Rapport de stage

MArche meSurée par le dispositif eGaiT chez la pERsonne âgée (Projet MASTER)

Manon Simonot

Nantes Université
M2 Ingénierie Statistique

2024-08-28

Encadrement :
Thibault Deschamps (LMIP)
Aymeric Stamm (LMJL)
Lise Bellanger (LMJL)

Table des matières

1 Introduction

Les **troubles de la marche** touchent de nombreuses personnes et leurs causes sont diverses. Ils impactent énormément le quotidien et diminuent fortement la qualité de vie des personnes concernées, pouvant provoquer des chutes et blessures, et se caractérisant par une perte de locomotion et donc d'autonomie. Les troubles de la marche sont notamment présents chez les personnes souffrant de maladies neurodégénératives et chez les populations âgées.

Dans ce contexte naît le projet **eGait** en 2017 au Laboratoire de Mathématiques Jean Leray (LMJL). Il a pour objet l'étude de la démarche grâce à un dispositif portatif placé à la hanche, mesurant des informations quantitatives sur la rotation de la hanche au cours du temps lors de la marche. L'objectif de ce projet est la détection précoce des troubles de la marche pour prévenir les chutes, mesurer l'état de santé des patients, voire permettre une rééducation anticipée adaptée pour prévenir ou même retarder les symptômes. Chez les personnes âgées, la démarche et spécifiquement certains paramètres comme la vitesse de marche ou la variabilité de la démarche sont de très bons indicateurs de leur état de santé générale, que ce soit leur santé physique ou leur cognition (Beauchet et al. [2]).

Ainsi, une collaboration entre le LMJL et le Laboratoire Motricité, Interactions, Performance (LMIP) a vu le jour, afin de se concentrer sur les troubles de la marche chez les personnes âgées. Mon projet de stage "MASTER" s'inscrit dans cette collaboration interdisciplinaire, permettant d'allier les connaissances du domaine de la compréhension et de l'analyse du mouvement humain, avec celles du domaine des mathématiques appliquées.

L'objectif principal de ce stage est l'**évaluation de la concordance** entre le dispositif eGait porté par le LMJL et le tapis de marche GAITRite© considéré comme méthode de référence, pour valider le dispositif eGait. Pour effectuer cette étude, il a fallut commencer par acquérir des données de références afin d'améliorer la segmentation des cycles de marche détectés par le dispositif eGait. Cela permet d'obtenir des **paramètres spatio-temporels** fiables dont nous pouvons nous servir pour comparer les dispositifs. La construction d'un modèle de machine learning permettant de segmenter le signal obtenu par le dispositif eGait a donc finalement occupé la place la plus importante au sein de ce stage, et sera détaillée dans ce rapport.

Pour information, tout le code a été réalisé en `R`, et les schémas de ce rapport ont été fait sur le site `escalidraw` [1].

[1] https://math.preview.excalidraw.com

2 Remerciements

Je tiens tout d'abord à remercier Lise Bellanger, Thibault Deschamps et Aymeric Stamm de m'avoir accompagnée tout au long de ce stage, étant toujours disponibles pour me guider dans mes recherches tout en me faisant confiance sur le travail effectué. Cette collaboration entre laboratoires m'a apporté de nombreuses connaissances et a été une très belle opportunité, et je vous remercie de m'accorder une fois de plus votre confiance pour continuer à travailler sur ce projet après ce stage.

Je tiens également à remercier les volontaires ayant participé aux acquisitions de données au CHU de Nantes : sans vous, aucun modèle n'aurait pu être construit. Je remercie donc chaleureusement Margot Bornet, Michael Barrion et Nadia Negab d'avoir marché avec enthousiasme sur ce tapis pendant mon stage, ainsi que tous les autres volontaires ayant participé à des acquisitions dans ce contexte dans le passé.

Enfin, ce stage m'a permis de participer à différents évènements, ce dont je suis reconnaissante. J'ai en effet pu participer aux "Rencontres R 2024" à Vannes, me permettant d'élargir mes connaissances sur le logiciel `R`. J'ai également participé à la "Journée Innovation Mathématiques" (JIM) avec mes collègues Nadia Negab et Klervi Le Gall afin de représenter le projet eGait. Enfin, auprès du réseau ELIT permettant le financement de mon stage, j'ai participé aux "Masteriales DELPHI ELIT HEMI FAME" afin de présenter mon stage aux autres bénéficiaires de cette bourse. Un entretien a également été publié auprès de ce réseau, décrivant mon expérience pendant ce stage [1].

[1] Cet entretien est disponible sur le carnet d'hypothèses ELIT : https://elit.hypotheses.org/1750.

3 Pré-requis

Mon stage se situant au sein du projet eGait, il est nécessaire de détailler certaines notions qui sont essentielles à la compréhension du travail effectué. Cette partie a donc pour objectif d'expliquer différents pré-requis.

3.1 Etude de la démarche et cycles de marche

Le projet eGait a pour sujet l'**étude de la démarche** avec pour objectif à long terme la détection précoce des troubles de la marche, avant même que les individus n'expérimentent des symptômes. Cette étude nécessite donc des connaissances sur la démarche humaine, que l'on peut segmenter en cycles de marche.

En effet, lorsqu'un sujet marche plusieurs pas, nous pouvons segmenter sa marche en pas ou en cycle. Un pas est considéré comme le mouvement qui commence lors de la pose d'un pied au sol et qui se termine lorsque l'autre pied se pose au sol. Un **cycle de marche** est la réalisation deux pas (cela est aussi appelé une foulée) : c'est l'ensemble des mouvements réalisés entre deux contacts consécutifs du talon d'un même pied avec le sol.

Figure 3.1: Schéma de pas sur le sol. [1]

Dans ce stage, nous nous interessons aux cycles de marche, et c'est eux que nous voulons segmenter chez les patients.

Nous pouvons définir plus précisément un cycle de marche. Celui-ci se divise en deux phases : la phase d'appui et la phase de balancement. Si nous commençons notre cycle de marche par le pied droit, alors la phase d'appui est celle pendant laquelle le pied droit est en contact avec le sol. Elle dure en moyenne 60% de la durée du cycle de marche complet, avant que le pied droit se soulève du sol, pendant la phase de balancement.

[1] Image issue du Mooc "Le mouvement humain" (F. Hug & T. Deschamps, Nantes Université, 2019).

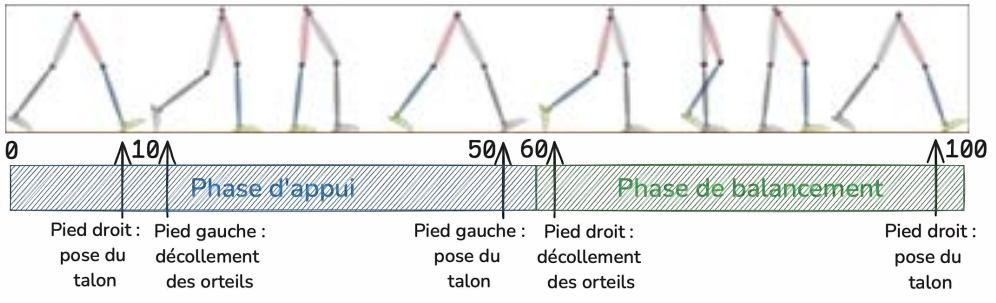


Figure 3.2: Schéma d'un cycle de marche. [2]

Le schéma précédent résume les différents évènements survenant lors d'un cycle de marche. Nous pouvons délimiter un cycle avec quatre points de références :

- La pose du talon au sol du pied droit : c'est le début de la phase d'appui.
- Le décollement du sol du pied gauche.
- La pause du talon au sol du pied gauche.
- Le décollement du sol du pied droit : c'est le début de la phase de balancement.

Ainsi, nous pouvons utiliser un de ces points, comme la pose du talon au sol d'un pied, pour délimiter une marche en différents cycles de marche, ce sur quoi nous reviendrons dans la suite.

3.2 Dispositifs eGait et GAITRite©

Afin d'accéder à des mesures de la démarche des patients, différents dispositifs ont été mis en place. Deux de ces dispositifs ont été employés dans ce stage et je les détaillerai donc dans cette partie.

Le premier dispositif est celui développé par le Laboratoire de Mathématique Jean Leray et s'appelle **eGait**, c'est celui que nous essayons de valider afin de le mettre en usage.

Le dispositif se présente sous la forme d'une centrale inertielle, composée d'un accéléromètre, d'un gyroscope et d'un magnétomètre, alignés sur trois axes orthogonaux. C'est un petit boîtier que les sujets portent à la hanche droite, en l'accrochant à une ceinture. C'est donc un **dispositif portatif**, léger, discret et non invasif.

[2] Schéma annoté à partir de celui de Jaquelin Perry sur Wikipedia.

Figure 3.3: Dispositif eGait porté à la hanche droite à l'aide d'une ceinture.

Le dispositif est relié en bluetooth à un smartphone sur lequel se trouve une application. Cette dernière nous permet de lancer les acquisitions de données de marche, puis de récupérer les données brutes à partir du téléphone. En effet, le capteur transfère toutes les 10 ms son orientation au smartphone qui les enregistre dans un fichier au format `csv`. Le dispositif eGait mesure l'évolution de l'orientation en 3D du capteur au cours du temps. Ainsi, lorsqu'il est placé à la hanche, les données brutes renvoyées par le capteur représentent la rotation de la hanche en 3D des patients, sous la forme de séries temporelles de quaternions unitaires. Ce format de données sera expliqué dans la partie suivante. Ce qu'il est important de retenir est que ce dispositif permet de collecter des **données quantitatives** et **non biaisées**, dans le sens où le capteur peut être porté dans la vie quotidienne, avec l'objectif que le patient oublie qu'il le porte, afin que sa démarche soit celle qu'il adopte réellement dans ses activités quotidiennes.

Le second dispositif est le **tapis de marche GAITRite©** (GAITRite [10]), qui est considéré comme une référence dans l'analyse de la démarche (Menz et al. [22]).

Figure 3.4: Tapis GAITRite© se trouvant à l'hôpital Bellier, au CHU de Nantes.

C'est un tapis mesurant environ 9 mètres, sur lequel les sujets marchent, qui possède de nombreux capteurs mesurant la pression des pieds. Cela nous permet d'avoir directement un résumé de la démarche du patient sous forme de fichiers `xlsx`, car il est relié à un ordinateur qui récupère les données fournies par les capteurs présents dans le tapis.

En revanche, comme ces tapis se situent dans des centres de santé comme des hôpitaux, cela nécessite donc que les patients puissent se déplacer dans une grande ville dans laquelle un hôpital possède ce dispositif. De plus, une fois sur place, les patients sont observés par le personnel hospitalier pendant qu'il marche sur le tapis, ce qui introduit un **biais d'observation**, signifiant que leur démarche ne sera pas forcément représentative de celle qu'ils adoptent dans leur vie quotidienne.

Dans le cadre de mon stage, nous avons pu utiliser le tapis GAITRite© se situant au CHU de Nantes, à l'hopital Bellier, dans le service de gérontologie. Comme le tapis est une référence, nous pouvons donc nous servir des données qu'il renvoie afin de les comparer aux données que nous pouvons obtenir avec le dispositif eGait, afin de valider le dispositif porté par le LMJL.

3.3 Quaternions

Le dispositif eGait mesure la **rotation en 3D de la hanche** au cours du temps. Une rotation de cette forme peut être représentée par plusieurs objets :

- Les matrices de rotation.
- Les angles d'Euler : selon la paramétrisation, on parle d'angles d'Euler ou d'angles de Tait-Bryan ou de paramétrisation Roll-Pitch-Yaw : ils représentent une rotation 3D comme une composition de 3 rotations autour d'axes fixes.
- La représentation axe-angle.
- Les quaternions unitaires.

Cette dernière représentation a été sélectionnée pour le dispositif eGait, notamment car c'est la paramétrisation la plus compressée, contenant seulement 4 valeurs au lieu de 6 pour les matrices de rotation par exemple. Elle permet également d'éviter le problème appelé *gimbal lock* (Wikipedia [26]), qui est la perte d'un degré de liberté lorsque deux des axes du cardan deviennent parallèles, cela bloquant le système dans un espace à deux dimensions et non trois. Ce problème apparait par exemple dans la représentation des angles d'Euler Roll-Pitch-Yaw, lorsque l'angle Pitch est tourné de 90° vers le haut ou le bas, car les angles Yaw et Roll représentent alors le même mouvement.

Les quaternions (décrits par Hamilton [11]) sont des vecteurs à quatre dimensions notés $\mathbf{q} = (q_w, q_x, q_y, q_z)$, mais peuvent aussi être vus comme des nombres hypercomplexes de rang 4. Les **quaternions unitaires** sont de norme 1 et permettent d'encoder une rotation 3D d'angle de rotation $\theta \in [0, 2\pi]$ et d'axe de rotation $\mathbf{u} = (u_x, u_y, u_z) \in S^2$, où S^2 est la 2-sphere, par la formule :

$$\mathbf{q} = \cos\left(\frac{\theta}{2}\right) + (u_x i + u_y j + u_z k)\sin\left(\frac{\theta}{2}\right) \tag{3.1}$$

avec :

- i, j, et k généralisant le nombre imaginaire i, tels que $i^2 = j^2 = k^2 = ijk = -1$.
- $||\mathbf{q}|| = \sqrt{\mathbf{q}\mathbf{q}^t} = 1$.

L'ensemble des quaternions unitaires, noté $\mathbb{H}_u$, est muni de propriétés intéressantes (Drouin [8]). Les quaternions $\mathbf{q}$ et $-\mathbf{q}$ représentent la même rotation. Le groupe est muni d'un élément neutre $\mathbf{q}^{(0)} = (1, 0, 0, 0)$ qui correspond à la rotation unité, tel que $\mathbf{q}\mathbf{q}^{(0)} = \mathbf{q}^{(0)}\mathbf{q} = \mathbf{q}$.

Il est possible d'utiliser la distance géodésique d_g entre deux quaternions $\mathbf{q}_1$ et $\mathbf{q}_2$ pour definir un espace métrique $(\mathbb{H}_u, d_g)$, avec :

$$d_g(\mathbf{q}_1, \mathbf{q}_2) = ||\log(\mathbf{q}_1^{-1}\mathbf{q}_2)|| \tag{3.2}$$

Dans notre cas, l'orientation du capteur est la rotation entre:

- Le référentiel d'origine, ici le référentiel terrestre $R_f = (f_1, f_2, f_3)$.
- Son propre référentiel $R_s = (s_1, s_2, s_3)$ formé par l'accéléromètre, le gyroscope et le magnétomètre.

Nous pouvons observer ces axes sur cette illustration, représentant également l'angle de la rotation θ et l'axe de la rotation $\mathbf{u}$ que nous venons d'évoquer :

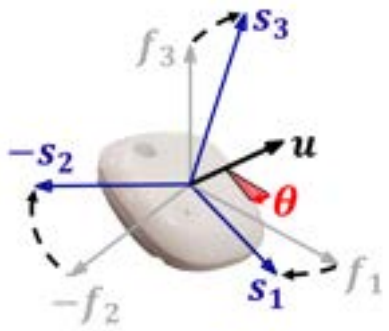


Figure 3.5: Représentation sur le capteur des référentiels ainsi que de l'axe et l'angle de rotation. [3]

Ainsi, nous récupérons grâce au capteur des **séries temporelles de quaternions unitaires**, nous permettant de suivre l'orientation de la hanche du patient au cours du temps lors de sa marche.

[3] Figure tirée de Drouin et al. [9].

3.4 Paramètres spatio-temporels

Après avoir récupéré les données brutes du capteur, nous segmentons le signal en cycles de marche, et cela nous permet ensuite d'obtenir des **paramètres spatio-temporels**. Ce sont des indicateurs pertinents sur la démarche du patient et son état de santé global.

Voici un résumé des paramètres spatio-temporels les plus fréquemment étudiés dans l'étude de la démarche humaine (Drouin [8]) :

- Paramètres temporels (correspondant aux rythmes) :
  - Durée des pas ou des cycles.
  - Nombre de pas ou de cycles effectués.
  - Durée des phases du cycle: phase d'appui et phase de balancement.
- Paramètres spatiaux (correspondant à des distances) :
  - Longueur et largeur du pas.
  - Longueur et largeur du cycle.
  - Hauteurs maximale et minimale du pied durant la phase de balancement.
  - Amplitude de rotation maximale de l'articulation de la hanche, de la cuisse, du tibia et/ou de la cheville au cours du cycle.
- Paramètre spatio-temporel :
  - Vitesse de marche.

Dans notre cas, à partir des séries de quaternions, nous pouvons calculer les paramètres suivants :

- La durée moyenne du cycle de marche.
- La durée moyenne de la phase d'appui.
- La durée moyenne de la phase de balancement.
- L'amplitude moyenne du cycle de marche.
- La vitesse angulaire moyenne du cycle de marche.

De plus, si nous connaissons la distance parcourue par les sujets, nous pouvons facilement calculer la vitesse de marche selon le temps pris pour effectuer cette distance. Ce dernier paramètre est très important, surtout chez les personnes âgées, car il est un réel indicateur de l'état de santé général des patients (Montero-Odasso et al. [23]).

Enfin, la variabilité de la démarche est également un facteur à prendre en compte dans l'analyse de la marche, surtout au sein de la population agée (Annweiler et al. [1]).

4 Matériel

4.1 Acquisitions de données

Dans ce projet, nous utilisons des données provenant de deux dispositifs différents : celles provenant du dispositif portatif eGait, et celles fournies par le tapis de marche GAITRite©.

Nous voulons collecter les données des deux dispositifs simultanément afin d'avoir des données synchronisées. Ainsi, l'acquisition des données s'est révélée moins simple qu'il n'y paraissait au départ. En effet, il a fallu réfléchir à une manière de déclencher la collecte des données sur les deux appareils au même moment, afin de ne pas avoir de décalage dans les données.

Le tapis GAITRite© se déclenche par un ordinateur, tandis que le dispositif eGait se déclenche sur une application sur smartphone. Après plusieurs essais et réflexions, le protocole choisi a été le suivant : une même personne appuie sur les deux dispositifs avec ses deux index. Cette technique permet une synchronisation satisfaisante et reste simple à mettre en place.

Ensuite, afin d'être certains que les données fournies par eGait sont fiables, une personne tenant le téléphone relié en bluetooth au capteur suit le sujet en train de marcher. Cette personne vérifie également que les capteurs (accéléromètre, gyroscope et magnétomètre) restent bien calibrer tout au long des acquisitions.

Avec ce protocole d'acquisition, nous avons pu récolter les données de 77 sessions de marche, sur trois sujets ne présentant pas de troubles de la marche (deux femmes et un homme). Nous avons également récolté des données avec différentes vitesses de marche pour avoir une certaine diversité au sein de nos données. Voici un tableau récapitulant les différentes vitesses de marche acquises :

Vitesse lente	Vitesse intermédiaire	Vitesse normale	Vitesse rapide
49 à 98 cm/s	104 à 128 cm/s	138 à 155 cm/s	178 à 204 cm/s

Table 4.1: Détail des vitesses de marches effectuées pendant les acquisitions.

4.2 Données

Comme évoqué précédemment, nous avons deux sources de données.

Premièrement, le dispositif eGait produit des séries de quaternions unitaires, en produisant un quaternion unitaire toutes les 10 ms. Cela représente la rotation en 3D de la hanche du sujet pendant la marche effectuée. Voici un exemple de série temporelle récoltée, avec les quatre coordonnées des quaternions :

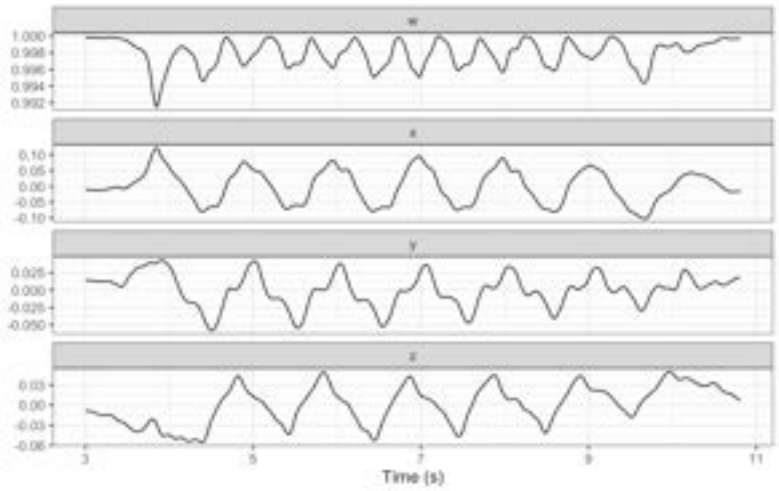


Figure 4.1: Série temporelle de quaternions unitaires récupérées par eGait, sur une session de marche en vitesse normale.

Nous observons sur ces courbes des cycles se répétant et étant très réguliers. Ces cycles représentent les pas effectués par le sujet, et une telle régularité se retrouve chez les sujets sains. Sur des patients présentant des troubles de la marche, les séries sont beaucoup moins régulières, et cela est tout le coeur du problème de segmentation des cycles de marche.

D'autre part, le tapis GAITRite© nous renvoie directement des indicateurs temporels et spatiaux comme la durée ou longueur du pas, ou la vitesse (voir Table 9.3 pour un détail de tous les paramètres récoltés). Les indicateurs nous intéressant tout particulièrement dans cette étude sont les temps de pose des pieds au sol. En effet, GAITRite© fournit, pour chacun des pieds, le moment où le pied est entré en contact avec le sol, ainsi que le moment où le pied s'est décollé du sol, et ce pour chaque pas effectué.

Ainsi, comme nous avons des données synchronisées, nous pouvons associer les temps de pose des pieds au sol à nos séries temporelles de quaternions. Voici un exemple de données récupérées par eGait sur lesquelles nous avons superposé les indicateurs de pose de pieds fournis par GAITRite© :

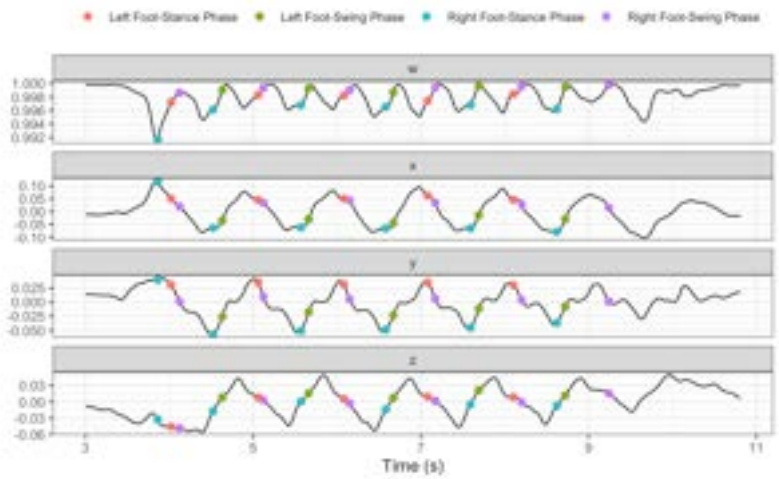


Figure 4.2: Superposition des points de contact donnés par GAITRite© sur les données eGait.

Ainsi, pour chaque temps des données produites par eGait, nous pouvons associer l'information de pose du pied. Cela nous permet d'avoir des données sur lesquelles un modèle d'apprentissage statistique peut apprendre, grâce à cette information fournit par le tapis de marche.

Plus précisément, pour segmenter le signal eGait en cycles de marche, nous allons vouloir prédire le temps de contact au sol du pied droit. C'est ce temps qui délimitera nos cycles de marche au sein du signal. Nous allons donc classer les temps de marche en deux catégories :

- `Right_stance`: Le pied droit vient de toucher le sol. Cela correspond au début de la phase d'appui.
- `None`: Autres temps de marche.

Nous pouvons noter que nous avons choisi d'utiliser les temps associés au pied droit car le capteur eGait se positionne sur la hanche droite des sujets.

Une caractéristique cruciale des données est le **déséquilibre de classes** présent. En effet, nous avons des proportions très différentes entre notre classe d'intérêt `Right_Stance` et la seconde classe :

Classe	Nombre d'observations	Proportion
`Right_Stance`	448	0.006
`None`	73981	0.994

Table 4.2: Proportion d'observations dans chacune de nos classes.

Cette particularité doit impérativement être prise en compte pour construire un modèle de machine learning sur nos données. Nous reviendrons dans la suite sur les méthodes mises en place pour gérer cette problématique.

4.3 Preprocessing des données eGait

Avant d'utiliser nos séries temporelles de quaternions dans notre modèle, nous effectuons une étape de pré-traitement sur ces données, appelée *centring*. Cela permet de centrer les séries de quaternions autour d'une moyenne, dont le calcul est spécifique à ce type de données (Le Gall et al. [18]).

Nous avons n observations de séries temporelles de quaternions unitaires, notées $\mathbf{Q}_1, \dots, \mathbf{Q}_n$, que l'on observe sur la même grille de temps $t_1, \dots, t_p$. On a donc $\mathbf{Q}_i(t_k) = \mathbf{q}_{ik} \in \mathbb{H}_u$, avec $i \in [\![1, n]\!]$ et $k \in [\![1, p]\!]$.

Pour calculer la série temporelle de quaternions moyenne, on calcule la moyenne des quaternions $\mathbf{q}_{1k}, \dots, \mathbf{q}_{nk}$ à chaque temps t_k de la grille initiale d'observations. Cette moyenne est appelée moyenne de Fréchet et est associée à la distance géodésique entre deux quaternions unitaires (cette distance a été définie dans la Section 3.3, Équation 3.2).

$$\mathbf{q}_k^{(m)} = \mathbf{Q}^{(m)}(t_k) = \underset{q \in \mathbb{H}_u}{\operatorname{argmin}} \sum_{i=1}^{n} d_g^2(\mathbf{q}_{ik}, \mathbf{q}), \qquad k \in [\![1, p]\!] \tag{4.1}$$

Ainsi, $\mathbf{Q}^{(m)}$ définie la série de quaternions moyenne par rapport aux n séries. On peut ensuite calculer les séries centrées grâce à cette moyenne.

$$\mathbf{q}_{ik}^{(c)} = \mathbf{Q}_i^{(c)}(t_k) = \left(\mathbf{q}_k^{(m)}\right)^{-1} \mathbf{q}_{ik}, \qquad k \in [\![1, p]\!], \;\; i \in [\![1, n]\!] \tag{4.2}$$

On a donc $\mathbf{Q}_1^{(c)}, \dots, \mathbf{Q}_n^{(c)}$ les séries de quaternions centrées après pré-traitement, qui peuvent désormais être utilisées dans notre modèle.

D'autre part, comme on ne peut pas calculer une dérivée de série temporelle, on doit passer à une représentation fonctionnelles en splines cubiques (J. O. Ramsay [15]). Cela nous permettra ensuite de pouvoir calculer des paramètres utilisés dans notre feature space.

4.4 Feature space

Pour implémenter un modèle d'apprentissage statistique, nous construisons un **feature space** contenant différentes variables sur lesquelles notre modèle va apprendre.

A partir des séries temporelles de quaternions unitaires de chacune des sessions de marche acquises, nous extrayons des grandeurs permettant de caractériser le mouvement de rotation : la **vitesse angulaire** ainsi que l'**accélération angulaire**. Si on suppose que l'on peut calculer les dérivées premières et secondes d'une série temporelle de quaternions par rapport au temps, ces variables sont calculées comme suit (Narayan [24]).

On note $\mathbf{\Omega}$ la vitesse angulaire par rapport au référentiel du corps, c'est-à-dire que le référentiel est fixé au corps en mouvement. $\mathbf{\Omega}$ est un vecteur qui a pour direction l'axe de rotation et pour grandeur la vitesse angulaire. On a :

$$\dot{\mathbf{q}} = \frac{d\mathbf{q}}{dt} = \frac{1}{2}\mathbf{q}\,\mathbf{\Omega} \tag{4.3}$$

Comme $\mathbf{\Omega}$ est le vecteur de la vitesse angulaire, il est vu comme un quaternion avec une partie scalaire nulle. De plus, c'est un vecteur unitaire. On peut donc écrire :

$$\mathbf{\Omega} = 2\mathbf{q}^{-1}\dot{\mathbf{q}} \tag{4.4}$$

Nous calculons également l'accélération angulaire, notée $\dot{\mathbf{\Omega}}$, qui est la dérivée de la vitesse angulaire.

$$\ddot{\mathbf{q}} = \frac{d^2\mathbf{q}}{dt^2} = \frac{1}{2}\left(\dot{\mathbf{q}}\,\mathbf{\Omega} + \mathbf{q}\,\dot{\mathbf{\Omega}}\right) \tag{4.5}$$

Il s'agit de nouveau d'un vecteur unitaire et on a donc :

$$\dot{\mathbf{\Omega}} = 2\left(\mathbf{q}^{-1}\ddot{\mathbf{q}} - (\mathbf{q}^{-1}\dot{\mathbf{q}})^2\right) \tag{4.6}$$

De plus, nous pouvons calculer les **angles d'Euler** appelés Roll, Pitch, et Yaw, qui représentent les rotations autour des trois principaux axes, comme représentés sur cette illustration :

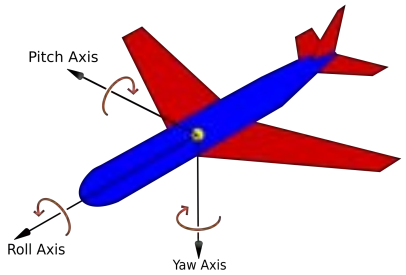


Figure 4.3: Schéma des mouvements Roll, Pitch et Yaw sur un avion en vol.[1]

Comme les quaternions unitaires, ces angles représentent une rotation en trois dimensions, et nous pouvons convertir nos séries de quaternions unitaires en séries d'angles Roll-Pitch-Yaw (Wikipedia [25]).

En effet, on peut associer un quaternion à une rotation autour d'un axe avec les expressions suivantes :

$$\begin{aligned} q_w &= \cos(\text{rotation angle}/2) \\ q_x &= \sin(\text{rotation angle}/2)\cos(\text{angle entre axe de rotation et axe } x) \\ q_y &= \sin(\text{rotation angle}/2)\cos(\text{angle entre axe de rotation et axe } y) \\ q_z &= \sin(\text{rotation angle}/2)\cos(\text{angle entre axe de rotation et axe } z) \end{aligned} \tag{4.7}$$

Cela nous amène à la matrice de rotation suivante pour passer du quaternion $\mathbf{q} = (q_w, q_x, q_y, q_z)$ aux angles d'Euler Roll, Pitch et Yaw :

$$\begin{bmatrix} \text{Roll} \\ \text{Pitch} \\ \text{Yaw} \end{bmatrix} = \begin{bmatrix} \text{atan2}\left(2(q_w q_x + q_y q_z), 1 - 2(q_x^2 + q_y^2)\right) \\ \text{asin}\left(2(q_w q_y - q_x q_z)\right) \\ \text{atan2}\left(2(q_w q_z + q_x q_y), 1 - 2(q_y^2 + q_z^2)\right) \end{bmatrix} \tag{4.8}$$

avec $\text{atan2}(y, x) = \arctan(y/x)$.

D'autre part, nous avions observé sur les graphiques des séries temporelles que les points de contact donnés par GAITRite© ne se situaient pas forcément aux mêmes endroits selon les **vitesses de marche**. Ce phénomène se remarquait sur les marches en vitesse lente, et nous pourrions émettre l'hypothèse qu'il serait causé par une démarche peu naturelle lorsque les sujets se forcent à marcher très lentement. Pour autant, pour être sûr de ne pas manquer d'informations, nous avons décidé d'ajouter également la vitesse de marche à notre feature space, à partir des données fournies par le tapis GAITRite©.

[1] Figure tirée de Wikipedia : Yaw Axis Corrected.svg.

Ainsi, pour résumer, notre feature space contient les variables suivantes :

- La vitesse angulaire : `vx`, `vy`, `vz`.
- L'accélération angulaire : `ax`, `ay`, `az`.
- Les angles d'Euler Roll-Pitch-Yaw : `roll`, `pitch`, `yaw`.
- La vitesse de marche : `speed`.

Ce feature space constitue les données qui seront fournies en entrée de notre modèle de machine learning. De plus, notre feature space contient deux hyperparamètres :

- `spar`, un paramètre de lissage de la courbe de notre série temporelle de quaternions. En effet, comme nous obtenons la vitesse et l'accélération angulaire en dérivant notre série, il peut être intéressant de lisser notre courbe initiale.
- `n_lag`, définissant le nombre de temps à conserver dans le passer pour les variables.

Par conséquent, le nombre de variables de notre feature space varie en fonction du paramètre `n_lag`. Plus précisément, il contiendra $10 + 9\times$ `n_lag` variables. Voici une illustration de notre feature space avec un paramètre de 1 pour le lag, contenant donc 19 variables :

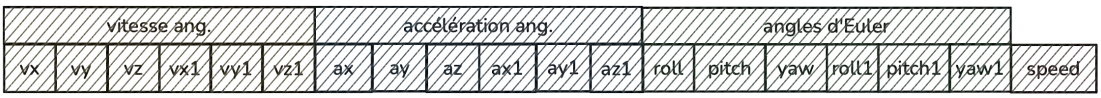


Figure 4.4: Illustration du feature space avec `n_lag`=1.

Enfin, notre feature space contient les classes `Right_Stance` et `None` évoquées précédemment, récupérées grâce au dispositif GAITRite© et associées aux temps de nos sessions. Comme nous classons des temps sur une fréquence de 100 Hz, les observations consécutives d'une session correspondent à des évènements très proches. Ainsi, nous décidons de prendre en compte une certaine incertitude et de labelliser comme observations `Right_Stance` pas uniquement le temps exact donné par GAITRite©, mais également k points autour de celui-ci. En prenant $k = 3$, cela correspond à une fenêtre de 70 ms, avec 7 points labellisés `Right_Stance`.

Figure 4.5: Illustration de la labellisation d'une plage de 6 points autour du point donné par GAITRite©.

Cette stratégie permet également d'avoir un déséquilibre un peu moins fort entre nos classes, comme nous rajoutons des observations dans notre classe minoritaire.

5 Méthodes statistiques

5.1 Modèles de classification binaire

De nombreux modèle d'apprentissage statistique existent pour classer nos données en deux classes. Nous allons ici détailler le fonctionnement des principaux utilisés dans ce projet, en nous basant sur l'ouvrage de référence des modèles implémentés dans le package {`tidymodels`} de `R` : *Applied Predictive Modeling* de Max Kuhn et Kjell Johnson (2013).

5.1.1 Régression logistique

Pour introduire la **régression logistique**, nous pouvons rappeler que la régression linéaire minimise la somme des carrés résiduels, définies ainsi :

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{5.1}$$

En minimisant cette quantité, nous produisons également un modèle qui produit l'estimateur du maximum de vraisemblance du paramètre (sous l'hypothèse que les résidus suivent une loi Normale). Cet estimateur, en maximisant la fonction de vraisemblance, est celui qui concorde le plus étroitement avec les données observées (David W. Hosmer [7]).

Quand nous avons deux classes au sein de nos données, la distribution de probabilité la plus souvent utilisée est la distribution Binomiale de paramètre p, avec p la probabilité d'une des classes. Si on a $y \sim B(p, n)$, alors la fonction de vraisemblance est de la forme suivante :

$$L(p, y) = \binom{n}{y} p^y (1-p)^{n-y} \tag{5.2}$$

Toutefois, nous voulons construire un modèle qui se base sur nos variables, donc nous allons re-paramétriser le modèle pour que p soit une fonction de ces variables.

Ainsi, comme pour la régression linéaire, nous allons avoir un *intercept* β_0 et des coefficents β_j associés aux variables. Comme p est une probabilité, on a $p \in [0, 1]$, il faut donc garantir que le modèle va contraindre les valeurs dans cet intervalle. Si nous avons p la probabilité d'un évènement, nous appelons côte de l'évènement la valeur $p/(1-p)$. Nous modélisons le logarithme de cette valeur comme une fonction linéaire :

$$\log\left(\frac{p}{1-p}\right) = \beta_0 + \beta_1 x + \cdots + \beta_P x_P \tag{5.3}$$

avec P le nombre de variables. Cela nous permet d'obtenir la formule suivante :

$$p = \frac{\exp^{\beta_0+\beta_1 x+\cdots+\beta_P x_P}}{1+\exp^{\beta_0+\beta_1 x+\cdots+\beta_P x_P}} \tag{5.4}$$

Ainsi, nous relions notre modèle au paramètre de la distribution Binomiale, et pouvons trouver des candidats pour nos coefficients et calculer la fonction de vraisemblance. On sélectionne ensuite les coefficients maximisant cette fonction, qui seront utilisés pour calculer les prédictions.

5.1.2 Arbre de décision

Les **arbres de décision** consistent en une suite imbriquée de déclarations `if-then` pour le prédicteur qui partitionne les données. Les données sont ainsi attribuées aux classes selon les valeurs prises par les variables. Voici un exemple d'une suite de condition et de sa représentation en arbre :

```
if VarA > 0.15 then
    if VarB > 0.20 then class = 1
    else class = 2
else class = 2
```

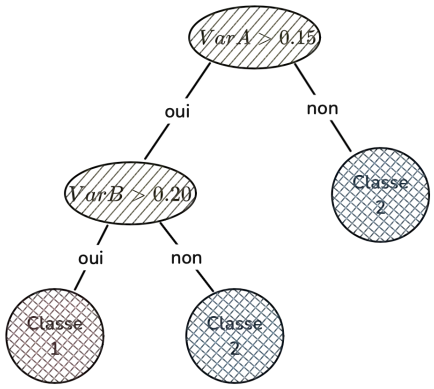


Figure 5.1: Exemple d'arbre de décision.

Nous avons ici un arbre à deux séparations, menant à trois noeuds terminaux, aussi appelés feuilles. Ce partionnement forme des régions rectangulaires au sein des observations.

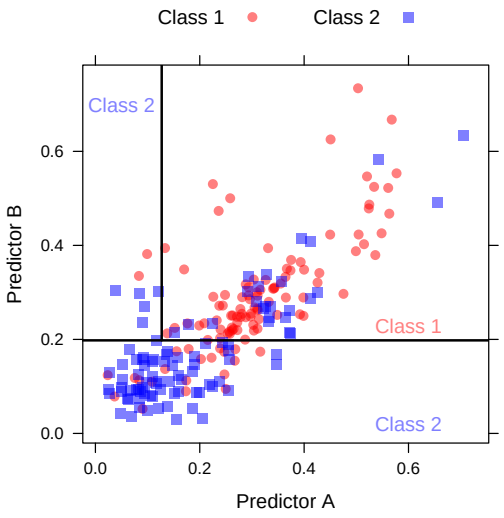


Figure 5.2: Exemple des régions définis par un modèle d'arbre de décision. [1]

Le but de ce modèle est de classer les observations en groupes plus petits et homogènes. On définit ici l'homogénéité comme le fait que les noeuds de la séparation soient "pures", c'est à dire qu'il y ait une plus grande proportion d'une des classes dans chaque noeud. Pour calculer cette "pureté", nous pouvons utiliser l'**indice de Gini** (Breiman [5]). Nous nous plaçons de nouveau dans le cas où nous avons deux classes.

L'indice de Gini

L'indice de Gini pour un certain noeud est :

$$\text{Gini} = p_1(1 - p_1) + p_2(1 - p_2) \tag{5.5}$$

avec p_1 et p_2 les probabilités des classes 1 et 2 respectivement.
Comme on a $p_1 + p_2 = 1$, on peut ré-écrire l'indice sous la forme suivante :

$$\text{Gini} = 2p_1p_2 \tag{5.6}$$

Nous voyons que cet indice est minimisé lorsqu'une des deux probabilités p_1 ou p_2 tend vers 0, cela signifie que le noeud est "pure" car nous avons donc une classe peu présente dans le noeud créé. A l'inverse, l'indice sera maximisé quand $p_1 = p_2$, et le noeud sera moins "pure".

[1] Figure tirée de (Kuhn [16]).

Le processus pour trouver la séparation optimale maximisant l'indice de Gini est le suivant. Tout d'abord, nous rangeons les observations selon les valeurs prises par la variable. Les points de séparations possibles sont les points médians entre chaque valeur unique prise par la variable. A chaque point de séparation, nous obtenons une table de contingence comme celle-ci :

	Classe 1	Classe 2	
> séparation	n_{11}	n_{12}	n_{+1}
<= séparation	n_{21}	n_{22}	n_{+2}
	n_{1+}	n_{2+}	n

Figure 5.3: Table de contingence pour le calcul de l'indice de Gini.

Ainsi, l'indice de Gini avant la séparation vaut :

$$\text{Gini}_{\text{ avant séparation}} = 2\left(\frac{n_{1+}}{n}\right)\left(\frac{n_{2+}}{n}\right) \tag{5.7}$$

On peut ensuite calculer l'indice après la séparation au sein de chaque nouveau noeud. Pour le côté supérieur à la séparation, il vaut $2\left(\frac{n_{11}}{n_{+1}}\right)\left(\frac{n_{12}}{n_{+1}}\right)$, tandis que pour le coté inférieur ou égal à la séparation, il vaut $2\left(\frac{n_{21}}{n_{+2}}\right)\left(\frac{n_{22}}{n_{+2}}\right)$.

Pour construire un indice global de l'après séparation, on place des poids correspondant à la proportion d'observations dans chaque partie de la séparation, valant respectivement $\left(\frac{n_{+1}}{n}\right)$ et $\left(\frac{n_{+2}}{n}\right)$. On obtient ainsi la formule suivante :

$$\text{Gini}_{\text{ après séparation}} = 2\left[\left(\frac{n_{11}}{n}\right)\left(\frac{n_{12}}{n_{+1}}\right) + \left(\frac{n_{21}}{n}\right)\left(\frac{n_{22}}{n_{+2}}\right)\right] \tag{5.8}$$

Si nous reprenons notre exemple précédent, nous obtenons la table de contingence suivante pour la variable B au point de séparation 0.20:

	Classe 1	Classe 2	
$VarB > 0.20$	91	30	121
$VarB \leq 0.20$	20	67	87
	111	97	208

Figure 5.4: Exemple de table de contingence pour le calcul de l'indice de Gini.

Nous obtenons un indice de 0.373 pour la séparation $VarB > 0.20$ et de 0.197 pour la séparation $VarB \leq 0.20$. Puis nous combinons ces valeurs avec les poids (valant respectivement 0.582 et 0.418) pour obtenir l'indice global après séparation, valant 0.365.

En pratique, le modèle va évaluer chaque point de séparation pour sélectionner celui minimisant l'indice de Gini. Ensuite, l'arbre se construit en continuant d'effectuer des séparations ainsi, jusqu'à ce qu'on atteigne un critère d'arrêt, ce qui est la plupart du temps une profondeur maximale indiquée.

Dans les arbres CART implémentés sous `R` dans le package {`parsnip`}, grâce à l'engine `rpart`, il y a trois paramètres que nous pouvons tuner :

Paramètre	Description	Valeur par défaut
`tree_depth`	Profondeur maximale de l'arbre.	30
`min_n`	Nombre minimal d'observations dans un noeud pour qu'il soit divisé.	2
`cost_complexity`	Paramètre de coût-complexité.	0.01

Table 5.1: Résumé des paramètres à tuner dans le modèle d'arbre de décision CART du package {`parsnip`}.

5.1.3 Bagged trees et forêt aléatoire

Le modèle de **bagged trees** (*bagging* étant l'abréviation de *bootstrap aggregation*) est un ensemble d'arbres de décision. L'algorithme pour construire le modèle est le suivant.

Algorithm 1 Bagging

1: **for** $i = 1$ to M **do**
2: Générer un échantillon bootstrappé des données originales
3: Entrainer un modèle d'arbre non élagué sur cet échantillon
4: **end for**

Un échantillon **bootstrappé** est un échantillon de même taille que les données originales, obtenu en sélectionnant des observations avec remise. Cela signifie qu'une fois qu'une observation est sélectionnée, elle reste toujours disponible pour être de nouveau sélectionnée dans l'échantillon. Nous pouvons donc avoir dans ce nouvel ensemble des observations apparaissant de nombreuses fois, et des observations qui ne seront jamais sélectionnées.

Chacun de ces M arbres de décision est utilisé pour générer une prédiction pour une nouvelle observation, ce que nous pouvons voir comme un vote pour une des classes. Ensuite, l'observation est classée dans la classe ayant récoltée le plus de votes au sein des arbres.

La **forêt aléatoire** est un modèle assez similaire, avec une différence que nous voyons dans son algorithme :

Algorithm 2 Random Forests

for $i = 1$ to M **do**
    Générer un échantillon bootstrappé des données originales
    Entrainer un modèle d'arbre sur cet échantillon
    for each split **do**
        Sélectionner aléatoirement k variables
        Sélectionner le meilleur prédicteur parmis ces k variables et partitionner les données
    end for
end for

De nouveau, chaque arbre vote pour une des classes et nous classons l'observation selon la majorité. Cette fois, les arbres ne sont pas forcément complets et la profondeur peut être déterminée.

Sous `R`, dans le package {`parsnip`}, les bagged trees sont implémentés avec l'*engine* `rpart`, et les forêts aléatoires avec l'*engine* `ranger`. Voici les paramètres que nous pouvons tuner dans ces modèles :

Paramètre	Description	Valeur par défaut
`tree_depth`	Profondeur maximale de l'arbre.	30
`min_n`	Nombre minimal d'observations dans un noeud pour qu'il soit divisé.	2
`cost_complexity`	Paramètre de coût-complexité.	0.01
`class_cost`	Coût sur les classes.	1

Table 5.2: Résumé des paramètres à tuner dans le modèle de bagged trees du package {`parsnip`}.

Paramètre	Description	Valeur par défaut
`trees`	Nombre d'arbres dans la forêt.	500
`min_n`	Nombre minimal d'observations dans un noeud pour qu'il soit divisé.	10
`m_try`	Nombre de variables aléatoirement sélectionnées (k dans Algorithm 2)	`floor(sqrt(ncol))`

Table 5.3: Résumé des paramètres à tuner dans le modèle de forêt aléatoire du package {`parsnip`}.

5.2 Déséquilibre de classes et algorithmes d'échantillonnage

La situation dans laquelle une classe est fortement **sous-représentée** au sein de données est courante dans les applications concrètes de nombreux domaines. Pour autant, dans la grande majorité des cas, les algorithmes de machine learning supposent que nous leur fournissons des distributions équilibrées de classes. Ainsi, quand ce n'est pas le cas, ils n'arrivent pas à représenter correctement les caractéristiques des données, l'entraînement se concentre sur la prédiction de la classe majoritaire. Nous présentons dans cette partie des algorithmes populaires proposant des solutions à ce problème (He et Garcia [13]).

Dans cette partie, nous introduisons les notations suivantes :

- S désigne notre échantillon d'apprentissage, tel que $S = \{(x_i, y_i)\}$ avec x_i une instance de notre feature space, et y_i la classe associée à cette observation. Ici, $y_i \in \{1, 2\}$.
- S_{min} est l'ensemble des observations de la classe minoritaire.
- S_{maj} est l'ensemble des observations de la classe majoritaire.

Nous nous concentrons dans ce rapport sur les méthodes dites d'échantillonnage, qui font référence à la modification de données déséquilibrées par certains mécanismes dans l'objectif de créer une distribution équilibrée.

5.2.1 Sur-échantillonnage et sous-échantillonnage aléatoires

Les algorithmes les plus simples de cette catégorie sont ceux de sur-échantillonnage et sous-échantillonnage aléatoires.

Dans le cas du **sur-échantillonnage aléatoire** (*random oversampling*), nous sélectionnons aléatoirement un ensemble de notre classe minoritaire S_{min}. Ensuite, nous augmentons les données originales S avec ces observations sélectionnées en les répliquant et en les ajoutant à S.

A l'inverse, dans le **sous-échantillonnage aléatoire** (*random undersampling*), nous sélectionnons aléatoirement un ensemble d'observations de notre classe majoritaire S_{maj}, puis nous les retirons de nos données originales S.

Nous voyons que ces deux méthodes sont simples à mettre en place et permettent de varier le degré d'équilibre de nos classes, en retirant ou en ajoutant plus ou moins d'observations. En revanche, elles apportent chacune certaines problématiques. En effet, dans le cas du sous-échantillonnage, retirer aléatoirement des observations peut entraîner une perte d'informations importante dans notre classe majoritaire. Dans le cas du sur-échantillonnage, comme nous ajoutons des observations répliquées à nos données, nous avons plusieurs fois les mêmes instances et cela peut conduire à du sur-apprentissage. Cela implique que notre modèle n'arrivera pas à classer de nouvelles données qu'il n'a pas utilisé lors de son ajustement.

5.2.2 Algorithme SMOTE (*Synthetic Sampling with Data Generation*)

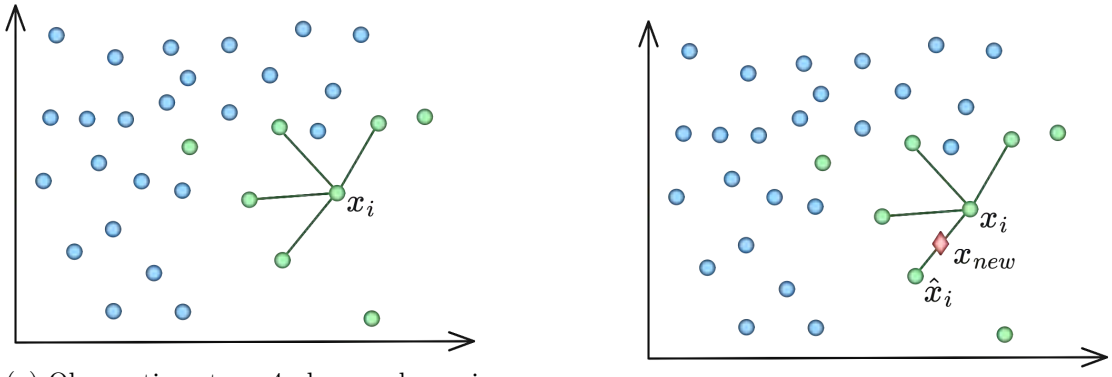
L'algorithme **SMOTE** (Chawla et al. [6]) créé de nouvelles **observations synthétiques** dans notre classe minoritaire, une fois de plus afin de ré-équilibrer nos classes. La création des nouvelles observations se fait de la manière suivante.

Pour chaque observation x_i de notre classe minoritaire S_{min}, on considère les K plus proches voisins de x_i appartement à S_{min}, selon la distance euclidienne, pour un entier donné K. Une fois ces K plus proches voisins identifiés, on en sélectionne un aléatoirement, que l'on note $\hat{x}_i$, et on créé une observation selon la formule suivante :

$$x_{new} = x_i + (\hat{x}_i - x_i)\ \delta \tag{5.9}$$

avec $\delta \in [0, 1]$ un nombre aléatoire.

Cela signifie que la nouvelle observation se situe sur le segment reliant l'observation originale et le plus proche voisin aléatoirement sélectionné, comme nous l'observons sur les schémas suivants :



(a) Observation et ses 4 plus proches voisins.

(b) Nouvelle observation créée.

Figure 5.5: Illustration de la création d'une observation par l'algorithme SMOTE, avec $K = 4$ plus proches voisins.

Les points bleus représentent les observations de la classe majoritaire S_{maj}, tandis que les points verts représentent celles de la classe minoritaire S_{min}. Nous observons que la nouvelle observation, représentée par le losange rouge, se situe entre l'observation x_i et son plus proche voisin sélectionné $\hat{x}_i$.

Ce processus est répété jusqu'à arriver à l'équilibre voulu entre les classes. Cela signifie que l'on créé autant de données synthétiques pour chacune de nos observations originales de la classe minoritaire, sans prendre en compte le voisinage de ces observations, ce qui peut augmenter le risque de chevauchement entre les classes.

5.2.3 Algorithme ADASYN (*Adaptative Synthetic Sampling*)

Pour contrer la limitation de l'algorithme SMOTE, des algorithmes adaptatifs ont été proposés, comme l'algorithme **ADASYN** (He et al. [14]). Le principe de cette méthode est de créer différentes quantités de données synthétiques selon la distribution de nos observations.

Tout d'abord, on calcule le nombre total de données synthétiques que nous devons générer pour la classe minoritaire, selon l'équilibre que nous voulons obtenir entre nos classes. Nous notons ce nombre G :

$$G = (|S_{maj}| - |S_{min}|)\,\beta \tag{5.10}$$

avec $\beta \in [0, 1]$ le paramètre qui nous permet de spécifier l'équilibre voulu.

Ensuite, pour chaque x_i dans notre classe minoritaire S_{min}, on trouve ses K plus proches voisins selon la distance euclidienne, et on calcule le ratio suivant :

$$\Gamma_i = \frac{\delta_i / K}{Z}, \qquad i = 1, \ldots, |S_{min}| \tag{5.11}$$

avec :

- δ_i le nombre d'observations parmi les K plus proches voisins qui appartiennent à la classe majoritaire S_{maj}.
- Z une constante de normalisation, afin que Γ_i soit une fonction de distribution, c'est à dire afin d'avoir $\sum_i \Gamma_i = 1$.

Cela signifie que, comparé à l'algorithme SMOTE, nous ne prenons pas en compte uniquement les voisins appartenant à S_{min} :

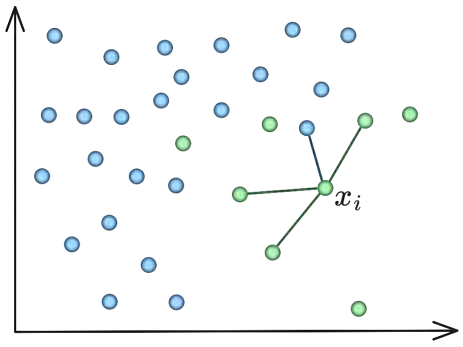


Figure 5.6: Illustration de $K = 4$ plus proches voisins avec l'algorithme ADASYN.

Dans cet exemple, nous avons $K = 4$, $\delta_i = 1$, et donc $\Gamma_i = \frac{1/4}{Z}$.

Une fois cette étape effectuée, pour chaque x_i dans S_{min}, on détermine le nombre de données synthétiques que nous allons générer pour cette observation :

$$g_i = \Gamma_i \, G \tag{5.12}$$

Il ne nous reste plus qu'à générer g_i nouvelles observations pour x_i, de la même manière que pour l'algorithme SMOTE, selon l' Équation 5.9.

Ainsi, cet algorithme génère de nouvelles données dans la classe minoritaire en prenant en compte la distribution de nos données, ce qui permet d'obtenir des données synthétiques représentant davantage nos données.

5.2.4 Méthodes sous `R`

Pour utiliser ces algorithmes sous `R` au sein de l'architecture {`tidymodels`}, il existe des étapes que nous pouvons incorporer à nos {`recipes`}, implémentées dans le package {`themis`}.

Algorithme	Fonction `R`	Paramètres
Sur-échantillonnage	`step_upsample()`	• `over_ratio` : fréquence de sur-échantillonage
Sous-échantillonnage	`step_downsample()`	• `under_ratio` : fréquence de sous-échantillonage
SMOTE	`step_smote()`	• `over_ratio` : fréquence de sous-échantillonage • `neighbors` : nombre de K plus proches voisins
ADASYN	`step_adasyn()`	• `over_ratio` : fréquence de sous-échantillonage • `neighbors` : nombre de K plus proches voisins

Table 5.4: Récapitulatif des fonctions implémentées dans le package

Malheureusement, même si ils ont été mis en place, ces algorithmes n'ont pas été très performants sur nos données, et nous n'avons donc pas choisi de conserver cette approche pour construire un modèle de machine learning sur nos données déséquilibrées.

5.3 Déséquilibre de classes et poids sur les classes

Dans le cas d'un **déséquilibre de classes**, le modèle apprend des informations adéquates sur la classe majoritaire, mais n'a pas assez d'informations sur la classe minoritaire. Cela implique beaucoup de mauvaises prédictions sur la classe minoritaire.

Pour contrer ce phénomène, nous pouvons mettre un poids plus élevé à la classe minoritaire, ce qui permet au modèle d'accorder plus d'attention aux patterns de cette classe. Plus précisément, le poids donné à chaque observation spécifie à quel point chaque observation influence l'estimation du modèle (Kuhn [17]). Pour que le modèle soit biaisé en faveur des observations considérées comme plus importantes, dans notre cas celles appartenant à la classe minoritaire, les poids des observations sont intégrés dans la fonction de coût. Cela permet de réguler le coût de mauvaise classification, dans le sens où mal classifier des observations plus importantes sera plus coûteux, incitant le modèle à éviter cette situation (Hashemi et Karimi [12], Lumley [20]).

Les poids les plus souvent placés sur les classes sont ceux venant de la méthode *inverse class frequency* (ce sont par exemples les poids implémentés par défaut pour les modèles en `python`).

Méthode *Inverse class frequency*

Le poids attribué à la classe j est le suivant :

$$\omega_j = \frac{n}{n_{\text{classes}}\, n_j} \tag{5.13}$$

avec :

- n le nombre total d'observations.
- n_{classes} le nombre de classes.
- n_j le nombre d'observation dans la classe j.

Cette méthode permet de placer des poids équilibrés sur les classes, mais lorsque nous avons un fort déséquilibre de classes, nous pouvons placer des poids manuellement ou bien tuner les poids, ce que nous ferons pour notre modèle.

Placer des poids sur nos classes a significativement amélioré nos modèles et nous avons donc choisi de mettre en place cette stratégie.

Comme nous l'avons résumé dans le tableau suivant, avec les {`tidymodels`} de `R`, nous ne pouvons pas placer des poids sur tous les modèles, ce qui a donc restreint notre choix de modèles. Par exemple, nous n'avons pas pu utilisé la méthode des k plus proches voisins ou les machines à vecteur de support.

Modèle	Fonction `R`	*Engine*	Poids autorisés
Bagged trees	`bag_tree`	`rpart`	Oui
Boosted trees	`boost_tree`	`xgboost`	Oui
Arbre de décision	`decision_tree`	`rpart`	Oui
Régression logistique	`logistic_reg`	`glm`	Oui
Régression logistique	`logistic_reg`	`glmnet`	Oui
Perceptron multicouche	`mlp`	`keras`	Non
k plus proches voisins	`nearest_neighbor`	`kknn`	Non
Forêt aléatoire	`rand_forest`	`ranger`	Oui
Machine à vecteur de support	`svm_rbf`	`kernlab`	Non

Table 5.5: Récapitulatif des modèles acceptant les poids sur les classes.

5.4 Métriques pour la classification binaire

Pour évaluer nos modèles, nous devons utiliser des **métriques** adaptés à notre modèle de classification. Dans une classification binaire, nous avons accès à une **matrice de confusion**, définie comme suivant :

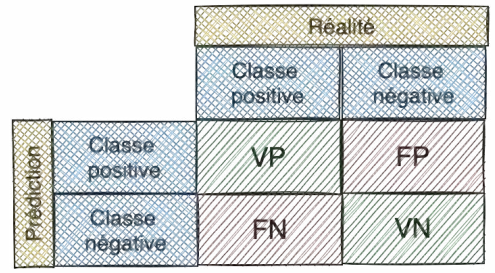


Figure 5.7: Matrice de confusion pour une classification binaire.

Elle permet de classer nos prédictions en différentes catégories. Par exemple, les Vrais Positifs (VP) correspondent à nos prédictions positives correctement classées, tandis que les Faux Négatifs (FN) sont les prédictions classées négativement alors que l'observation appartenait à la classe positive.

Une des métriques la plus employée dans des tâches de classification est l'accuracy, elle est définit ainsi :

$$\text{accuracy} = \frac{\text{prédictions correctes}}{\text{toutes predictions}} = \frac{\text{VP} + \text{VN}}{\text{VP} + \text{FP} + \text{VN} + \text{FN}} \tag{5.14}$$

En revanche, dans le cas d'un déséquilibre de classes, nous ne pouvons pas utiliser cette métrique. En effet, l'accuracy sera toujours haute car le modèle détectera beaucoup de points dans notre classe majoritaire, et le score ne prendra pas en compte les oublis de détection de notre classe minoritaire. Par exemple, si nous obtenons la matrice de confusion suivante, en notant que la classe positive est notre classe minoritaire :

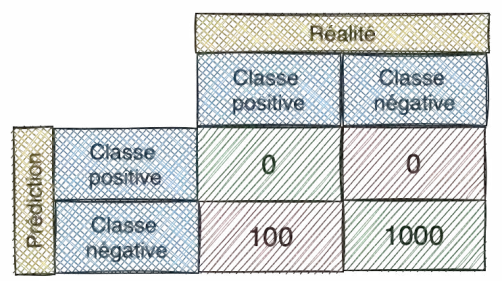


Figure 5.8: Exemple de matrice de confusion dans un cas d'un déséquilibre de classes.

Nous voyons bien que le modèle n'a détecté aucun point de notre classe minoritaire, mais pour autant l'accuracy vaut $\frac{1000}{1000+100} = 0.91$. Cela est un bon score alors que notre modèle n'arrive pas à détecter la classe nous interessant le plus.

Nous devons donc utiliser d'autres métriques nous donnant des informations plus précises sur nos prédictions. Voici les métriques classiques que nous pouvons utiliser dans notre cas :

Le Rappel (ou Sensibilité)

Le **rappel** se définit comme suivant :

$$\text{Rappel} = \frac{\text{VP}}{\text{VP+FN}} \tag{5.15}$$

Il permet de mesurer à quel point le modèle a trouvé toutes les instances de notre classe positive.

La Précision

La **précision** se définit comme suivant :

$$\text{Précision} = \frac{\text{VP}}{\text{VP+FP}} \tag{5.16}$$

Elle permet de mesurer à quelle fréquence le modèle a prédit correctement la classe positive.

La Spécificité

La **spécificité** se définit comme suivant :

$$\text{Spécificité} = \frac{\text{VN}}{\text{VN+FP}} \tag{5.17}$$

Elle permet de mesurer si nous avons correctement prédit la classe négative.

Dans notre cas, nous voulons prédire correctement notre classe positive `Right_Stance`, donc le rappel nous intéresse tout particulièrement. Au contraire, nous pouvons être plus indulgent sur la précision, car nous acceptons le fait de prédire trop de points de contact. Nous pouvons enfin nous intéresser à la spécificité pour restreindre tout de même un minimum le nombre de points prédits dans notre classe positive.

Cette réflexion nous amène à l'idée d'utiliser une métrique qui prend en compte à la fois la sensibilité et la spécificité, mais en mettant plus de poids sur la sensibilité. Nous utilisons donc le métrique suivant :

Weighted Youden index

L'**indice de Youden pondéré** [19] est définit comme :

$$J_w = 2(w * \text{Sensibilité} + (1-w) * \text{Spécificité}) - 1 \tag{5.18}$$

avec $w \in [0, 1]$.

Dans notre cas, nous choisissons de poser $w = 0.7$ pour mettre un poids plus important sur la sensibilité, mais tout en prenant tout de même en compte la spécificité. Ce sera donc sur cette métrique que nous allons tuner les différents paramètres de notre modèle.

5.5 Tuning et évaluation avec ré-échantillonage

Pour tuner et évaluer notre modèle sur les métriques que nous venons de présenter, nous devons séparer notre jeu de données en différentes parties.

Nous commençons par séparer nos données en deux ensembles : des **données d'apprentissage** sur lesquelles nous allons tuner et ajuster notre modèle, et des **données de test** qui vont nous permettre d'évaluer notre modèle sur des données qu'il n'a jamais rencontré.

Ensuite, au sein de nos données d'apprentissage, nous créons des échantillons par ré-échantillonage, appelés *folds*, de tailles semblables. Ces ensembles nous permettent d'ajuster le modèle sur plusieurs échantillons pour tuner les hyperparamètres du modèles, ou pour évaluer le modèle plus fiablement.

Voici un schéma résumant les séparations de nos données :

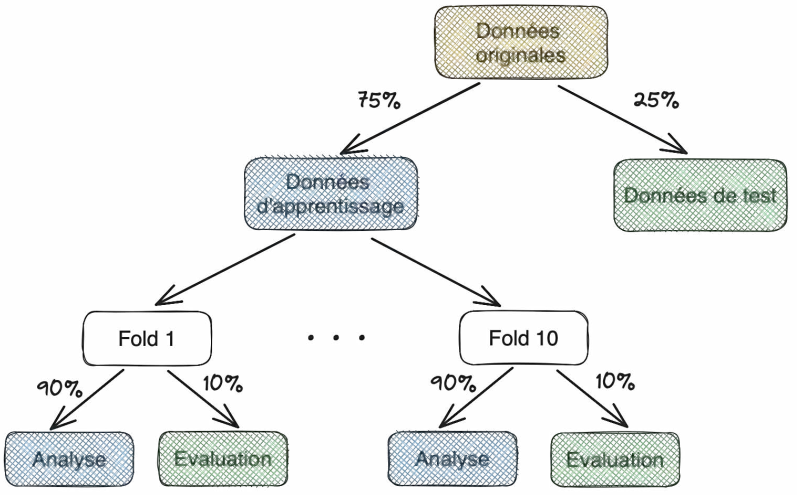


Figure 5.9: Séparation de nos données pour le tuning et l'évaluation du modèle.

Plus précisément, voici les étapes effectuées sur ces ensembles :

1. Tuning des hyperparamètres en ajustant le modèle sur les échantillons d'analyse des *folds*, et en l'évaluant sur les échantillons d'évaluation des *folds*.
2. Ajustement du modèle sur l'ensemble des données d'apprentissage avec les hyperparamètres sélectionnés.
3. Evaluation du modèle sur les données de test, qui n'ont pas encore été utilisées par le modèle.

Dans la pratique, il y a plusieurs manières de séparer nos données originales en différents ensembles. La manière la plus courante est d'attribuer aléatoirement des observations dans chacun des sous-ensembles. Dans notre cas, nous avons choisi de faire en sorte que des sessions entières soient gardées dans nos sous-ensembles. Cette attribution nous semblait plus cohérente et nous permettait surtout par la suite d'afficher nos prédictions sur les sessions de notre ensemble de test. Sous `R`, cela peut être effectué grâce à la fonction `group_initial_split()`, en choisissant de grouper les observations par nom de session.

Pour la construction des *folds*, nous avons également un choix à faire. Cette fois, si nous choisissons de construire nos *folds* par sessions, cela signifie que nous avons un *fold* par session, et donc un grand nombre de *folds*. De plus, cela ne nous semblait pas intuitif car nous voulons que nos *folds* soient représentatif du jeu de données. Nous avons donc décidé de construire les *folds* par attribution aléatoire, mais tout en gardant la certitude que les proportions de nos classes soient identiques dans nos *folds* et dans nos données originales. Sous `R`, cela est pris en compte par l'argument `strata` de la fonction `vfold_cv()`.

6 Applications des méthodes et résultats

6.1 Application des méthodes sous R

Nous avons implémenté les modèles de machine learning sous R, avec le package {tidymodels}, qui permet une implémentation claire grâce à une structure basée sur la {tidyverse}, avec notamment l'utilisation du *pipe operator* (%>%). La construction des modèles suit un schéma avec différentes étapes, que nous détaillerons dans cette partie, et utilise différents packages dont la plupart sont déjà installés directement avec {tidymodels}.

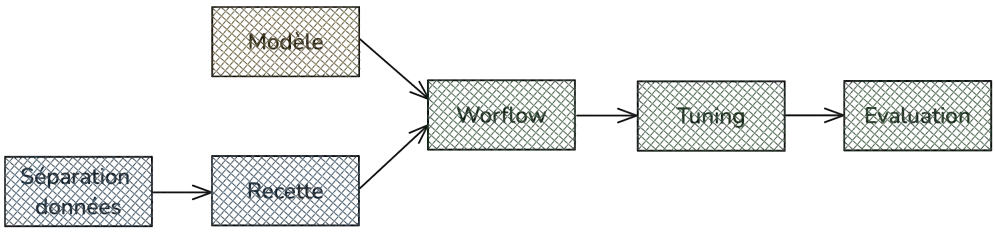


Figure 6.1: Schéma des étapes de la contruction d'un modèle avec {tidymodels}.

6.1.1 Métriques

Afin de tuner nos hyperparamètres et d'évaluer notre modèle, nous avons besoin de choisir des métriques. Dans l'environnement {tidymodels}, de nombreuses métriques sont implémentées dans le package {yardstick}. Nous retrouvons ainsi les métriques sensitivity, specificity et roc_auc correspondant respectivement à la sensibilité, la spécificité et l'aire sous la courbe ROC.

En revanche, l'indice de Youden pondéré, que nous avions sélectionné pour tuner les paramètres du modèle, n'est pas directement implémenté dans le package. Il a donc fallu l'implémenter nous-même, mais cela s'est révélé assez simple grâce au tutoriel présent dans la documentation du package [1]. Le code correspondant est présent en annexes (Voir Listing 9.1).

[1] https://www.tidymodels.org/learn/develop/metrics/#class-example-miss-rate

Enfin, il suffit de créer un ensemble de métriques grâce à la fonction `metric_set()` :

```
all_metrics <- metric_set(sensitivity, specificity, roc_auc,
 ↪  weighted_youden)
```

Nous passerons ensuite cet ensemble aux fonctions utilisées pour le tuning et l'évaluation du modèle.

6.1.2 Feature space et séparation des données

Pour construire le feature space, une fonction a été construite, prenant en entrée les sessions de marches et calculant les différentes variables. Elle récupère également les points de contact donnés par le tapis de marche afin d'ajouter une colonne nommées `class` et contenant les classes `Right_Stance` et `None` en les associant aux temps des séries temporelles.

Une fois que nous avons notre feature space, nous ajoutons une colonne à ce dernier avec les poids que nous voulons mettre sur nos classes, avec la fonction `importance_weights()` :

```
fs <- build_binary_feature_space_window(all_sessions, spar = 0.6, n_lag
 ↪  = 5, k = 3)

fs <- fs %>%
  mutate(
    case_wts = ifelse(class == "Right_Stance", 50, 1),
    case_wts = importance_weights(case_wts)
  )
```

Il est ensuite temps de séparer nos données en différents ensembles. Nous commençons par créeer un ensemble de données d'apprentissage et un ensemble de données de test. Cela est fait avec la fonction `group_initial_split()` du package {`rsample`} car nous avons choisis de garder des sessions entières dans nos ensembles. Ainsi, nous avons dans notre feature space une colonne `session` qui contient le nom des sessions, et qui est passé en argument de cette fonction. Cette colonne ne servant qu'à séparer nos données, nous la supprimons ensuite. Cette fonction place par défaut 75% de nos données dans l'ensemble d'apprentissage et 25% dans l'ensemble de test.

```
splits <- group_initial_split(fs, session)
fs_train <- training(splits)
fs_test  <- testing(splits)
```

Enfin, nous créons également 10 folds sur nos données d'apprentissage, qui nous servirons lors du tuning des paramètres du modèle. La fonction `vfold_cv()` du package {`rsample`} créé par défault 10 folds, et nous ajoutons l'argument `strata` sur notre colonne `class` afin d'avoir les mêmes proportions de chaque classes dans tous les folds.

```
fs_folds <- vfold_cv(fs_train, strata = class)
```

Ces différents ensembles nous servirons lors de la construction et l'évaluation du modèle.

6.1.3 Recette, modèle et workflow

Une fois que nous avons nos ensembles de données, la prochaine étape est la construction d'une recette avec le package {`recipes`}. C'est une suite d'actions à effectuer sur les données avant de les fournir en entrée de notre modèle. Dans notre cas, notre recette n'a qu'une étape de standardisation, qui est réalisée avec l'étape `step_normalize`. La recette permet également de préciser que c'est la variable `class` que nous voulons prédire ensuite.

```
recipe <-
  recipe(class ~ ., data = fs_train) %>%
  step_normalize(all_numeric(), -all_outcomes())
```

Il est maintenant temps de créeer notre modèle de machine learning. De très nombreux modèles sont disponibles dans le package {`parsnip`}, que ce soit pour faire de la classification ou de la régression, et avec différents *engines*. Ces derniers sont des manières différentes d'estimer le modèle : par exemple, pour l'arbre de décision, nous pouvons choisir d'utiliser les arbres CART avec l'*engine* `rpart` ou bien de faire des arbres C5.0 avec l'*engine* du même nom. Nous précisons donc à la fois l'*engine* et le mode lors de la création du modèle :

```
tree_mod <-
  decision_tree(tree_depth = tune(), min_n = tune()) %>%
  set_engine("rpart") %>%
  set_mode("classification")
```

Nous choisissons par exemple ici de créer un modèle d'arbre de décision CART pour de la classification. C'est également à cette étape que nous précisons au modèle les paramètres que nous allons tuner dans la suite. Ainsi, nous ne leur passons pas de valeur mais la fonction `tune()`.

Enfin, une fois que nous avons créé la recette et le modèle, nous les assemblons au sein d'un *workflow*, une structure simplifiant ensuite les étapes de tuning, d'ajustement et d'évaluation.

```
tree_workflow <-
  workflow() %>%
  add_model(tree_mod) %>%
  add_recipe(recipe) %>%
  add_case_weights(case_wts)
```

Nous voyons ici qu'en plus d'ajouter la recette et le modèle, nous donnons le rôle de poids à la colonne ajoutée précédemment au feature space, afin que le modèle comprenne que cette colonne n'est pas une variable mais bien un poids à appliquer sur les classes.

6.1.4 Tuning et ajustement

Pour tuner les paramètres du modèle sur nos *folds*, dans cet exemple ceux de l'arbre de décision, nous utilisons la fonction `tune_grid()` du package {`tune`}. Nous précisons que nous voulons tuner sur les *folds* créés à partir des données d'apprentissage, et nous précisons les métriques qui doivent être calculées durant cette étape.

```
tree_res <-
  tree_workflow %>%
  tune_grid(resamples = fs_folds,
            grid = 20,
            metrics = all_metrics)
```

Nous pouvons soit passer en argument de cette fonction une grille de paramètres à tester que nous choisissons nous même, soit laisser la fonction construire cette grille elle-même. Dans ce dernier cas, nous passons à la fonction la taille de la grille, dans cette exemple 20. Cette grille est construite semi-aléatoirement grâce à la fonction `grid_latin_hypercube()` du package {`dials`}, en utilisant la méthode d'échantillonnage par hypercube latin (McKay, Beckman et Conover [21]). Cela signifie que contrairement à un échantillonnage complètement aléatoire, chaque échantillon est positionné de manière à ne pas avoir de coordonnées communes avec les autres échantillons précédemment positionnés.

Il est ensuite possible de récupérer différents résultats du tuning. Nous pouvons récupérer les cinq combinaisons de paramètres ayant le mieux performés selon une métrique avec la fonction `show_best()`. D'autre part, nous pouvons aussi avoir la combinaison de paramètres ayant maximisés une métrique de notre choix, avec `select_best()`. Les scores associés aux paramètres sont également retournés dans les deux cas.

```
best_param_tree <-
  tree_res %>%
  select_best(metric = "weighted_youden")
```

Ainsi, nous conservons par exemple les paramètres ayant maximisés l'indice de Youden pondéré. Cela nous permet ensuite de finaliser notre *workflow*, c'est à dire de fixer les paramètres de l'arbre de décision avec ces paramètres.

```
last_tree_workflow <-
  tree_workflow %>%
  finalize_workflow(best_param_tree)
```

Une fois que les paramètres ont été fixé, il faut ajuster le modèle sur les données d'apprentissage et l'évaluer sur les données de test. La fonction `last_fit()` est notre alliée dans ce cas car elle effectue ces deux actions, et elle sait qu'il ne faut pas appliquer les poids des classes sur les données de test mais uniquement sur les données d'apprentissage. Nous lui donnons en argument un objet {`rsample`}, dans notre cas la séparation entre données d'apprentissage et de test, ainsi que les métriques sur lesquelles évaluer le modèle.

```
last_tree_fit <-
  last_tree_workflow %>%
  last_fit(splits, metrics = all_metrics)
```

6.1.5 Evaluation et prédictions

Pour évaluer le modèle, nous pouvons tout d'abord récupérer les scores de l'évaluation sur les données de test après ajustement sur les données d'apprentissage.

```
last_tree_fit %>%
  collect_metrics()
```

Cela nous affichera les scores pour les métriques que nous avions donné en argument de la fonction `last_fit()`.

D'autre part, nous pouvons visualiser le score d'importance des variables. Cette fois, nous ne pouvons pas utiliser directement l'objet créé par `last_fit()`, et nous devons donc de nouveau ajuster notre modèle sur nos données d'apprentissage.

```
final_tree <- tree_workflow %>% fit(fs_train)

final_tree %>%
  extract_fit_parsnip() %>%
  vip()
```

Cette fonction nous affiche un graphique ayant par défaut les 10 variables les plus importantes, avec leurs importances respectives. Nous pouvons diminuer ou augmenter le nombre de variables affichées.

De plus, dans le cas d'un arbre de décision, nous pouvons bien sur visualiser l'arbre construit, de nouveau sur un objet provenant d'un `fit()`.

```
final_tree %>%
  extract_fit_engine() %>%
  rpart.plot(roundint = FALSE, yesno = 2, box.palette = "Blues")
```

De nombreux paramètres sont disponibles dans la fonction `rpart.plot()` afin de modifier la représentation de l'arbre [2]. Par exemple, nous choisissons ici d'ajouter des *"yes"* et *"no"* à chaque séparation pour plus de lisibilité, et choisissons une palette de couleurs dans les tons bleus.

Enfin, si nous voulons effectuer des prédictions, il suffit d'utiliser la fonction `predict()` sur le modèle ajusté. Dans notre cas, nous avons une fonction nous permettant de construire le feature space à prédire à partir d'une session de marche, qui calcule donc uniquement les variables décrivant la marche.

```
features_to_predict <- build_features4predictions(qts, speed, spar =
 ↪  0.6, n_lag = 5)
preds <-
  final_tree %>%
  predict(features_to_predict)
```

Il suffit ensuite d'afficher ces prédictions sur les séries de marche ainsi que de récupérer les temps prédits pour ensuite segmenter le signal.

[2] Vignette de la fonction : http://www.milbo.org/rpart-plot/prp.pdf.

6.2 Résultats

6.2.1 Comparaison des modèles

Pouc choisir un modèle de machine learning, nous avons comparé différents modèles sur nos données. Pour cela, nous avons fixé les paramètres du feature space ainsi que les poids placés sur nos classes avec des valeurs qui nous paraissaient cohérentes. Pour le feature space, nous avons fixé le paramètre de lissage `spar` à 0.5, et le lag `n_lag` à 3. Pour les poids sur nos classes, nous avons choisi de placer les poids venant de la formule *inverse class frequency* que nous avons détaillé dans la partie précédente (Équation 5.13). Le code pour cette fonction est présent en annexe (Voir Listing 9.3).

Ensuite, nous avons tuné les paramètres de chaque modèle sur une grille de taille 10 ou 15 selon le nombre de paramètres à tester. Nous avons sélectionné les paramètres ayant maximisés l'indice de Youden pondéré, ce dernier ayant un poids de 0.7 sur la sensibilité et un poids de 0.3 sur la spécificité.

Voici les résultats des différents modèles testés :

Modèle	Sensibilité	Spécificité	Indice de Youden pondéré	Aire sous la courbe ROC
Régression logistique	0.99	0.74	0.83	0.93
Arbre de décision	0.93	0.84	0.81	0.92
Bagged trees	0.85	0.95	0.75	0.95
Boosted trees	0.76	0.96	0.65	0.97
Forêt aléatoire	0.61	0.98	0.44	0.97

Table 6.1: Scores des modèles après tuning sur données d'apprentissage et évaluation sur données de test.

D'après ces résultats, nous voyons que la régression logistique a le plus grand score pour l'indice de Youden pondéré avec un score de 0.83, juste au dessus de l'arbre de décision qui a un score de 0.81. Toutefois, en observant les autres métriques, nous voyons que pour la régression logistique, la sensibilité est excellente alors que la spécificité est moins bonne. En revanche, pour l'arbre de décision, nous avons un meilleur équilibre entre ces deux métriques, ce qui semble être donc un meilleur choix pour notre modèle.

En effet, si nous observons les prédictions effectuées par le modèle de régression logistique, nous observons souvent trop de points prédits, comme sur cette exemple :

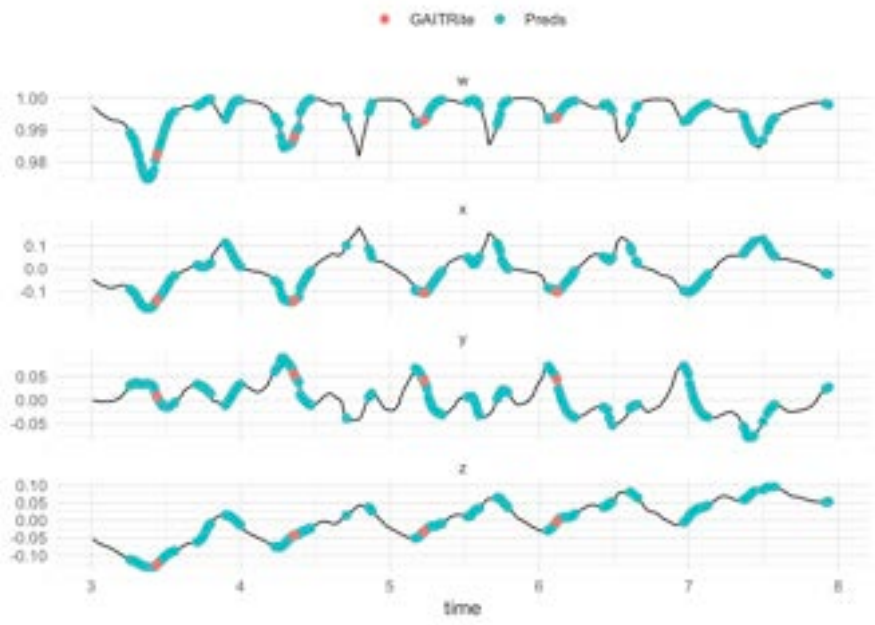


Figure 6.2: Points prédits par la régression logistique et points de référence, sur une session de nos données de test.

Pour les modèles à plusieurs arbres, nous voyons que les performances se dégradent. Ces modèles ont de très bons scores pour la spécificité mais une moins bonne sensibilité, alors que nous souhaitions prendre davantage en compte cette dernière. La forêt aléatoire est par ailleurs le modèle le plus décevant, avec un indice de Youden à seulement 0.44 et une sensiblité de 0.61, indiquant que le modèle oublie trop de points dans la classe d'intérêt.

Ainsi, d'après cette comparaison, nous choisissons d'utiliser le modèle d'**arbre de décision** pour classer les temps des sessions de marche. En effet, nous obtenons de bons scores avant même de tuner les hyperparamètres du feature space et les poids à placer sur les classes. De plus, l'arbre de décision permet à la fois des temps de calcul rapide et une bonne interprétabilité du modèle grâce à la visualisation de l'arbre.

Enfin, nous avons testé un modèle d'arbre de décision sans poids sur nos classes pour nous assurer de la pertinence de cette implémentation. Nous trouvions un score de seulement 0.20 pour l'indice de Youden car le modèle oubliait beaucoup de points dans notre classe minoritaire. Ce résultat confirme qu'il est essentiel de placer des poids sur nos classes dans notre étude.

6.2.2 Résultats du tuning

Grâce à des fonctions implémentées, nous avons tuné plusieurs paramètres dans notre modèle, à différents niveaux. Nous avons tuné le poids à appliquer sur la classe minoritaire, en fixant le poids de la classe majoritaire à 1. De plus, nous avons tuné les paramètres du feature space `spar` and `n_lag`. Enfin, nous avons tuné les paramètres de l'arbre de décision `tree_depth` et `min_n`. Pour les paramètres de poids et du feature space, nous avons testé les grilles de paramètres résumées dans le tableau suivant. Pour les paramètres de l'arbre de décision, nous avons laissé {`tidymodels`} construire la grille de paramètres à tester, en lui demandant de tester 20 couples de paramètres différents.

Poids sur classe minoraire	`weight_RS`	`c(5, 10, 20, 35, 50)`
Feature Space	`spar`	`c(0.3, 0.4, 0.5, 0.6, 0.7)`
	`n_lag`	`c(1, 2, 3, 4, 5)`

Table 6.2: Grilles de paramètres testées lors du tuning du modèle d'arbre de décision.

Après avoir tuné les paramètres pour maximiser l'indice de Youden pondéré, en plaçant un poids de 0.7 sur la sensibilité et un poids de 0.3 sur la spécificité, nous pouvons choisir les paramètres à conserver pour construire notre modèle. Les 10 combinaisons de paramètres ayant le mieux performés par rapport à l'indice de Youden pondéré se situent en annexe (Voir Figure 9.1). Ainsi, les paramètres sélectionnés sont les suivants :

Poids sur classe minoraire	`weight_RS`	50
Feature Space	`spar`	0.6
	`n_lag`	5
Arbre de décision	`tree_depth`	7
	`min_n`	38

Table 6.3: Paramètres sélectionnés après tuning pour construire le modèle.

Avec ces paramètres, nous obtenons pendant le tuning un score de 0.86 pour l'indice de Youden, un score de 0.98 pour la sensibilité et un score de 0.83 pour la spécificité. Ce sont donc des scores satisfaisants nous indiquant que le modèle est performant pour prédire les temps de pose au sol du pied.

6.2.3 Arbre de décision et importance des variables

Un des avantages du modèle d'arbre de décision est son **interprétabilité**. En effet, nous pouvons observer à chaque étape de l'arbre la séparation effectuée selon nos variables.

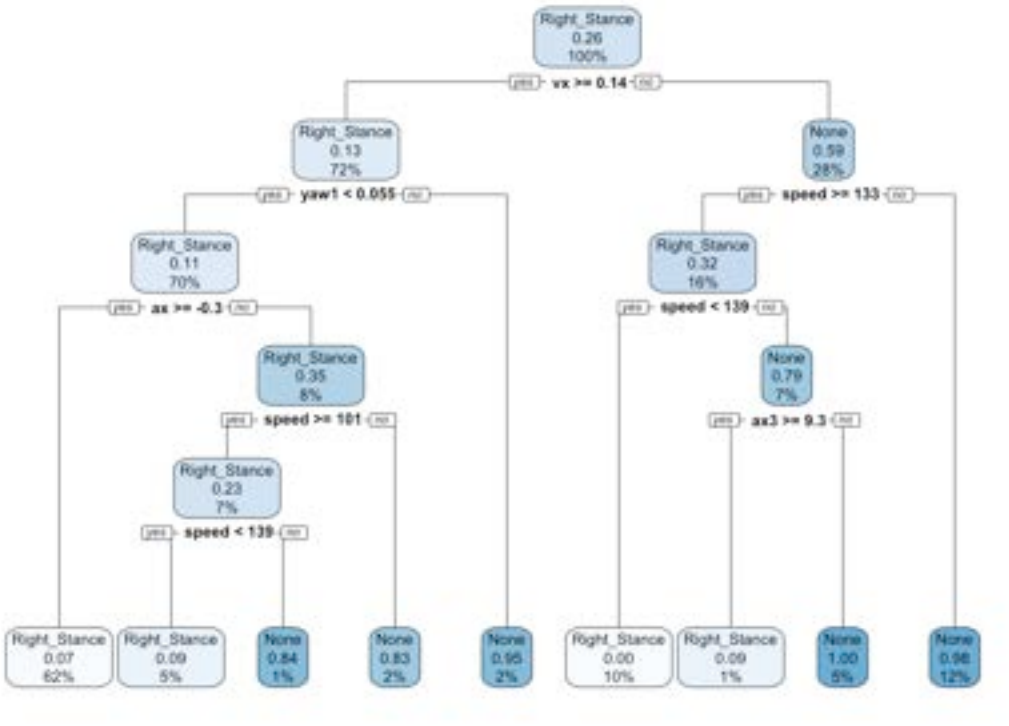


Figure 6.3: Arbre de décision après ajustement sur les données d'apprentissage.

Nous visualisons donc un arbre d'une profondeur de 5, et nous pouvons remarquer qu'à la fois la vitesse angulaire, l'accélération angulaire, l'angle d'Euler Yaw, et la vitesse sont utilisés pour construire notre arbre. Cela signifie donc que les variables ajoutées dans le feature space sont bien pertinentes.

Plus précisément, nous voyons que la majorité des points détectés comme `Right_Stance` sont ceux ayant une vitesse angulaire selon x supérieure à 0.14, un angle Yaw inférieur à 0.01, et une accélération angulaire selon x supérieure à -0.3. Ensuite, les autres séparations dans l'arbre se font majoritairement grâce à la vitesse de marche, avec des séparations autour des vitesses de 100 cm/s et 135 cm/s. Cela correspond aux vitesses séparant respectivement les marches lentes des marches intermédiaires, et les marches intermédiaires des marches normales. Ainsi, comme nous l'avions remarqué en visualisant les sessions de marche, la vitesse de marche a bien un impact sur les temps où nous posons notre pied droit au sol.

Une autre manière d'étudier la construction du modèle est de regarder l'**importance des variables**.

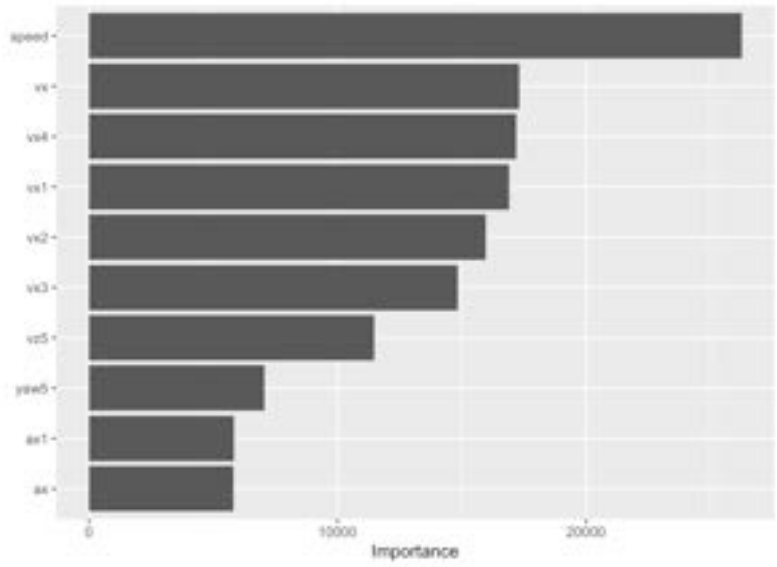


Figure 6.4: Importance des variables dans notre modèle.

Cela nous permet de voir directement que la vitesse de marche est bien la variable la plus importante dans notre modèle, comme nous nous en doutions. Ensuite, ce sont les variables de la vitesse angulaire qui sont les plus importantes, surtout celle selon x, et avec différents lags dans le temps.

6.2.4 Résultats sur données de test

Une fois les paramètres sélectionnés grâce au tuning effectué sur les *folds* des données d'apprentissage, nous construisons un modèle que nous ajustons sur les données d'apprentissage et que nous évaluons sur les données de test. Ainsi, voici les scores obtenus sur les données de test :

Indice de Youden pondéré	Sensibilité	Spécificité	Aire sous la courbe ROC
0.84	0.99	0.77	0.90

Table 6.4: Résultats des métriques sur les données de test.

De nouveau, nous avons de bons résultats, signifiant que le modèle arrive à prédire sur de nouvelles données après avoir été ajusté sur nos données d'apprentissage. Nous obtenons particulièrement une très haute sensibilité, ce qui a pour conséquence la prédiction de plages de points et non pas d'un point unique caractérisant la pose du pied au sol.

En complément de ces scores, nous devons observer sur les sessions de marches de nos données de test les prédictions faites par notre modèle. Nous les affichons sur les séries temporelles de quaternions, accompagnées des points données par le dispositif de référence GAITRite©.

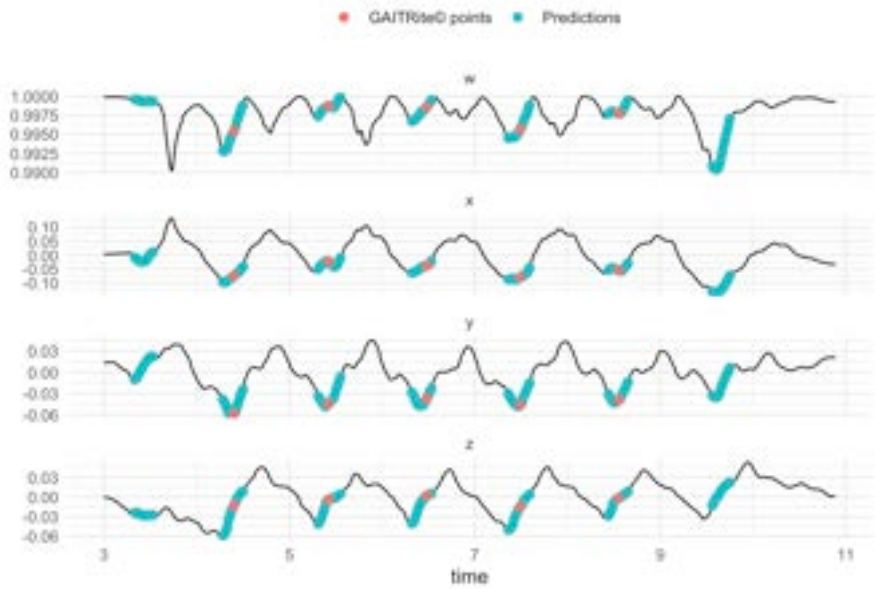


Figure 6.5: Plages prédites et points de référence, sur une marche en vitesse normale de nos données de test.

Nous observons ci-dessus un exemple de résultat obtenu en vitesse de marche normale, des résultats concernant les vitesses lentes et rapides sont présents en annexes (Voir Figure 9.2 et Figure 9.3). De plus, chacune de ces sessions de test avait été réalisée par un sujet différent, cela montrant que le modèle arrive à faire de bonnes prédictions quelque soit la vitesse de marche et le sujet.

Nous observons des résultats similaires sur toutes les sessions de nos données de test, avec une plage de points prédits contenant le point de référence donné par le tapis de marche. Ainsi, notre modèle prédit trop de points de contact avec le sol, mais cela permet qu'il n'en oublie pas et les points prédits sont consécutifs.

Si nous souhaitons obtenir le nombre exact de points de contacts réellement effectués par le sujet, nous pouvons choisir de sélectionner un point par plage prédite. Nous pouvons donc choisir de conserver uniquement le point central des plages obtenues (code en annexes, voir Listing 9.4), afin d'obtenir le même nombre de points que le dispositif de référence. Cela peut nous aider à mieux visualiser les prédictions obtenues et à comparer les deux dispositifs.

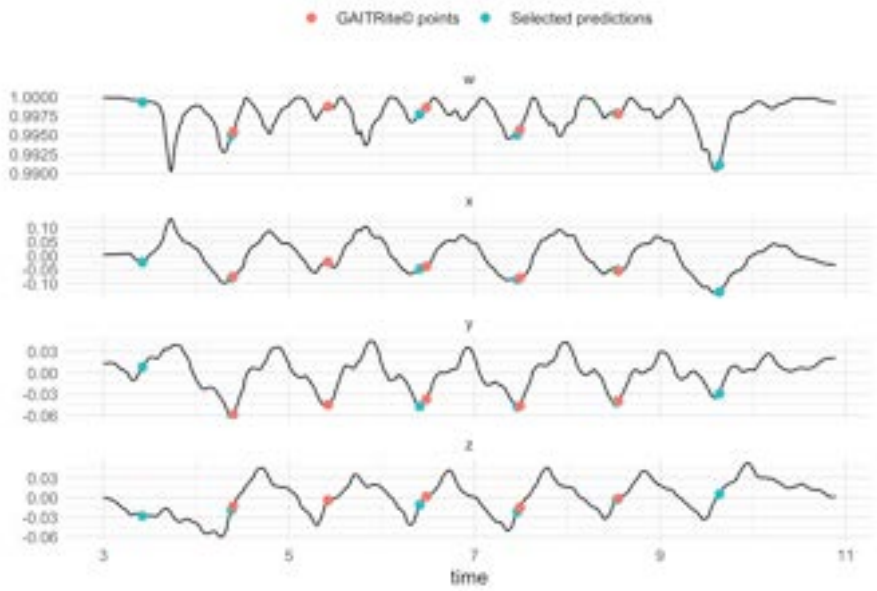


Figure 6.6: Points sélectionnés d'après nos prédictions et points de référence, sur une marche en vitesse normale de nos données de test.

Sur cet exemple, nous voyons que les points sélectionnés, se situant au centre des plages de points prédits, sont très proches des points donnés par GAITRite©. Ce résultat est donc très satisfaisant, notre modèle semble bien prédire les temps où le pied droit se pose au sol, et donc semble segmenter correctement le signal brut renvoyé par le capteur eGait.

6.2.5 Résultats sur données AMIES

En complément de nos données de test, nous avons voulu tester le modèle sur des sessions de marche totalement différentes. Nous avons donc utilisé une base de données appelées AMIES et contenant des données de marche de patients atteints de sclérose en plaque et donc souffrant plus ou moins de troubles de la marche. Ces donnés ont été acquises en 2022 sur 44 patients, chacun effectuant quatre marches de 7.62 mètres, dans le cadre du test T25FW (*Timed 25 Foot Walk*) effectué par les patients souffrant de sclérose en plaques.

Sur ces données, comme les patients ont uniquement marché avec le capteur eGait et non sur le tapis GAITRite©, nous pouvons uniquement afficher les prédictions effectuées par notre modèle. Cela nous permet tout de même d'observer graphiquement la segmentation des cycles de marche sur le signal, pour voir si elle nous parait cohérente. Voici un exemple de prédictions sur ces données, d'autres exemples se situent en annexes (Voir Figure 9.4).

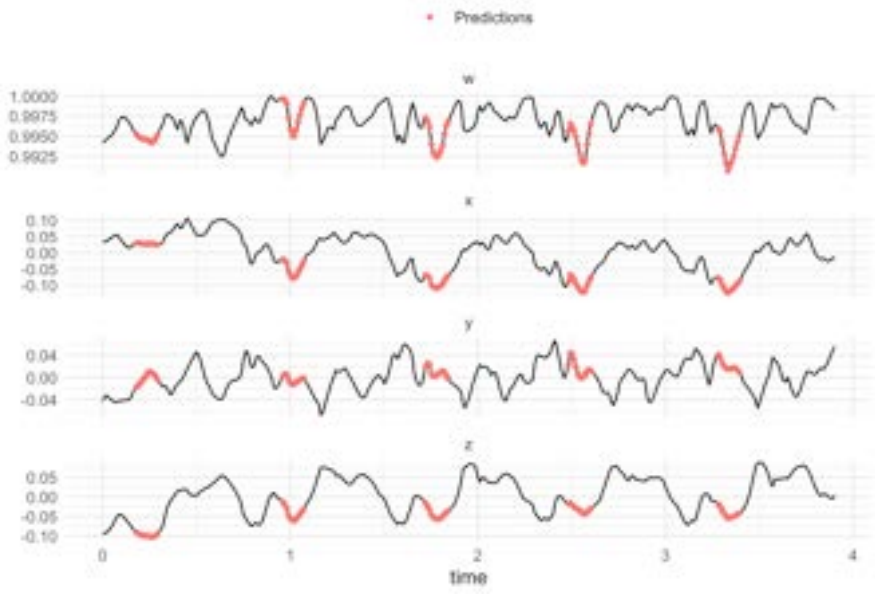


Figure 6.7: Prédictions sur sessions de marche d'un patient AMIES.

Sur cette session, nous voyons que même si la série est moins lisse et régulière que celles acquises sur des sujets sains, le modèle arrive quand même à segmenter le signal, avec des prédictions à des intervalles de temps régulier et nous semblant cohérentes.

Malheureusement, comme sur l'exemple ci-dessous, le modèle n'arrive pas à faire de très bonnes prédictions sur toutes les sessions.

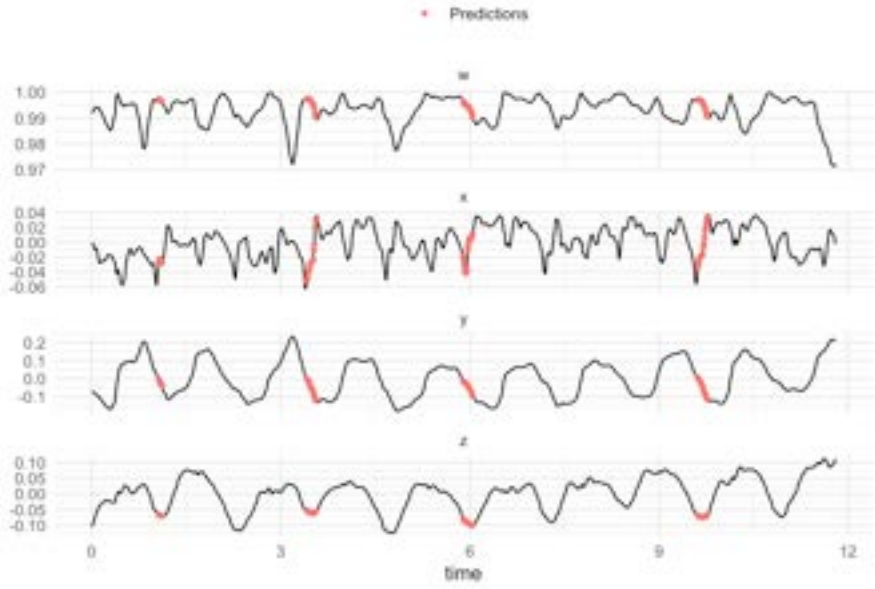


Figure 6.8: Prédictions sur sessions de marche d'un patient AMIES.

Sur cette session, le modèle semble en effet avoir oublié de détecter des points de contact avec le sol, car les courbes sont assez régulières, mais les prédictions ne se situent pas régulièrement au niveau des courbes.

La majorité des résultats obtenus sur les données de patients atteints de sclérose en plaques sont satisfaisants. En revanche, sur certaines sessions, le signal n'est pas bien segmenté, indiquant que notre modèle pourrait être amélioré pour mieux performer sur des marches de sujets souffrant de troubles de la marche. Ces résultats restent encourageant car la majorité des sessions paraissent tout de même correctement segmentées.

7 Comparaison des dispositifs eGait et GAITRite©

Comme nous avons réussi à construire un modèle détectant le temps de contact au sol du pied droit lors d'une session de marche, nous pouvons **segmenter les cycles de marche** des sessions récoltées en utilisant simultanément les dispositifs eGait et GAITRite©. Pour cela, nous sélectionnons le point central des plages de points prédits par le modèle pour obtenir des points comparables à ceux fournis par le tapis de marche.

Pour comparer les deux dispositifs, nous avons du faire attention à un phénomène qui ne nous avait pas posé de problèmes précédemment. En effet, le tapis de marche détecte uniquement les pas effectués sur le tapis au niveau des capteurs, ce qui signifie que le premier pas et/ou le dernier pas effectué ne sera pas forcément enregistré selon la position des pieds. A l'inverse, avec le capteur eGait, tous les pas sont enregistrés, qu'importe où ils ont été effectués. Cela est visible sur la Figure 6.5 où des points supplémentaires sont prédits au début et à la fin de la session. Nous avons donc retiré les points prédits qui se situaient :

- 0.5 secondes avant le premier point détecté par GAITRite©.
- 0.5 secondes après le dernier point détecté par GAITRite©.

Cela nous permet de comparer les dispositifs sur une même plage de temps.

7.1 Méthode

7.1.1 Données utilisées pour la comparaison

Pour comparer les deux dispositifs, nous avons tout d'abord utilisé les mêmes données que celles utilisées pour entrainer notre modèle de machine learning. Ces données contiennent 77 sessions de marche, en vitesse lente, intermédiaire, normale et rapide. Elles sont très régulières et synchronisées, et ont été récoltées en juin et juillet dernier.

D'autre part, nous pouvons également effectuer la comparaison avec des données récoltées en novembre 2023 et février 2024. Ces données n'ont pas été utilisées précédemment car elles n'ont pas été acquises aussi précisément : sans ceinture officielle, sans réelle synchronisation des dispositifs, etc. Nous avons dans ces données 135 sessions de marche, effectuées par 12 sujets (5 femmes et 7 hommes), sur des vitesses de marche lentes, normales et rapides.

7.1.2 Paramètres de comparaison

Nous avons utilisé différents **paramètres spatio-temporels** pour comparer les résultats renvoyés par le capteur eGait et par le tapis GAITRite©. Premièrement, il y a des paramètres que nous pouvons récolter avec les deux dispositifs au sein d'une session de marche :

- Le nombre de cycles effectués.
- La durée moyenne des cycles.
- La variation de la durée des cycles.

Nous pouvons aussi étudier des paramètres qui ne sont pas identiques mais qui peuvent être mis en relation :

- La vitesse angulaire moyenne calculée sur les données eGait, et la vitesse de marche calculée par GAITRite©.
- L'amplitude rotative moyenne calculée sur les données eGait, et la longueur moyenne d'une enjambée calculée par GAITRite©.

Pour ces derniers paramètres, nous allons simplement étudier la présence d'une relation linéaire et utiliser la corrélation de Pearson. Pour les premiers paramètres, nous allons pouvoir analyser leurs similarités avec des tests appariés, ainsi qu'étudier leur concordance avec le diagramme de Bland-Altman, présenté dans la suite.

7.1.3 Test de Wilcoxon apparié

Comme nous sommes dans le cas de mesures effectuées sur les mêmes sujets, pour comparer deux dispositifs, nous pouvons effectuer des tests dits appariés. Le test de Student apparié est très utilisé pour comparer des données quantitatives, en évaluant l'égalité de leurs moyennes. Cependant, il réside sur l'hypothèse de normalité des données, alors que nos données ne respectent pas cette condition. Dans ce cas, nous pouvons utiliser le **test de Wilcoxon apparié** (Wikipedia [27]) qui est une alternative non-paramétrique au test de Student. Il se base sur l'hypothèse que la différence entre les échantillons appariés suit une distribution symétrique autours d'un centre noté θ. Nous voulons tester si ce centre est égal à 0, l'hypothèse nulle est donc $H_0 : \theta = 0$.

Ce test se base sur les rangs des observations. On range dans l'ordre croissant les valeurs absolues des différences $|Z_i|, ... |Z_n|$, et on note R_i le rang de $|Z_i|$. On définit la fonction indicatrice Ψ_i tel que $\Psi_i = 1$ si $Z_i > 0$, et 0 sinon. La statistique de test est :

$$T^+ = \sum_{i=1}^{n} R_i \Psi_i \tag{7.1}$$

Ainsi, on rejette H_0 si $T^+ \notin \left[\frac{n(n+1)}{2} - t_{\alpha/2}, t_{\alpha/2} \right]$ avec t_α choisi tel que le risque de première espèce est égal à α.

7.1.4 Diagramme de Bland-Altman

Le **diagramme de Bland-Altman** (Bland et Altman [4]) permet d'évaluer la concordance entre de mêmes mesures prises avec deux appareils différents. Il est souvent utilisé pour comparer une nouvelle technique de mesure avec une technique déjà reconnue, pour voir si la nouvelle méthode peut remplacer l'ancienne. Le diagramme permet également d'identifier un biais fixe, c'est à dire une différence systématique entre les deux mesures, qui pourrait être corrigé en le soustrayant à la nouvelle méthode, ou encore de repérer des outliers.

Le diagramme représente les différences entre les méthodes par rapport à leurs moyennes.

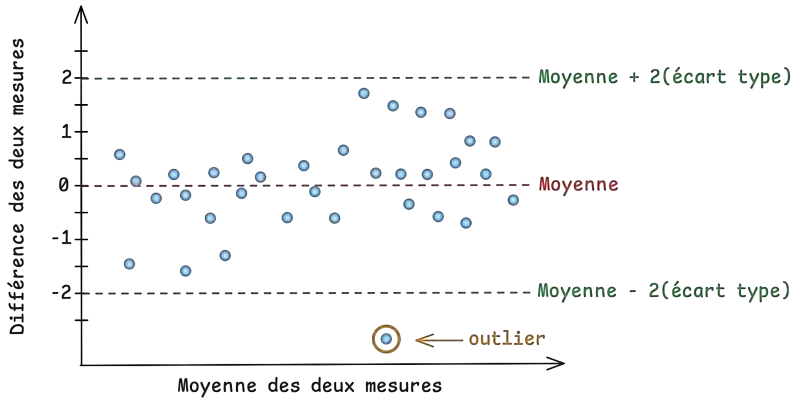


Figure 7.1: Schématisation du diagramme de Bland-Altman.

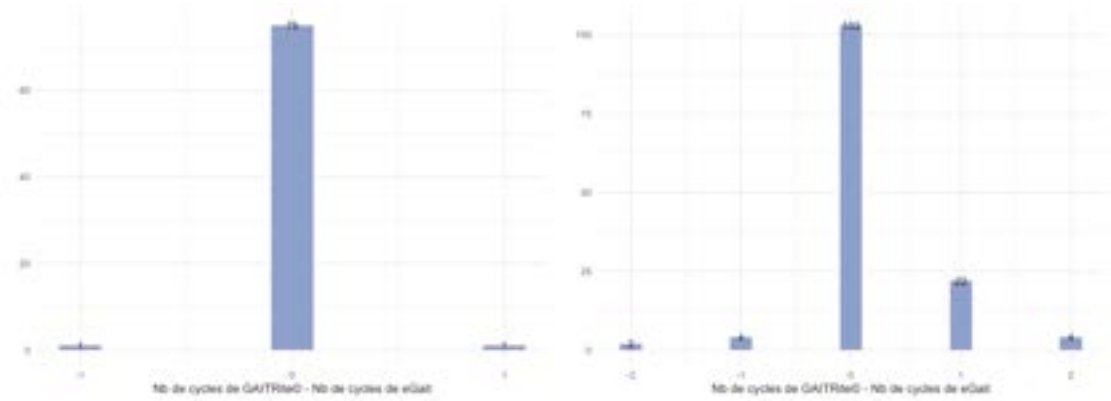
La différence moyenne, représentée en rouge sur la figure, représente le biais estimé. L'écart type des différences mesure les fluctuations autour de ce biais. On calcule également ce qu'on appelle les limites de la concordance à 95%, ce qui est la différence moyenne plus ou moins 2 fois l'écart type, ce qui est représenté par les traits verts sur la figure. Cela nous permet de voir à quel point les mesures sont éloignées pour la plupart des observations.

Ainsi, si nous décidons que les différences comprises dans cet intervalle sont cliniquement acceptables, alors nous concluons sur le fait de pouvoir utiliser les deux méthodes de mesure de manière interchangeable.

7.2 Résultats

7.2.1 Nombre de cycles

Nous commençons par comparer le **nombre de cycles de marche** détecté par l'appareil de référence GAITRite© et celui que nous obtenons par l'algorithme de segmentation mis en place sur les données récoltées par le dispositif eGait. Pour chaque session, nous calculons la différence entre le nombre de cycles obtenu par GAITRite© et celui obtenu par eGait, et nous affichons les résultats avec un diagramme à barres.



(a) Sessions de juin et juillet.

(b) Sessions de novembre et février.

Figure 7.2: Différence du nombre de cycles détecté dans une session de marche entre les deux dispositifs.

Nous observons que pour les sessions récoltées récemment avec un protocole plus stricte, nous obtenons pour 75 séries sur 77 exactement le même nombre de cycles avec les deux dispositifs. Pour les sessions acquises en novembre et février, les performances sont un peu moins bonnes mais restent satisfaisantes, avec 103 sessions où l'on détecte le même nombre de cycles, et 22 sessions où le tapis détecte un cycle de marche supplémentaire.

Nous avons réalisé ces graphiques en fonction de la vitesse de marche (les graphiques sont présents en annexes, voir Figure 9.5 et Figure 9.6), nous montrant que nous détectons aussi bien les cycles de marche qu'importe la vitesse. Nous avons également réalisé les graphiques en fonction du sexe pour les sessions de novembre et février, comme nous avions davantage de sujets et de diversité par rapport aux sessions récentes comportant uniquement deux femmes et un homme.

Ce graphique nous permet de voir que sur ces sessions, nous arrivons moins bien à détecter le nombre de cycles au sein d'une session de marche lorsque le sujet est une femme.

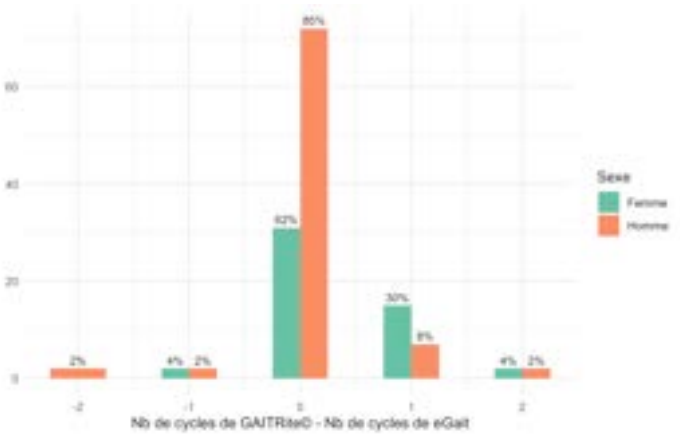


Figure 7.3: Différence entre les nombres de cycles sur les sessions de novembre et février par rapport au sexe.

7.2.2 Durée moyenne des cycles

Dans un deuxième temps, nous comparons la **durée moyenne des cycles** détectés par les deux dispositifs. Nous pouvons effectuer un test de Wilcoxon apparié pour comparer nos deux échantillons. Pour les données acquises récemment, nous obtenons une p-value de $0.17 > 0.05$, signifiant que nous ne pouvons pas rejeter l'hypothèse nulle déclarant que le centre de la distribution des différences vaut 0. Pour les données plus anciennes, nous devons rejeter cette hypothèse.
Nous réalisons également le graphique de Bland-Altman pour analyser la concordance entre les mesures (voir Figure 9.7 en annexes pour les sessions de novembre et février).

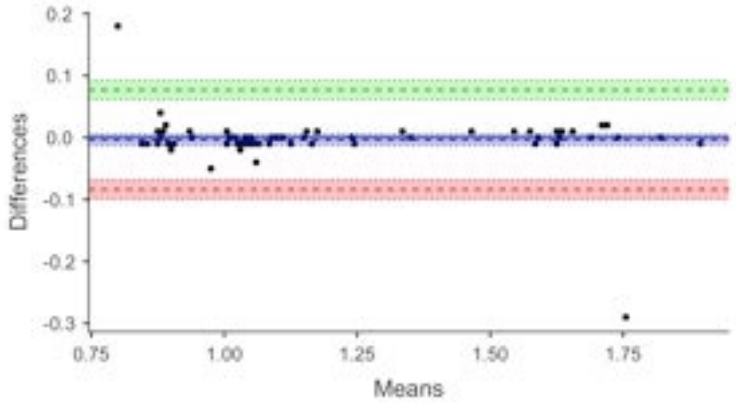


Figure 7.4: Diagramme de Bland-Altman pour la durée moyenne des cycles, sur les sessions de juin et juillet. [1]

[1] Graphique réalisé sous `R` avec le package `blandr`.

Sur ce graphique, nous observons deux outliers, mais toutes les autres différences entre les mesures se situent au sein de l'intervalle de confiance $[-0.08, 0.08]$. Un cycle de marche durant en moyenne 1 seconde, une différence de moins de 0.1 seconde entre la durée des cycles est donc très basse. Nous observons également un biais nul, cela nous permettant de conclure que nous obtenons une durée moyenne de cycles similaire avec le nouvel algorithme de segmentation des données acquises par eGait par rapport au dispositif de référence.

D'autre part, nous étudions également la **variation des durées des cycles** de marche. Pour cela, nous avons calculé le coefficient de variation, définit par $CV = \frac{\text{moyenne}}{\text{écart type}}$. Pour le test de Wilcoxon apparié, nous obtenons cette fois des p-values inférieures à 0.05 pour toutes les sessions, nous forçant à rejeter l'hypothèse nulle. Nous affichons tout de même les graphiques de Bland-Altman (voir Figure 9.8) et obtenons des résultats similaires à ceux obtenus précédemment pour la durée moyenne des cycles.

Pour conclure sur la moyenne et la variation des durées de cycles, nous obtenons de très bons résultats sur les sessions de juin et juillet, nous permettant d'affirmer que notre méthode de détection de cycles peut être utilisée au même titre que le dispositif de référence. Les résultats sont légèrement moins bons sur les sessions de marche plus anciennes mais restent assez satisfaisants compte tenu des circonstances d'acquisition. Nous pouvons noter que nous observons des outliers sur les diagrammes de Bland-Atlman qu'il serait intéressant d'analyser.

7.2.3 Vitesse angulaire moyenne et vitesse de marche

Nous comparons la vitesse angulaire moyenne calculée sur les séries de quaternions renvoyées par eGait [2] et la vitesse de marche calculée par le tapis GAITRite©.

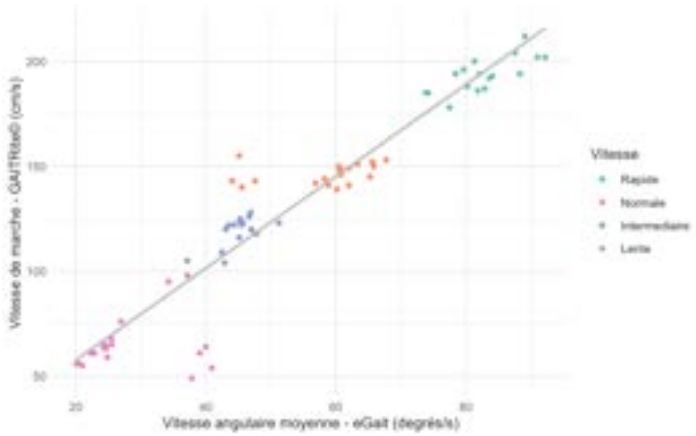


Figure 7.5: Vitesse angulaire moyenne (eGait) et vitesse de marche (GAITRite©) pour les sessions de juin et juillet, par rapport à la vitesse.

[2] La vitesse angulaire d'une série de quaternions est calculée avec la fonction `qts2avts()` du package `squat`.

Le graphique pour les sessions de novembre et février est présent en annexe (voir Figure 9.9).

Ainsi, sur les sessions de juin et juillet, nous observons une corrélation linéaire entre les deux variables, avec une corrélation de Pearson valant 0.96. Il y a quelques observations de marche en vitesse lente et en vitesse normale plus éloignées de la droite de régression, il serait donc intéressant d'analyser ces sessions de marche pour voir si elles appartiennent par exemple à un même sujet.

Il y a donc une corrélation assez forte entre ces deux variables, que ce soit sur les sessions de marche récentes ou plus anciennes.

7.2.4 Amplitude moyenne et longueur moyenne de cycles

Enfin, nous comparons de la même manière l'amplitude moyenne calculée sur les séries de quaternions renvoyées par eGait [3] et la longueur moyenne des cycles calculée par le tapis GAITRite©.

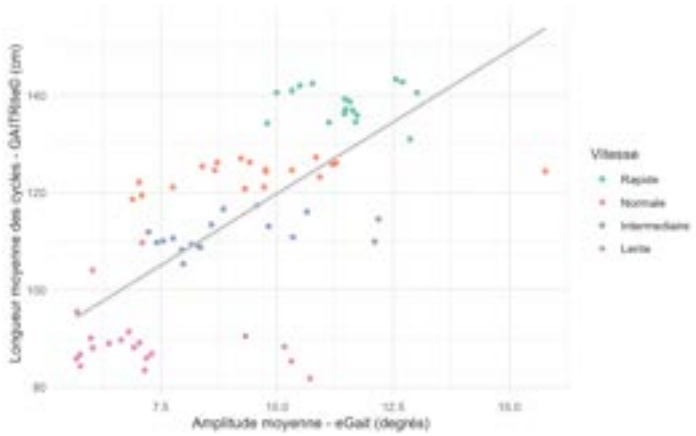


Figure 7.6: Amplitude moyenne (eGait) et longueur moyenne des cycles (GAITRite©) pour les sessions de juin et juillet, par rapport à la vitesse.

Pour les sessions récentes, nous visualisons une corrélation linéaire mais avec des points se situant assez loin de la droite de régression, surtout en marche lente. Nous obtenons cette fois une corrélation de Pearson de 0.69. Pour les sessions plus anciennes (voir Figure 9.10), nous n'observons pas vraiment de corrélation linéaire entre les deux variables, avec un nuage de points assez dispersés. Il est donc plus compliqué de conclure sur la similarité de ces mesures entre les deux dispositifs.

[3] L'amplitude d'une série de quaternions est calculée avec la fonction `qts2ats()` du package `squat`.

8 Conclusion et perspectives

Au sein de ce stage, nous avons acquis de nouvelles données de marche synchronisées entre les dispositifs eGait et GAITRite©. Cela nous a permis de construire un modèle de machine learning par **arbre de décision** pour prédire les temps de pose au sol du pied droit lors d'une session de marche. Ces prédictions servent ensuite de **points de segmentation** pour segmenter le signal renvoyé par eGait en cycles de marches. Nous avons donc ensuite pu calculer des **paramètres spatio-temporels** nous permettant d'analyser la **concordance** entre les deux dispositifs. Cette analyse nous a permis de constater que le dispositif eGait pourrait être utilisé cliniquement car nous obtenons des résultats similaires à l'appareil de référence. En effet, le nouveau modèle détecte au sein des sessions un nombre de cycles, une durée moyenne de cycle, et une variation de durée de cycle similaires à GAITRite©.

La difficulté principale rencontrée lors de l'implémentation du modèle de machine learning a été celle causée par le **déséquilibre de classes** au sein de nos données. En effet, nous nous sommes concentré pendant une période assez longue sur les algorithmes d'échantillonnage sans obtenir de résultats satisfaisants. Lorsque nous avons ensuite découvert la possibilité de placer des poids sur les classes, cette méthode a rapidement montré de bons résultats et cela nous a débloqué dans cette problématique.

Les résultats obtenus par ce modèle sont très bons sur les données acquises récemment, mais nous obtenons des prédictions moins bonnes sur les sessions récoltées dans le passé. Différentes causes peuvent expliquer ce phénomène : un problème de référentiel, une mauvaise calibration du capteur, un placement non précis du capteur sur les sujets, etc.

Ainsi, ce modèle est améliorable et sera perfectionné dans le futur. Tout d'abord, à court terme, nous allons acquérir davantage de données sur de nouveaux sujets, car pour l'instant le modèle n'apprend que sur trois sujets. Ajouter de la diversité dans les données peut permettre au modèle d'améliorer ses prédictions sur les séries acquises dans d'autres contextes. D'autre part, pour améliorer le modèle, nous pouvons envisager de combiner les méthodes d'échantillonnage à la méthode des poids pour gérer au mieux la problématique de déséquilibre de classes.

Plus globalement dans le projet eGait, de nouveaux capteurs sont en cours de fabrication, qui auront le grand avantage de calculer directement la vitesse de marche. Nous allons donc effectuer une phase de test pour ces capteurs, et cela nous permettra de pouvoir intégrer la vitesse de marche dans notre modèle sans devoir connaitre la distance parcourue par les sujets lors de la marche.

A plus long terme, il sera nécessaire d'acquérir des données de marche de personnes âgées afin de tester notre modèle sur cette population, afin de voir si des ajustements devraient être fait pour aider au mieux cette population à hauts risques de trouble de la marche.

Enfin, dans le cadre de ce projet, il est prévu le développement d'une **application web** associée au dispositif eGait. Elle sera mise à disposition du personnel hospitalier afin de leur fournir des résumés graphiques et tabulaires, comme outil intuitif d'aide à la décision et au suivi clinique. Cette application sera le fruit d'une réflexion commune entre le LMJL, le LMIP et le CHU de Nantes. Une première version de cette application avait été implémentée l'an passé lors de mon stage de Master 1, et il est désormais temps d'en faire une version déployable. Pour cela, j'ai pu implémenter une structure de modules `shiny` basée sur les packages `rhino` et `bslib` au début de ce stage (des captures d'écran de l'application sont présentes en annexes, Figure 9.11 et Figure 9.12), que je pourrai reprendre en continuant à travailler sur ce projet à l'issu de ce stage.

9 Annexes

Distance (cm)
Durée (s)
Vitesse (cm/s)
Vitesse Normalisée
Nombre de pas
Cadence (pas/mn)
Différence durée du pas (s)
Différence longueur de pas (cm)
Différence durée du cycle (s)

Table 9.1: Paramètres globaux

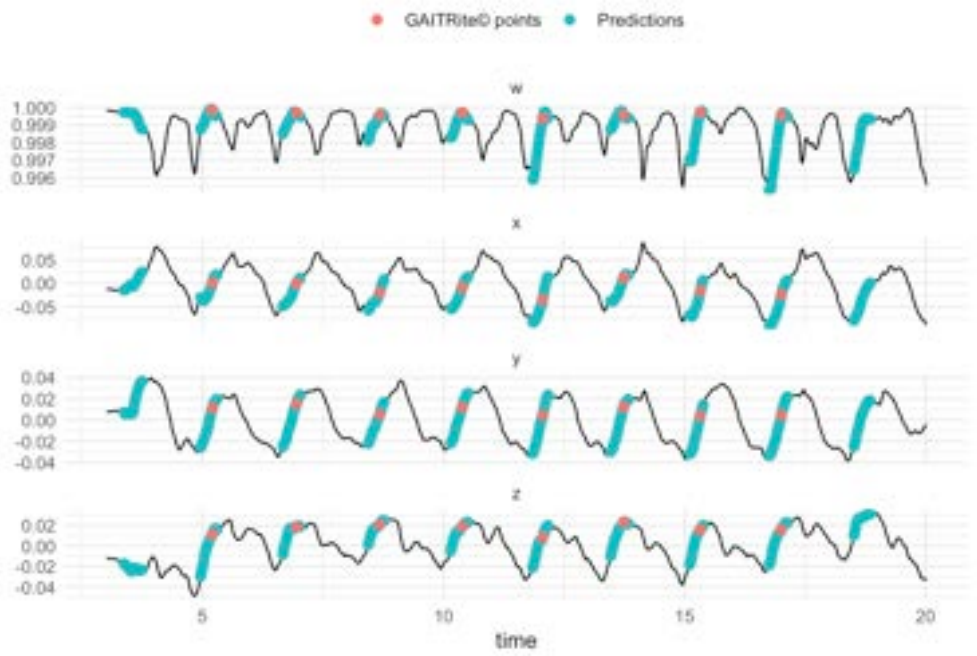
Durée du pas (s)
Durée du cycle (s)
Longueur du pas (cm)
Longueur d'enjambée (cm)
Base appui (cm)
Double appui (%)
Phase oscillante (%)
Phase de support (%)
Pas Normalisé
Rotation du pied (deg)

Table 9.2: Paramètres bilateraux

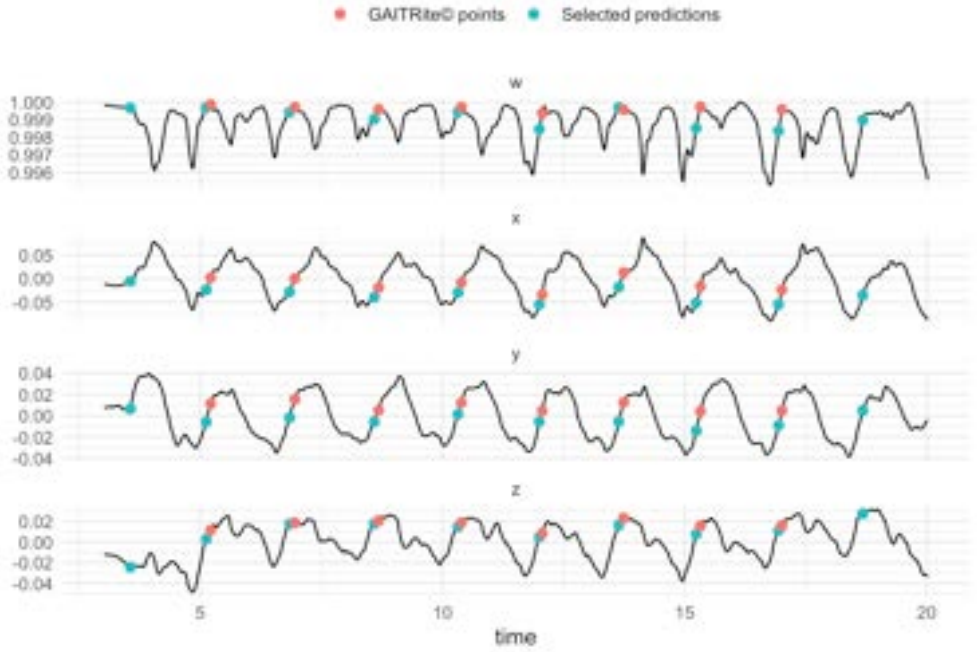
Table 9.3: Paramètres spatio-temporels récoltés par le tapis GAITRite© (Biometrics [3])

spar <dbl>	n_lag <dbl>	weight_RS <dbl>	weight_None <dbl>	tree_depth <dbl>	min_n <dbl>	weighted_youden <dbl>	sensitivity <dbl>	specificity <dbl>	roc_auc <dbl>
0.6	5	50	1	7	38	0.8649070	0.9770408	0.8284165	0.9120363
0.6	4	50	1	12	15	0.8644386	0.9779221	0.8255795	0.9052271
0.7	4	35	1	10	18	0.8616074	0.9813665	0.8128239	0.8985726
0.4	5	35	1	6	11	0.8600708	0.9744898	0.8263086	0.9367788
0.6	3	50	1	10	25	0.8565633	0.9855832	0.7945780	0.9030842
0.7	3	35	1	15	23	0.8537620	0.9973545	0.7624428	0.8833912
0.7	2	50	1	10	5	0.8516900	0.9837093	0.7908284	0.8884889
0.4	2	35	1	8	22	0.8507832	0.9824561	0.7922411	0.9138454
0.4	3	35	1	10	38	0.8475539	0.9639640	0.8300072	0.9236218
0.6	2	35	1	14	23	0.8448648	0.9706275	0.8099771	0.8946713

Figure 9.1: Résultats du tuning de nos hyperparamètres, rangés par score décroissant de l'indice de Youden pondéré.

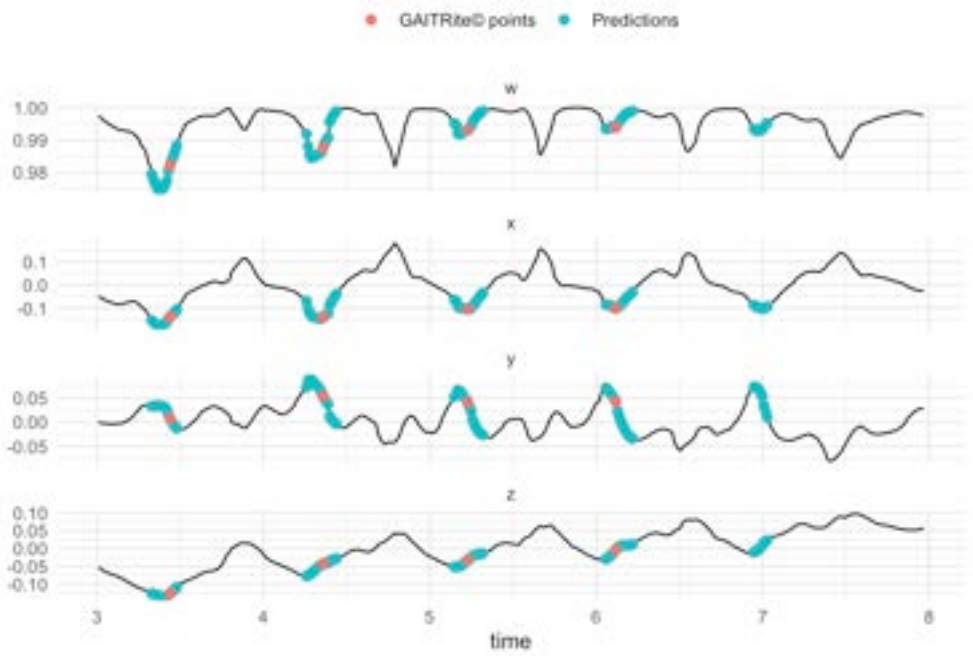


(a) Plages prédites et points de référence.

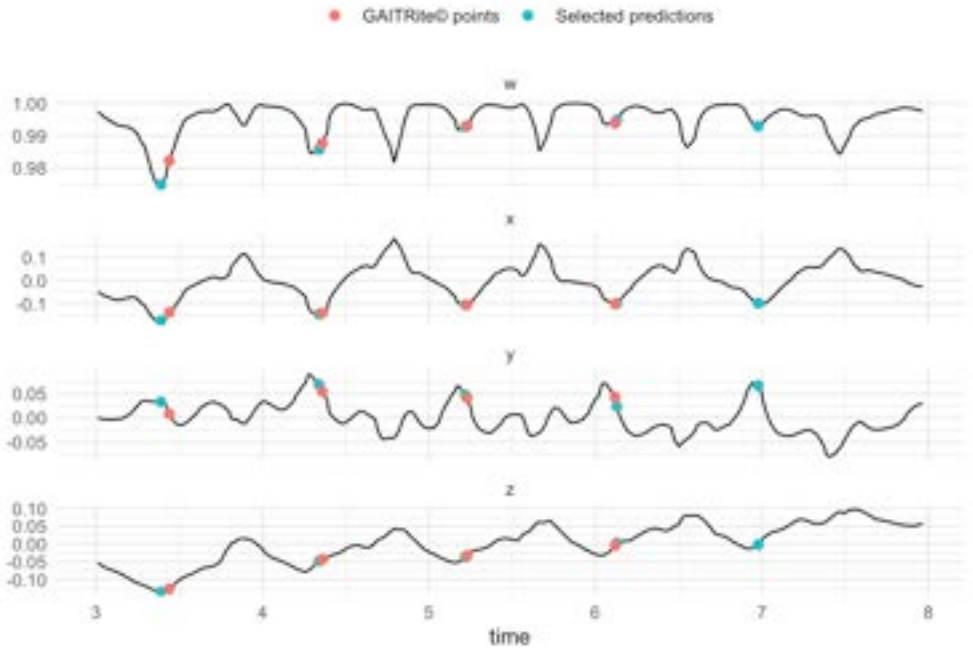


(b) Points centraux sélectionnés et points de référence.

Figure 9.2: Résultats des prédictions sur une marche en vitesse lente de nos données de test.

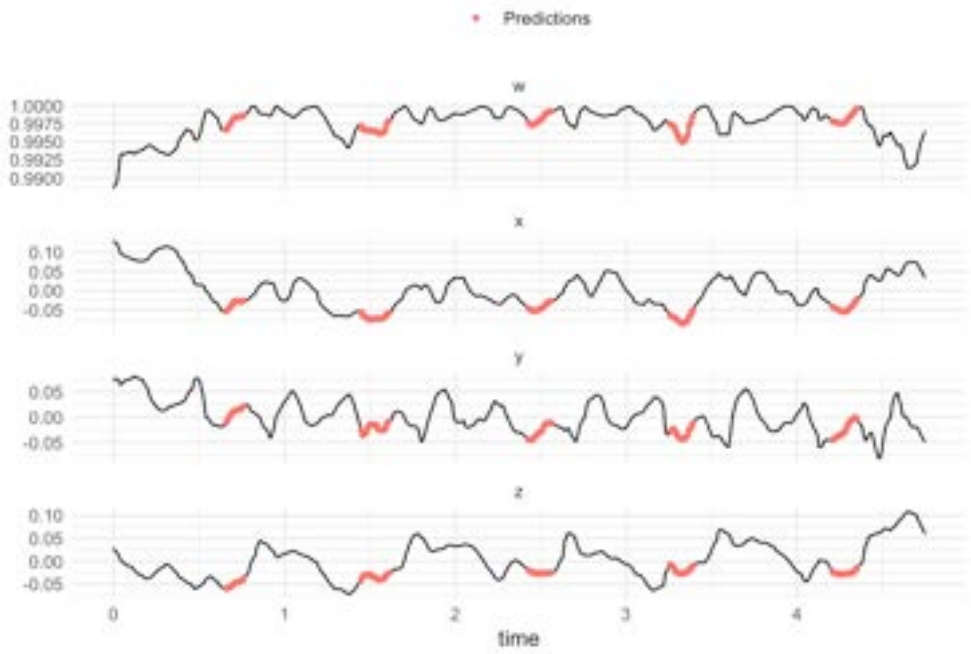


(a) Plages prédites et points de référence.

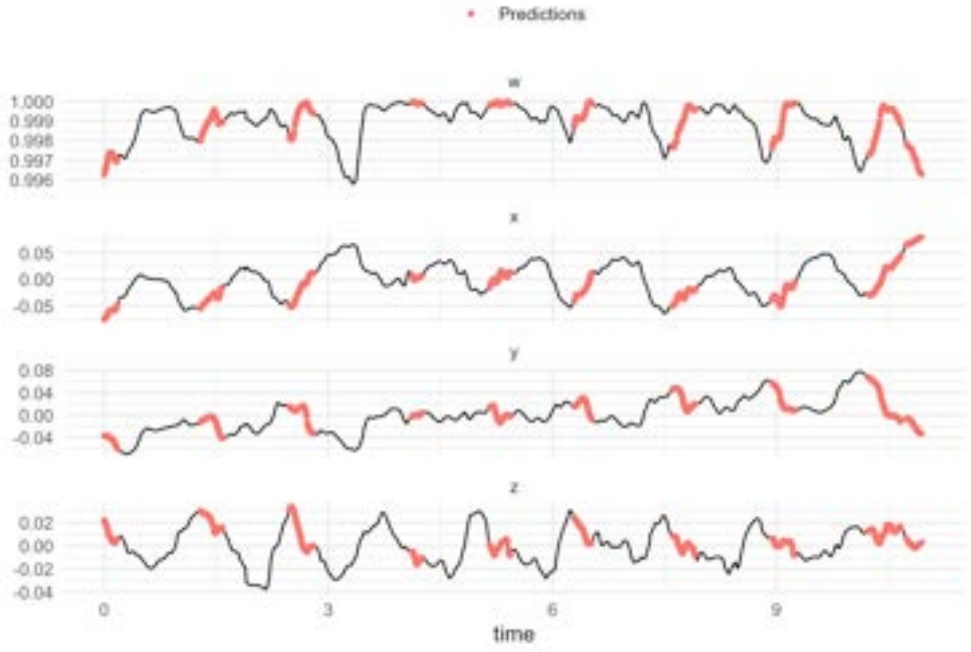


(b) Points centraux sélectionnés et points de référence.

Figure 9.3: Résultats des prédictions sur une marche en vitesse rapide de nos données de test.



(a)



(b)

Figure 9.4: Résultats des prédictions sur des patients atteints de sclérose en plaques.

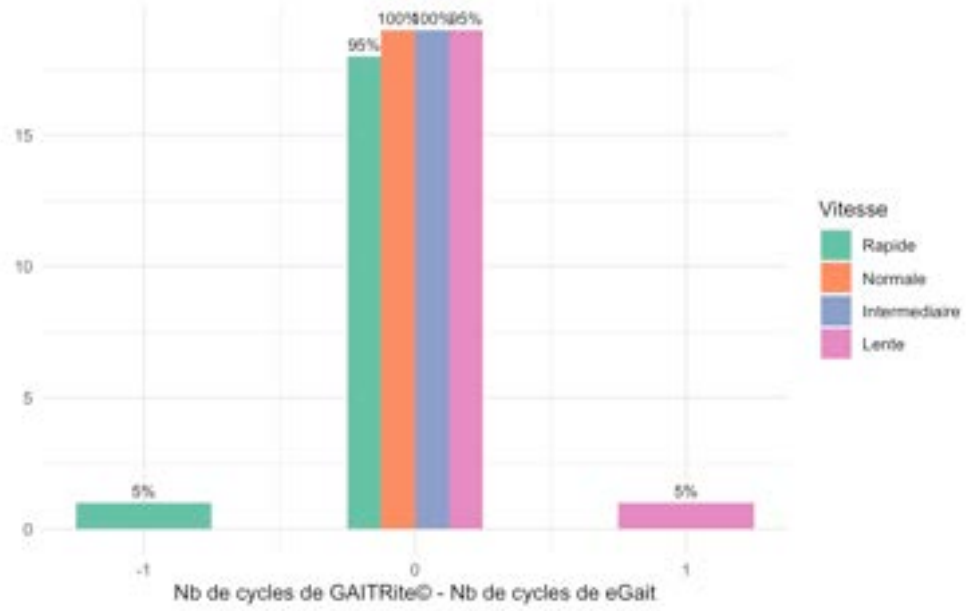


Figure 9.5: Différence du nombre de cycles de marche détectés entre les deux dispositifs, sur les sessions de juin et juillet, par rapport à la vitesse de marche.

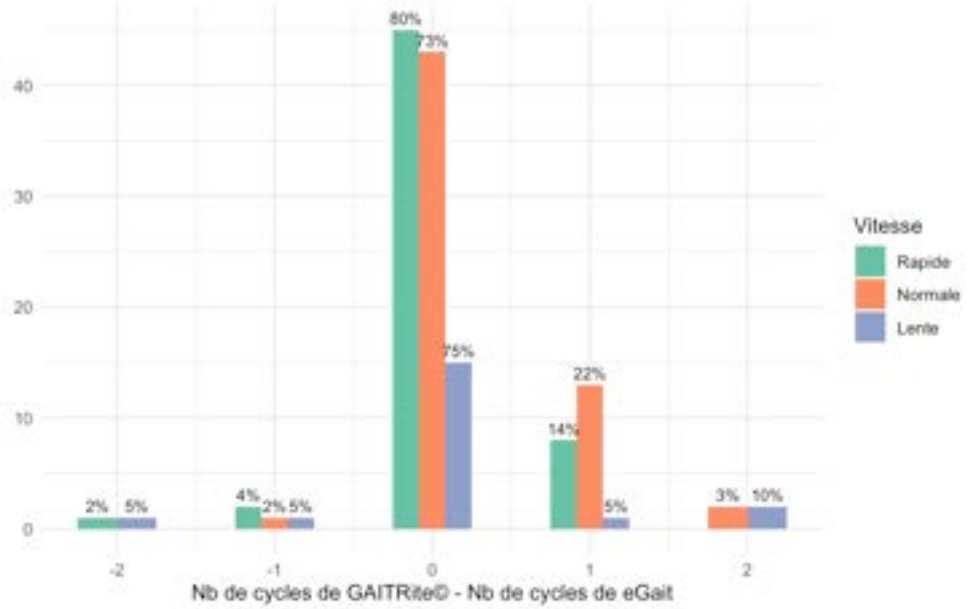


Figure 9.6: Différence du nombre de cycles de marche détectés entre les deux dispositifs, sur les sessions d'octobre et de juillet, par rapport à la vitesse de marche.

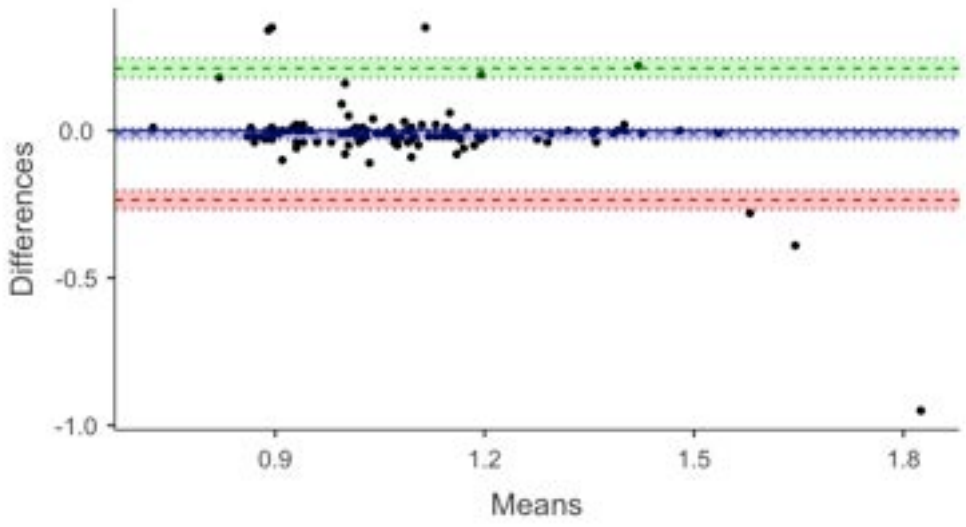


Figure 9.7: Diagramme de Bland-Altman pour la durée moyenne des cycles, sur les sessions de novembre et février.

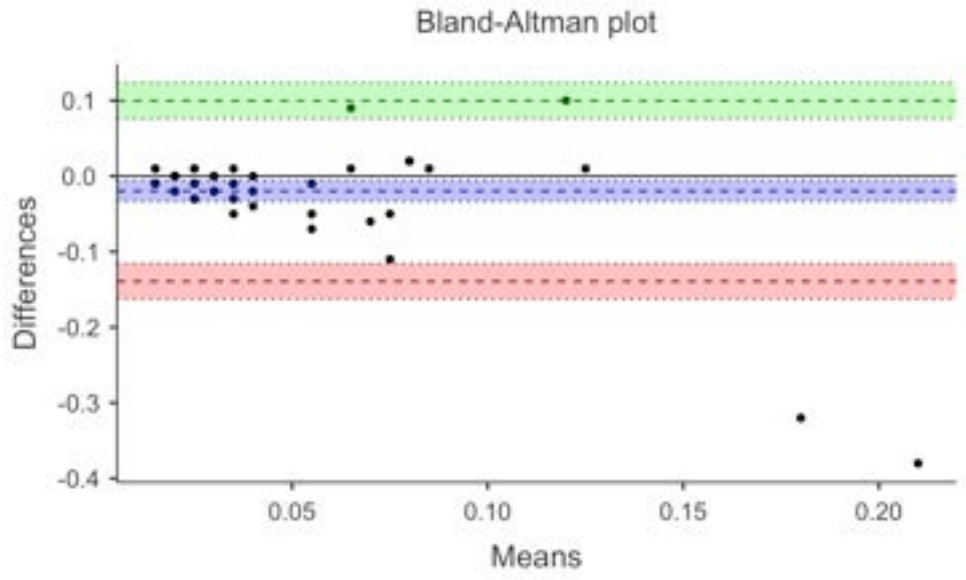


Figure 9.8: Diagramme de Bland-Altman pour la variation de la durée des cycles, sur les sessions de juin et juillet.

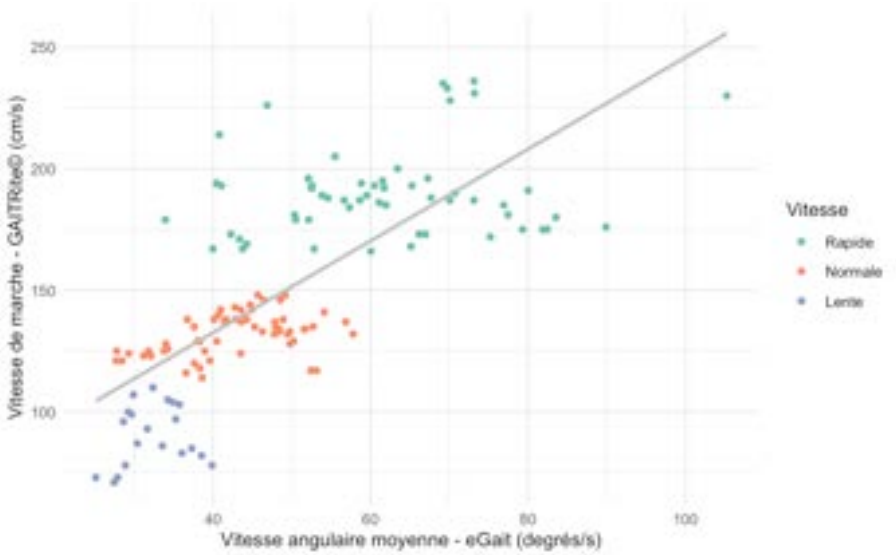


Figure 9.9: Vitesse angulaire moyenne (eGait) et vitesse de marche (GAITRite©) pour les sessions de novembre et février, par rapport à la vitesse.

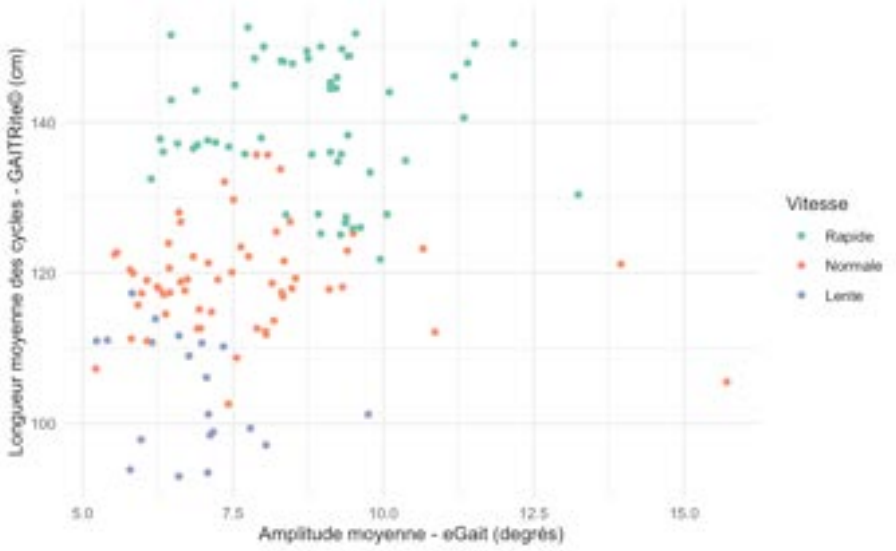


Figure 9.10: Amplitude moyenne (eGait) et longueur moyenne des cycles (GAITRite©) pour les sessions de novembre et février, par rapport à la vitesse.

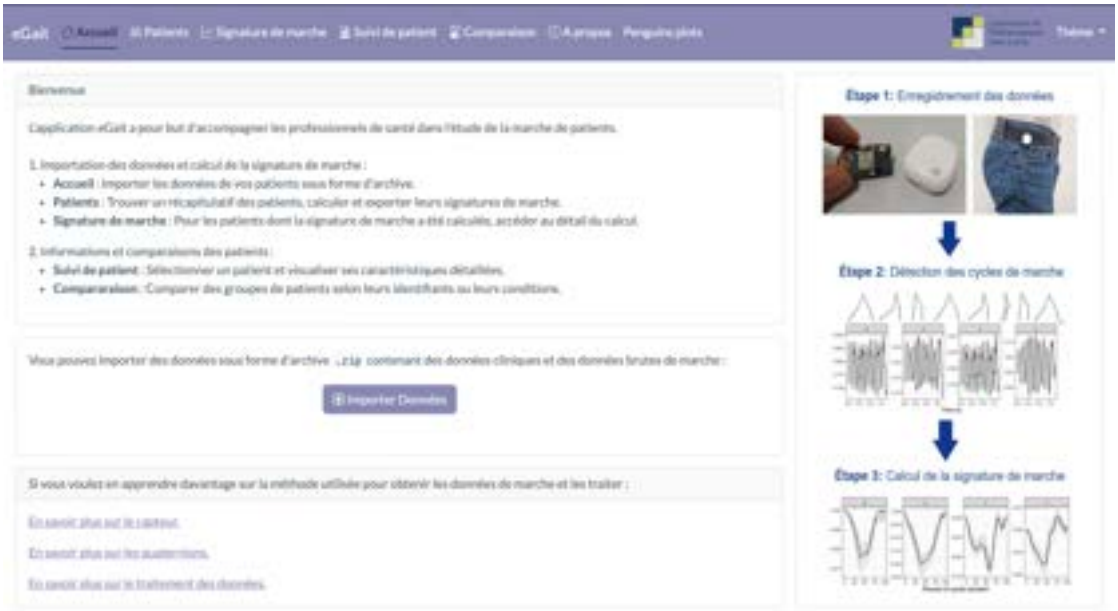


Figure 9.11: Page d'accueil de l'application web en développement.

Figure 9.12: Page regroupant les patients de l'application web en développement.

Listing 9.1 Création de la métrique de Youden pondérée

```
# pour avoir le nom de la classe positive
event_col <- function(xtab, event_level) {
  if (identical(event_level, "first")) {
    colnames(xtab)[[1]]
  } else {
    colnames(xtab)[[2]]
  }
}

# implémentation de la métrique
weighted_youden_impl <- function(truth, estimate, event_level) {
  xtab <- table(estimate, truth)       # matrice de confusion
  col <- event_col(xtab, event_level)  # nom de la classe positive
  col2 <- setdiff(colnames(xtab), col) # nom de la classe négative

  tp <- xtab[col, col]      # vrais positifs
  tn <- xtab[col2, col2]    # vrais négatifs
  fp <- xtab[col, col2]     # faux positifs
  fn <- xtab[col2, col]     # faux négatifs

  sen <- tp / (tp + fn)     # sensibilité
  spe <- tn / (tn + fp)     # spécificité

  2 * (0.7 * sen + 0.3 * spe) - 1  # métrique avec poids choisis
}

# implémentation de l'utilisation sur vecteur
weighted_youden_vec <- function(truth,
                           estimate,
                           estimator = NULL,
                           na_rm = TRUE,
                           case_weights = NULL,
                           event_level = "first",
                           ...) {
  estimator <- finalize_estimator(truth, estimator)

  check_class_metric(truth, estimate, case_weights, estimator)

    if (na_rm) {
    result <- yardstick_remove_missing(truth, estimate, case_weights)

    truth <- result$truth
    estimate <- result$estimate
    case_weights <- result$case_weights
  } else if (yardstick_any_missing(truth, estimate, case_weights)) {
    return(NA_real_)
  }

  weighted_youden_impl(truth, estimate, event_level)
}
```

Listing 9.2 Création de la métrique de Youden pondérée

```r
# définition du métrique
weighted_youden <- function(data, ...) {
  UseMethod("weighted_youden")
}

# préciser que cette métrique est à maximiser
weighted_youden <- new_class_metric(weighted_youden, direction =
 ↪  "maximize")

# implémentation de l'utilisation sur data frame
weighted_youden.data.frame <- function(data,
                                       truth,
                                       estimate,
                                       estimator = NULL,
                                       na_rm = TRUE,
                                       case_weights = NULL,
                                       event_level = "first",
                                       ...) {
  class_metric_summarizer(
    name = "weighted_youden",
    fn = weighted_youden_vec,
    data = data,
    truth = !!enquo(truth),
    estimate = !!enquo(estimate),
    estimator = estimator,
    na_rm = na_rm,
    case_weights = !!enquo(case_weights),
    event_level = event_level
  )
}
```

Listing 9.3 Fonction retournant les poids de la méthode *inverse class frequency*

```
# retourne un vecteur avec un poids pour chaque classe
# ordonné de la classe avec le moins d'observations à la classe avec le
 ↪  plus d'observations
inverse_class_freq <- function(class) {
  n_samples <- length(class)
  classes <- levels(class)
  n_classes <- length(classes)

  # on calcule le nombre d'observatin dans chaque classe pour ordonner
   ↪  les poids
  n_samples_classes <- 1:n_classes
  ordered_classes <- 1:n_classes
  for (i in 1:n_classes){
    n_samples_classes[i] <- length(class[class %in% classes[i]])
  }
  ordered_classes <- classes[order(n_samples_classes)]

  # on créé le vecteur de poids
  weights <- 1:n_classes
  for (i in 1:n_classes){
    n_samples_i <- length(class[class %in% ordered_classes[i]])
    weights[i] <- n_samples/(n_classes*n_samples_i)
  }
  return(weights)
}
```

Listing 9.4 Sélection du point central de la plage de points prédits

```
select_central_points <- function(predictions) {

  if(length(predictions)==0) {return()}
  if(length(predictions)==1){return(predictions)}

  selected_points <- c()
  window <- c()

  for (i in 1:(length(predictions)-1)) {
    current_point <- predictions[i]
    next_point <- predictions[i+1]

    # si nous sommes toujours dans la même plage
    if (next_point-current_point < 0.1) {
      # on conserve le point et on continue
      window <- c(window, current_point)
    }
    else {
      # si on est arrivé au bout de la plage, on sélectionne le point
      ↪ central
      central_point <- window[floor(length(window)/2)]
      selected_points <- c(selected_points, central_point)
      # puis on reset notre plage
      window <- c()
    }
  }
  return(selected_points)
}
```

Références

[1] Cédric Annweiler et al. “Risk factors for falls in geriatric inpatients: interaction between increased stride time variability and history of falls”. In : *Annales de Gérontologie* 2.1 (2009), p. 1-3.

[2] Olivier Beauchet et al. “Poor gait performance and prediction of dementia: results from a meta-analysis”. In : *Journal of the American Medical Directors Association* 17.6 (2016), p. 482-490.

[3] Biometrics. *GAITRite pro*. URL : https://biometrics.fr/web/nos-technologies/78-gaitrite.html.

[4] J. Martin Bland et Douglas G. Altman. “Statistical Methods For Assessing Agreemment Between Two Methods Of Clinical Measurement”. In : *The Lancet* 327.8476 (1986), p. 307-310. ISSN : 0140-6736. DOI : https://doi.org/10.1016/S0140-6736(86)90837-8.

[5] Leo Breiman. *Classification and regression trees*. Routledge, 2017.

[6] N. V. Chawla et al. “SMOTE: Synthetic Minority Over-sampling Technique”. In : *Journal of Artificial Intelligence Research* 16 (juin 2002), p. 321-357. ISSN : 1076-9757. DOI : 10.1613/jair.953. URL : http://dx.doi.org/10.1613/jair.953.

[7] Stanley Lemeshow David W. Hosmer. *Applied Logistic Regression*. Wiley Series in Probability et Statistics, 2000.

[8] Pierre Drouin. “”Amélioration du suivi des patients atteints de maladies neurodégénératives à l’aide d’objets connectés””. Theses. Nantes Université, sept. 2022. URL : https://theses.hal.science/tel-04046965.

[9] Pierre Drouin et al. “Semi-supervised clustering of quaternion time series: Application to gait analysis in multiple sclerosis using motion sensor data”. In : *Statistics in Medicine* 42.4 (2023), p. 433-456.

[10] GAITRite. *The GAITRite Gold Standard*. URL : https://www.gaitrite.com.

[11] William Rowan Hamilton. “LXXVIII. On quaternions; or on a new system of imaginaries in Algebra”. In : *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 25.169 (1844), p. 489-495. DOI : 10.1080/14786444408645047. URL : https://doi.org/10.1080/14786444408645047.

[12] Mahdi Hashemi et Hassan Karimi. “Weighted Machine Learning”. In : *Statistics, Optimization & Information Computing* 6 (nov. 2018). DOI : 10.19139/soic.v6i4.479.

[13] Haibo He et Edwardo A. Garcia. “Learning from Imbalanced Data”. In : *IEEE Transactions on Knowledge and Data Engineering* 21.9 (2009), p. 1263-1284. doi : 10.1109/TKDE.2008.239.

[14] Haibo He et al. “ADASYN: Adaptive synthetic sampling approach for imbalanced learning”. In : (2008), p. 1322-1328. doi : 10.1109/IJCNN.2008.4633969.

[15] B. W. Silverman J. O. Ramsay. *Functional Data Analysis*. Springer New York, NY, 2005. doi : https://doi.org/10.1007/b98888.

[16] M Kuhn. *Applied Predictive Modeling*. Springer New York, 2013.

[17] Max Kuhn. *Using case weights with tidymodels*. url : https://www.tidyverse.org/blog/2022/05/case-weights.

[18] Klervi Le Gall et al. “Generation of synthetic gait data: application to multiple sclerosis patients’ gait patterns”. In : *The International Journal of Biostatistics* (2024).

[19] Dan-ling Li et al. “Weighted Youden index and its two-independent-sample comparison based on weighted sensitivity and specificity”. In : *Chinese medical journal* 126.6 (2013), p. 1150-1154.

[20] Thomas Lumley. *”Weights in statistics”*. url : https://notstatschat.rbind.io/2020/08/04/weights-in-statistics/.

[21] M. D. McKay, R. J. Beckman et W. J. Conover. “A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code”. In : *Technometrics* 21.2 (1979), p. 239-245. issn : 00401706. url : http://www.jstor.org/stable/1268522 (visité le 13/08/2024).

[22] Hylton B Menz et al. “Reliability of the GAITRite® walkway system for the quantification of temporo-spatial parameters of gait in young and older people”. In : *Gait & Posture* 20.1 (2004), p. 20-25. issn : 0966-6362. doi : https://doi.org/10.1016/S0966-6362(03)00068-7. url : https://www.sciencedirect.com/science/article/pii/S0966636203000687.

[23] Manuel Montero-Odasso et al. “Gait velocity as a single predictor of adverse events in healthy seniors aged 75 years and older”. In : *The Journals of Gerontology Series A: Biological Sciences and Medical Sciences* 60.10 (2005), p. 1304-1309.

[24] Ashwin Narayan. *How to Integrate Quaternions*. 2017. url : https://www.ashwinnarayan.com/post/how-to-integrate-quaternions/.

[25] Wikipedia. *Conversion between quaternions and Euler angles*. url : https://en.wikipedia.org/wiki/Conversion_between_quaternions_and_Euler_angles.

[26] Wikipedia. *Gimbal lock*. url : https://en.wikipedia.org/wiki/Gimbal_lock.

[27] Wikipedia. *Test des rangs signés de Wilcoxon*. url : https://fr.wikipedia.org/wiki/Test_des_rangs_sign%C3%A9s_de_Wilcoxon.