Pour effectuer cette première classification, les *features* sont générées grâce à la méthode Hankel-DMD sur les données de type RGSF. Ces *features* sont ensuite fournies au modèle `ExtraTreesClassifier` dont le paramètre `max_depth` n'est pas fixé au préalable par nos soins. Les données attribuées à la classe "conscient ou en phase de réveil" sont laissées momentanément de côté et les autres sont reclassifiées par un deuxième modèle. Celui-ci génère les *features* de la façon décrite dans la troisième classification de la section 4.1 puis utilise le modèle `ExtraTreesClassifier` avec le paramètre `max_depth` est fixé à 4 pour séparer les états "moyennement endormi" et "profondément endormi". L'*accuracy* moyenne obtenue après dix répétition de cette double classification vaut 0,72 et c'est la classe "conscient ou en phase de réveil" qui est la mieux détectée par ce processus.

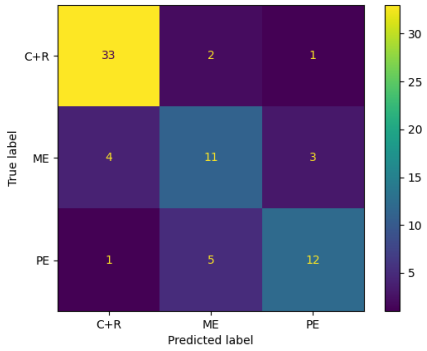

FIGURE 5.19 – Matrice de confusion pour l'itération n°4 de la classification considérant les trois classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Cette classification combine les deux modèles de classification présentés dans cette section.

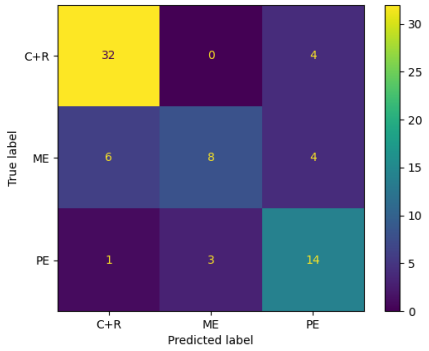

FIGURE 5.20 – Matrice de confusion pour l'itération n°7 de la classification considérant les trois classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Cette classification combine les deux modèles de classification présentés dans cette section.

5.5 Conclusion

Dans ce chapitre, nous avons testé plusieurs techniques dans le but d'améliorer les meilleurs résultats obtenus dans le chapitre précédent. Tout d'abord, nous avons augmenté le nombre de données alimentant la classification grâce à la méthode d'augmentation des données appelée *Window Slicing*. Ensuite, nous avons testé deux nouvelles manières de sélectionner les *features* des séries temporelles contenant les signaux BOLD. L'une est basée sur la méthode du sac de mots tandis que l'autre utilise la matrice de Hankel ainsi que l'algorithme Hankel-DMD. Pour terminer, deux méthodes de classification ont été combinées pour en créer une seule permettant de distinguer les états de conscience en trois classes au lieu de quatre. Il s'agit des classes "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". En effet, aucune méthode de classification assez efficace n'a été trouvée pour distinguer les états "conscient" et "en phase de réveil". Dans ce chapitre, **la meilleure *accuracy* moyenne obtenue pour une classification des quatre états de conscience vaut 0,52 et 0,72 lorsqu'ils sont répartis en seulement trois classes : "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". De plus, l'état de conscience "profondément endormi" est généralement mieux distingué lors des classifications à quatre classes mais c'est la classe "conscient ou en phase de réveil" qui obtient le meilleur *F1-score* moyen pour la classification à trois classes**.

Conclusions et perspectives

L'objectif de ce mémoire était l'élaboration d'une méthode pour classifier les états de conscience de patients n'étant pas dans un état physiologique normal, à partir de données temporelles produites par l'imagerie par résonance magnétique fonctionnelle.

Les trois premiers chapitres nous ont permis d'introduire les concepts théoriques nécessaires à ce mémoire. Le quatrième et le cinquième chapitre présentent les résultats des différentes classifications expérimentées durant la recherche. Chaque section de ces deux chapitres présente une piste d'amélioration que nous avons explorée dans le but d'améliorer la classification des états de conscience. Les meilleurs résultats obtenus dans chaque section sont rappelés dans le Tableau 5.9.

Dans le quatrième chapitre, nous avons d'abord testé différentes manières de sélectionner les *features* des séries temporelles parmi les éléments spectraux fournis par la décomposition en modes dynamiques exacte. Suite à ces premières classifications, nous nous sommes rendus compte que les *features* obtenues en considérant les parties réelles des valeurs propres de l'opérateur de Koopman approximées permettaient de séparer plus efficacement les données. Ensuite, nous avons varié la valeur de l'un des paramètres du modèle *Extra-trees* afin de déterminer son importance pour le classificateur. Ce paramètre régule la profondeur des arbres de décision des *Extra-trees*, c'est-à-dire le nombre de nœuds par lesquels passent les données. Après plusieurs expériences, nous avons conclu que les meilleurs résultats étaient obtenus lorsque ce paramètre était fixé à la valeur de 4. Pour terminer, la performance du modèle *Extra-trees* a été comparée à celle d'autres classificateurs, qui se sont avérés moins bons pour répondre à la problématique de la détection des états de conscience. Au terme de ce chapitre, nous avons conclu que la classification la plus efficace était celle qui considérait le modèle *Extra-trees* dont le paramètre de profondeur est fixé à 4, ainsi que les *features* composées des parties réelles des valeurs propres de l'opérateur de Koopman approximées par l'algorithme de décomposition en modes dynamiques exacte. Le type de données utilisé pour obtenir ces résultats est le type *reduced GS*. Dans ce cas, l'*accuracy* moyenne obtenue sur dix répétitions de cette classification vaut 0,46 tandis que le *log loss* moyen vaut 1,28. Grâce aux matrices de confusion et aux *F1-score* moyens, nous avons également pu déterminer que la classe la mieux prédite est celle correspondant à l'état de conscience "profondément endormi".

Enfin, dans le cinquième chapitre, d'autres pistes moins conventionnelles ont été explorées dans le but d'améliorer les résultats obtenus dans le chapitre précédent. Premièrement, nous avons augmenté le nombre de données prises en entrée par le classificateur en extrayant des sous-séries à partir des séries temporelles de départ. Deuxièmement, nous avons mis en pratique le modèle du sac de mots pour générer les *features* de la classification à partir des séries temporelles contenant les signaux BOLD. Malheureusement, aucune de ces manipulations n'a permis d'améliorer les meilleurs résultats obtenus dans le chapitre précédent. Ensuite, nous avons appliqué la

décomposition en modes dynamiques de Hankel au lieu de celle dite exacte afin de produire d'une manière différente des éléments spectraux caractérisant les séries temporelles. En les passant en entrée du modèle de classification des *Extra-trees* pour lequel nous n'avons pas fixé le paramètre de profondeur au préalable, nous avons réussi à améliorer les résultats lorsque les données de type *reduced GS filtered* étaient considérées. En effet, pour cette classification, l'*accuracy* moyenne atteint les 52% et la valeur du *log loss* est réduite à 1,14. Grâce aux matrices de confusion et aux *F1-score* moyens, nous avons conclu que la classe pour laquelle la classification est la plus efficace est à nouveau celle correspondant à l'état de conscience "profondément endormi". Pour terminer, nous avons combiné les meilleures méthodes de classification du quatrième et cinquième chapitre pour effectuer une classification à trois classes, correspondant aux états "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Cette combinaison nous a permis d'obtenir de meilleurs résultats, à savoir une *accuracy* moyenne de 0,72. La classe la mieux prédite pour cette dernière expérience est la classe "conscient ou en phase de réveil".

Au terme de ce mémoire, nous sommes donc parvenus à proposer une nouvelle variante de la méthode de détection des états de conscience sur base des signaux BOLD obtenus via l'imagerie par résonance magnétique fonctionnelle. Pour la classification des quatre états de conscience, les résultats obtenus avec la décomposition en modes dynamiques de Hankel sont encourageants. Cependant, dans ce travail, les performances de cette classification semblent atteindre un plafond ne dépassant pas les 55% d'*accuracy*. Dans une perspective de travaux futurs, nous allons mettre en évidence les problèmes soulevés lors de la recherche et proposer plusieurs pistes de solution.

Une première explication de cette limitation de performance est le nombre insuffisant de données disponibles pour entraîner les modèles de classification. En effet, le groupe *Coma Science Group* de l'Université de Liège a récolté des signaux BOLD de 18 voire 17 patients seulement, ce qui nous donne un nombre maximal de $18 \times 4 = 72$ données. Cela est largement insuffisant pour permettre aux modèles de classification de créer un ensemble de règles séparant efficacement les états de conscience. Pour résoudre ce problème, il faudrait donc récolter plus de données sur d'autres patients car l'augmentation des données à partir de celles déjà disponibles ne suffit pas.

Une seconde raison est le manque d'expertise dans le domaine médical, et plus précisément en ce qui concerne la simulation d'un état végétatif ou l'imagerie par résonance magnétique fonctionnelle. En effet, le sédatif administré lors de l'étude réalisée par le *Coma Science Group* n'agit pas forcément à la même vitesse sur tous les sujets. Il y a donc peut-être une partie des données qui biaise les résultats. Par exemple, les signaux BOLD de certains sujets ont été récoltés sur 197 points de temps, tandis que pour d'autres sujets, ils ont été récoltés sur 347 points de temps. De plus, une meilleure connaissance des signaux BOLD nous permettrait également de générer des *features* plus adéquates pour faciliter la séparation des états de conscience. Pour toutes ces raisons, il serait intéressant d'organiser une collaboration avec des experts du domaine.

Si une plus grande quantité de données est collectée, il pourrait être envisagé de recourir à des techniques de *Deep Learning* pour classifier les états de conscience. Nous pourrions faire appel aux réseaux de neurones récurrents [16] et en particulier aux *Long Short-Term Memory* (LSTM) [15]. Par exemple, cette technique a été utilisée très récemment sur base de données électroencéphalographiques pour définir la conscience selon deux composantes : l'éveil et la perception de soi et de l'environnement [24]. Toujours dans le domaine des réseaux de neurones récurrents, nous pourrions également nous tourner vers le *Reservoir Computing* [21].

Section	**Type de données**	***accuracy* moyenne**	***log loss* moyen**	**Classe la mieux classifiée**	***F1-score* moyen correspondant**
Premières classifications	RGS	0,44	1,28	PE	0,61
Variation du paramètre `max_depth`	RGS	0,46	1,28	PE	0,60
Autres classificateurs	RGS	0,46	1,33	PE	0,60
Augmentation des données	RGS	0,44	1,20	PE	0,65
Modèle du sac de mots	RGS	0,44	1,31	PE	0,43
Hankel-DMD	RGSF	0,52	1,14	PE	0,50
Combinaison de méthodes (3 classes)	RGSF puis RGS	0,72	/	C+R	0,84

TABLE 5.9 – Tableau récapitulatif des valeurs moyennes des métriques mesurant la qualité d'une classification. Pour chaque section, les meilleurs résultats ont été retenus. C, ME, PE et R correspondent respectivement aux états de conscience "conscient", "moyennement endormi", "profondément endormi" et "en phase de réveil".

Bibliographie

[1] ARBABI H., MEZIĊ I., *Ergodic theory, Dynamic Mode Decomposition and Computation of Spectral Properties of the Koopman operator*, SIAM Journal on Applied Dynamical Systems 16.4, 2017.

[2] BENZAKI Y., *Tout ce que vous voulez savoir sur l'algorithme K-Means*, `https://mrmint.fr/algorithme-k-means`, consulté pour la dernière fois le 11 mai 2021.

[3] BEX T., *Comprehensive Guide to Multiclass Classification Metrics*, `https://towardsdatascience.com/comprehensive-guide-on-multiclass-classification-metrics-af94cfb83fbd`, consulté pour la dernière fois le 7 mai 2022.

[4] BROWNIEE J., *A Gentle Introduction to the Bag-of-Words Model*, `https://machinelearningmastery.com/gentle-introduction-bag-words-model/`, consulté pour la dernière fois le 1 mai 2021.

[5] BRUNTON S., *Notes on Koopman Operator Theory*, University of Washington, 2019.

[6] CUI M., *Introduction to the K-Means Clustering Algorithm Based on the Elbow Method*, Clausius Scientific Press, Accounting, Auditing and Finance 1 : 5-8, 2020.

[7] DEBAUCHE V., *Etude de l'Opérateur de Koopman en théorie du contrôle : Application à la prédiction et la p-dominance*, Mémoire présenté pour l'obtention du grade académique de master en Sciences Mathématiques, Université de Namur, Faculté des Sciences, 2019.

[8] DELAUNOIS B., *Approche Spectrale pour la Classification d'Etats de Conscience*, Mémoire présenté pour l'obtention du grade académique de master en Sciences Mathématiques à finalité spécialisée en perspectives professionnelles des mathématiques appliquées, Université de Namur, Faculté des Sciences, 2018.

[9] DEVLIN H. et al., *Introduction to FMRI*, `https://www.ndcn.ox.ac.uk/divisions/fmrib/what-is-fmri/introduction-to-fmri`, consulté pour la dernière fois le 20 avril 2021.

[10] FRENAY B., *Cours : Machine learning et data mining*, IDASM102, Université de Namur, 2021.

[11] GAUTIER L., *Différence entre l'oxyhémoglobine et la désoxyhémoglobine*, `https://fr.differbetween.com/article/difference_between_oxyhemoglobin_and_deoxyhemoglobin#what_is_the_function_of_oxyhemoglobin`, consulté pour la dernière fois le 17 mai 2022.

[12] GEURTS P., ERNST D., WEHENKEL L., *Extremely randomized trees*, Machine Learning, 63(1), 3–42, 2006.

[13] GOSSERIES O. et al., *Que mesure la neuro-imagerie fonctionnelle : IRMf, TEP & MEG ?*, Rev Med Liege, 2007.

[14] GRANDINI M., BAGLI E., VISANI G., *Metrics for Multi-Class Classification : an Overview*, 2020.

[15] HOCHREITER S., SCHMIDHUBER J., *Long short-term memory*, Neural Computation 9, p. 1735–1780, 1997.

[16] HÜSKEN M., STAGGE P., *Recurrent neural networks for time series classification*, Neurocomputing 50, p. 223-235, 2003.

[17] INCONNU, *Choosing the right estimator* ; `https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html`, consulté pour la dernière fois le 19 mai 2022.

[18] INCONNU, *De la RMN à l'IRM*, `https://lasciencepourtous.cafe-sciences.org/articles/de-la-rmn-a-lirm/`, consulté pour la dernière fois le 20 avril 2021.

[19] INCONNU, *L'imagerie par résonance magnétique fonctionnelle (IRMf)*, `https://www.reflexions.uliege.be/cms/c_7613/`, consulté pour la dernière fois le 18 mai 2022.

[20] IWANA B.K., UCHIDA S., *An empirical survey of data augmentation for time series classification with neural networks*, PLOS ONE 16(7), 2021.

[21] JAEGER H., *The "echo state" approach to analysing and training recurrent neural networks - with an Erratum note*, GMD Report 148, 2010.

[22] KARABIBER F., *TF-IDF — Term Frequency-Inverse Document Frequency*, `https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/`, consulté pour la dernière fois le 12 mai 2022.

[23] KOOPMAN B.O., *Hamiltonian Systems and Transformation in Hilbert Space*, Proceedings of the National Academy of Sciences USA, 17(5) : p. 315–318, 1931.

[24] LEE M., SANZ L.R.D., BARRA A. et al., *Quantifying arousal and awareness in altered states of consciousness using interpretable deep learning*, Nature Communications volume 13, 1064, 2022. `https://doi.org/10.1038/s41467-022-28451-0`

[25] MAIESE K., *Locked-in syndrome (syndrome d'enfermement)*, `https://www.msdmanuals.com/fr/professional/troubles-neurologiques/coma-et-troubles-de-la-conscience/locked-in-syndrome-syndrome-enfermement`, consulté pour la dernière fois le 19 avril 2021.

[26] MANNOR S., *k-Armed Bandit*. In : Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, 2011.

[27] MAUROY A., *Cours : Méthodes avancées pour les systèmes non-linéaires*, SMATM227, Université de Namur, 2021.

[28] MEZIC I., *Spectral Properties of Dynamical Systems, Model Reduction and Decompositions*, Nonlinear Dynamics, 41 : p. 309–325, 2005.

[29] NAVLANI A., *KNN Classification using Scikit-learn*, `https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn`, consulté pour la dernière fois le 11 mai 2021.

[30] QIAN S., CHOU C.-A., *A Koopman-operator-theoretical approach for anomaly recognition and detection of multi-variate EEG system*, Biomedical Signal Processing and Control 69, 2021.

[31] QUINLAN J.R., *Induction od decision trees*, Machine Learning, 1(1), 81-106, Kluwer Academic Publishers, 1986.

[32] ROWLEY C.W. et al., *Spectral analysis of nonlinear flows*, Journal of Fluid Mechanics, 641 : p. 115-127, 2009.

[33] SCHMID P.J., SESTERHENN J., *Dynamic mode decomposition of numerical and experimental data*, Journal of Fluid Mechanics, vol. 656, p. 5–28, 2008.

[34] SURANA A., *Koopman Operator Framework for Times Series Modelling and Analysis*, Journal of Nonlinear Science, 30, 1973-2006, 2018.

[35] TU J.H. et al., *On Dynamic Mode Decomposition : Theory and Applications*, Journal of Computational Dynamics, 1(2), 391-421, 2014.

[36] VANHAUDENHUYSE A. et al., *Détecter les signes de conscience chez le patient en état de conscience minimale*, Réanimation, 16, 527-532, 2007.

[37] VILLANI C., *Optimal transport, old and new*, Springer, p. 105-120, 2008.

[38] YOHAN C., *Qu'est ce que l'algorithme KNN ?*, `https://datascientest.com/knn`, consulté pour la dernière fois le 6 mai 2021.

Annexe A

Résultats

A.1 Premières classifications

A.1.1 Classification sur base de tous les éléments spectraux

Itération	***accuracy* RGSF**	***accuracy* RGS**
1	0,33	0,35
2	0,38	0,43
3	0,33	0,35
4	0,43	0,39
5	0,31	0,29
6	0,33	0,38
7	0,31	0,42
8	0,33	0,38
9	0,39	0,39
10	0,38	0,26

TABLE A.1 – Tableau récapitulatif des *accuracy* de la première classification pour les deux types de données RGSF et RGS.

Itération	***F1-score* RGSF**				***F1-score* RGS**			
	C	**ME**	**PE**	**R**	**C**	**ME**	**PE**	**R**
1	0,19	0,19	0,32	0,31	0,19	0,22	0,37	0,33
2	0,06	0,42	0,40	0,32	0,28	0,31	0,44	0,32
3	0,08	0,28	0,40	0,25	0,17	0,18	0,32	0,43
4	0,22	0,26	0,37	0,51	0,20	0,33	0,32	0,34
5	0,16	0,19	0,28	0,31	0,13	0,10	0,25	0,31
6	0,08	0,30	0,35	0,26	0,30	0,11	0,44	0,41
7	0,16	0,16	0,24	0,26	0,30	0,31	0,37	0,37
8	0,23	0,23	0,31	0,30	0,14	0,21	0,52	0,33
9	0,20	0,30	0,36	0,38	0,19	0,29	0,35	0,48
10	0,21	0,36	0,27	0,30	0,15	0,07	0,29	0,31

TABLE A.2 – Tableau récapitulatif des *F1-score* de la première classification pour chaque état de conscience et pour les deux types de données RGSF et RGS.

Itération	***log loss* RGSF**	***log loss* RGS**
1	1,32	1,34
2	1,33	1,33
3	1,32	1,34
4	1,33	1,35
5	1,37	1,34
6	1,33	1,32
7	1,35	1,34
8	1,33	1,36
9	1,29	1,35
10	1,31	1,37

TABLE A.3 – Tableau récapitulatif des *log loss* de la première classification pour les deux types de données RGSF et RGS.

A.1.2 Classification sur base d'une sélection des éléments spectraux

Itération	***accuracy* RGSF**	***accuracy* RGS**
1	0,18	0,33
2	0,31	0,35
3	0,22	0,28
4	0,25	0,33
5	0,24	0,39
6	0,22	0,31
7	0,29	0,35
8	0,18	0,31
9	0,28	0,31
10	0,18	0,33

TABLE A.4 – Tableau récapitulatif des *accuracy* de la deuxième classification pour les deux types de données RGSF et RGS.

Itération	***F1-score* RGSF**				***F1-score* RGS**			
	C	ME	PE	R	C	ME	PE	R
1	0,09	0,11	0,23	0,11	0,32	0,15	0,44	0,09
2	0,31	0,19	0,32	0,10	0,25	0,13	0,53	0,18
3	0,30	0,09	0,20	0,09	0,21	0,11	0,42	0,04
4	0,21	0,24	0,29	0,06	0,25	0,22	0,39	0,17
5	0,15	0,13	0,37	0,17	0,35	0,32	0,54	0,06
6	0,19	0,06	0,33	0,05	0,32	0,17	0,34	0,11
7	0,11	0,18	0,50	0,11	0,29	0,12	0,43	0,17
8	0,06	0,09	0,41	0,06	0,21	0,21	0,38	0,19
9	0,26	0,18	0,34	0,07	0,16	0,19	0,40	0,20
10	0,20	0,06	0,22	0,06	0,18	0,24	0,43	0,26

TABLE A.5 – Tableau récapitulatif des *F1-score* de la deuxième classification pour chaque état de conscience et pour les deux types de données RGSF et RGS.

Itération	***log loss* RGSF**	***log loss* RGS**
1	1,44	1,33
2	1,41	1,35
3	1,42	1,35
4	1,40	1,36
5	1,45	1,32
6	1,45	1,33
7	1,46	1,33
8	1,43	1,34
9	1,42	1,34
10	1,46	1,35

TABLE A.6 – Tableau récapitulatif des *log loss* de la deuxième classification pour les deux types de données RGSF et RGS.

A.1.3 Classification sur base d'une moyenne des éléments spectraux

Itération	***accuracy* RGSF**	***accuracy* RGS**
1	0,31	0,40
2	0,32	0,44
3	0,26	0,54
4	0,28	0,46
5	0,26	0,43
6	0,33	0,43
7	0,33	0,39
8	0,31	0,40
9	0,32	0,44
10	0,27	0,42

TABLE A.7 – Tableau récapitulatif des *accuracy* de la troisième classification pour les deux types de données RGSF et RGS.

Itération	***F1-score* RGSF**				***F1-score* RGS**			
	C	**ME**	**PE**	**R**	**C**	**ME**	**PE**	**R**
1	0,26	0,19	0,43	0,13	0,28	0,14	0,61	0,31
2	0,28	0,09	0,49	0,13	0,29	0,26	0,63	0,30
3	0,22	0,12	0,44	0,04	0,40	0,43	0,58	0,39
4	0,18	0,19	0,42	0,03	0,26	0,36	0,61	0,35
5	0,13	0,15	0,40	0,19	0,28	0,15	0,70	0,32
6	0,27	0,16	0,51	0,09	0,35	0,21	0,54	0,29
7	0,14	0,20	0,51	0,18	0,33	0,18	0,53	0,23
8	0,43	0,09	0,38	0,06	0,23	0,13	0,63	0,35
9	0,30	0,23	0,44	0,04	0,34	0,20	0,61	0,30
10	0,19	0,06	0,45	0,10	0,27	0,23	0,67	0,21

TABLE A.8 – Tableau récapitulatif des *F1-score* de la troisième classification pour chaque état de conscience et pour les deux types de données RGSF et RGS.

Itération	***log loss* RGSF**	***log loss* RGS**
1	1,43	1,31
2	1,41	1,25
3	1,41	1,30
4	1,38	1,26
5	1,42	1,28
6	1,44	1,29
7	1,40	1,28
8	1,43	1,27
9	1,37	1,28
10	1,40	1,26

TABLE A.9 – Tableau récapitulatif des *log loss* de la troisième classification pour les deux types de données RGSF et RGS.

A.2 Variation du paramètre régulant la profondeur des arbres du modèle *Extra-trees*

Itération	***accuracy* RGS**	***F1-score* RGS**				***log loss* RGS**
		C	**ME**	**PE**	**R**	
1	0,40	0,32	0,11	0,50	0,35	1,29
2	0,49	0,34	0,36	0,59	0,36	1,31
3	0,47	0,35	0,20	0,61	0,36	1,30
4	0,47	0,31	0,23	0,57	0,46	1,29
5	0,49	0,33	0,25	0,56	0,45	1,30
6	0,49	0,39	0,27	0,59	0,33	1,26
7	0,43	0,30	0,19	0,59	0,31	1,31
8	0,51	0,39	0,31	0,61	0,42	1,28
9	0,39	0,20	0,23	0,57	0,29	1,29
10	0,43	0,32	0,20	0,54	0,36	1,28

TABLE A.10 – Tableau récapitulatif des métriques mesurant la qualité de la classification utilisant la classe `GridSearchCV` sur le paramètre `max_depth` selon les valeurs reprises dans le vecteur $(3, 4, 5, 8, 10, 20, 30)$.

Itération	***accuracy* RGS**	***F1-score* RGS**				***log loss* RGS**
		C	**ME**	**PE**	**R**	
1	0,46	0,39	0,33	0,56	0,32	1,28
2	0,42	0,22	0,33	0,54	0,31	1,28
3	0,42	0,29	0,26	0,61	0,22	1,27
4	0,53	0,43	0,37	0,63	0,41	1,26
5	0,42	0,24	0,28	0,62	0,19	1,28
6	0,46	0,28	0,20	0,59	0,40	1,29
7	0,47	0,30	0,29	0,63	0,37	1,29
8	0,49	0,30	0,40	0,59	0,39	1,28
9	0,47	0,41	0,32	0,61	0,27	1,28
10	0,46	0,26	0,23	0,61	0,36	1,28

TABLE A.11 – Tableau récapitulatif des métriques mesurant la qualité de la classification prenant pour paramètre `max_depth` $= 4$ dans le modèle`ExtraTreesClassifier`.

A.3 Augmentation des données

Itération	***accuracy* RGS**	***F1-score* RGS**				***log loss* RGS**
		C	**ME**	**PE**	**R**	
1	0,43	0,33	0,38	0,67	0,21	1,27
2	0,42	0,28	0,35	0,66	0,23	1,27
3	0,41	0,29	0,37	0,63	0,18	1,27
4	0,45	0,33	0,43	0,66	0,22	1,27
5	0,41	0,26	0,38	0,64	0,23	1,27
6	0,43	0,28	0,40	0,65	0,22	1,27
7	0,42	0,28	0,35	0,68	0,21	1,27
8	0,43	0,32	0,35	0,69	0,20	1,26
9	0,43	0,29	0,40	0,66	0,19	1,27
10	0,43	0,30	0,40	0,65	0,20	1,27

TABLE A.12 – Tableau récapitulatif des métriques mesurant la qualité de la classification considérant 576 données en entrée grâce à une technique d'augmentations des données appelée *Window Slicing*. Lors de cette classification, les données sont considérées indépendantes les unes des autres.

Itération	***accuracy* RGS**	***F1-score* RGS**				***log loss* RGS**
		C	**ME**	**PE**	**R**	
1	0,39	0,21	0,32	0,67	0,07	1,20
2	0,43	0,31	0,30	0,69	0,09	1,20
3	0,40	0,24	0,32	0,63	0,12	1,20
4	0,43	0,23	0,25	0,69	0,29	1,19
5	0,46	0,25	0,33	0,65	0,22	1,20
6	0,49	0,30	0,41	0,69	0,21	1,19
7	0,39	0,34	0,15	0,63	0,16	1,19
8	0,47	0,34	0,33	0,67	0,23	1,20
9	0,50	0,53	0,27	0,63	0,24	1,19
10	0,42	0,26	0,34	0,61	0,24	1,20

TABLE A.13 – Tableau récapitulatif des métriques mesurant la qualité de la classification considérant 576 données en entrée grâce à une technique d'augmentations des données appelée *Window Slicing*. Lors de cette classification, les séries temporelles de départ sont reformées et les classes attribuées à celles-ci sont celles prédites le plus grand nombre de fois par le classificateur pour les fenêtres extraites de cette série temporelle.

A.4 Modèle du sac de mots

Itération	***accuracy* RGSF**	***accuracy* RGS**
1	0,31	0,46
2	0,28	0,44
0 3	0,31	0,47
4	0,29	0,40
5	0,29	0,42
6	0,29	0,46
7	0,26	0,44
8	0,28	0,42
9	0,26	0,44
10	0,29	0,44

TABLE A.14 – Tableau récapitulatif des *accuracy* de la classification dont les *features* sont générée grâce à la méthode du sac de mots pour les deux types de données RGSF et RGS. Dans cette méthode, le nombre k de clusters considérés par l'algorithme k-médoïdes vaut 150 pour les données de type RGSF et 100 pour celles de type RGS.

Itération	*F1-score* RGSF				*F1-score* RGS			
	C	**ME**	**PE**	**R**	**C**	**ME**	**PE**	**R**
1	0,26	0,15	0,23	0,33	0,36	0,49	0,45	0,21
2	0,23	0,13	0,18	0,30	0,28	0,44	0,48	0,23
3	0,24	0,15	0,30	0,27	0,46	0,41	0,44	0,25
4	0,22	0,14	0,32	0,22	0,24	0,40	0,39	0,24
5	0,25	0,20	0,24	0,23	0,27	0,42	0,44	0,25
6	0,27	0,11	0,36	0,22	0,36	0,46	0,39	0,35
7	0,23	0,17	0,24	0,22	0,42	0,37	0,41	0,22
8	0,19	0,15	0,30	0,26	0,26	0,42	0,36	0,37
9	0,22	0,13	0,28	0,19	0,31	0,39	0,50	0,29
10	0,21	0,20	0,24	0,24	0,27	0,43	0,43	0,30

TABLE A.15 – Tableau récapitulatif des *F1-score* pour chaque état de conscience de la classification dont les *features* sont générée grâce à la méthode du sac de mots pour les deux types de données RGSF et RGS. Dans cette méthode, le nombre k de clusters considérés par l'algorithme k-médoïdes vaut 150 pour les données de type RGSF et 100 pour celles de type RGS.

Itération	*log loss* RGSF	*log loss* RGS
1	1,36	1,31
2	1,36	1,31
3	1,37	1,31
4	1,37	1,31
5	1,36	1,31
6	1,36	1,30
7	1,36	1,31
8	1,38	1,31
9	1,38	1,30
10	1,37	1,31

TABLE A.16 – Tableau récapitulatif des *log loss* de la classification dont les *features* sont générée grâce à la méthode du sac de mots pour les deux types de données RGSF et RGS. Dans cette méthode, le nombre k de clusters considérés par l'algorithme k-médoïdes vaut 150 pour les données de type RGSF et 100 pour celles de type RGS.

A.5 Hankel DMD

Itération	***accuracy* RGSF**	***F1-score* RGSF**				***log loss* RGSF**
		C	**ME**	**PE**	**R**	
1	0,50	0,43	0,33	0,54	0,44	1,19
2	0,51	0,37	0,40	0,48	0,46	1,12
3	0,53	0,43	0,33	0,48	0,53	1,12
4	0,51	0,41	0,36	0,48	0,47	1,11
5	0,51	0,40	0,33	0,56	0,47	1,12
6	0,53	0,50	0,36	0,48	0,46	1,15
7	0,50	0,41	0,34	0,48	0,40	1,14
8	0,51	0,41	0,35	0,52	0,46	1,17
9	0,54	0,48	0,37	0,54	0,45	1,13
10	0,53	0,44	0,37	0,48	0,45	1,10

TABLE A.17 – Tableau récapitulatif des métriques mesurant la qualité de la classification dont les *features* sont générées à l'aide de la méthode Hankel-DMD les données RGSF.

A.6 Combinaison de méthodes de classification

Itération	***accuracy* RGSF**	***F1-score* RGSF**		***log loss* RGSF**
		C+R	**ME+PE**	
1	0,83	0,82	0,82	0,44
2	0,79	0,78	0,76	0,47
3	0,82	0,82	0,79	0,45
4	0,76	0,75	0,75	0,46
5	0,81	0,80	0,79	0,45
6	0,78	0,77	0,75	0,46
7	0,83	0,82	0,81	0,46
8	0,83	0,81	0,82	0,45
9	0,86	0,86	0,83	0,44
10	0,82	0,82	0,79	0,44

TABLE A.18 – Tableau récapitulatif des métriques mesurant la qualité de la classification à deux classes, celles-ci étant "conscient ou en phase de réveil" et "moyennement ou profondément endormi". Les *features* de cette classification sont générées à l'aide de l'algorithme Hankel-DMD pour les données de type RGSF.

Itération	***accuracy* RGS**	***F1-score* RGS**		***log loss* RGS**
		ME	**PE**	
1	0,75	0,72	0,61	0,61
2	0,72	0,67	0,59	0,61
3	0,72	0,68	0,61	0,62
4	0,64	0,56	0,59	0,64
5	0,69	0,64	0,61	0,63
6	0,72	0,67	0,59	0,62
7	0,75	0,72	0,61	0,61
8	0,75	0,72	0,61	0,63
9	0,69	0,65	0,54	0,62
10	0,67	0,61	0,54	0,63

TABLE A.19 – Tableau récapitulatif des métriques mesurant la qualité de la classification à deux classes, celles-ci étant "moyennement endormi" et "profondément endormi". Les *features* de cette classification sont celles sélectionnées lors de la troisième classification de la section 4.1 pour les données de type RGS.

Itération	***accuracy***	***F1-score***		
		C+R	**ME**	**PE**
1	0,71	0,82	0,51	0,67
2	0,68	0,82	0,47	0,63
3	0,75	0,82	0,65	0,70
4	0,78	0,89	0,61	0,71
5	0,72	0,84	0,52	0,67
6	0,71	0,82	0,50	0,67
7	0,75	0,85	0,55	0,70
8	0,67	0,78	0,39	0,67
9	0,71	0,85	0,48	0,62
10	0,71	0,85	0,45	0,63

TABLE A.20 – Tableau récapitulatif des métriques mesurant la qualité de la classification à trois classes, celles-ci étant "conscient ou en phase de réveil", "moyennement endormi" et "profondément endormi". Cette classification résulte de la combinaison de deux modèles de classification : celui dont les *features* sont générées à l'aide de l'algorithme Hankel-DMD et la troisième classification de la section 4.1.

Annexe B

Codes

Le langage utilisé pour implémenter tous les codes ayant permis d'obtenir les résultats présentés dans ce mémoire est le langage `Python` (version 3.7.3). Ces codes sont disponibles via l'URL suivante : `https://github.com/chdarden/Memoire_Dardenne_Charline/archive/main.zip`.