UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

SCIENCES DU MOUVEMENT ET DU SPORT
BEWEGUNGS- UND SPORTWISSENSCHAFTEN

Comparaison qualitative du coup droit au tennis entre le jugement d'un expert et des algorithmes d'apprentissage.

Travail de fin d'études en vue de l'obtention du titre de
Master of Science en sciences du sport
Option enseignement

déposé par

Fanny Mauron

à
l'Université de Fribourg, Suisse
Faculté des sciences et de médecine
Section Médecine
Département des neurosciences et sciences du mouvement

en collaboration avec la
Haute école fédérale de sport de Macolin

Référent
Prof. Jean-Pierre Bresciani

Conseiller / Conseillère
Prof. Thibaut Le Naour

Fribourg, Juin 2019

Table des matières

Résumé

Connaissance. De nombreuses nouvelles avancées dans le domaine informatique ont permis de développer l'utilisation d'algorithme dans des domaines de la vie quotidienne. Le but de ce travail est de mettre en relation le Machine Learning et le domaine sportif. Il s'agit de définir s'il est possible de prédire un jugement qualitatif humain par le biais d'un apprentissage machine supervisé. Comme mouvement de base, nous avons choisi le coup droit au tennis.

Méthodes. 559 coups droits exécutés par des sujets hommes experts et débutants au tennis ont été capturés en 3D. Ces mouvements ont ensuite été jugés qualitativement par les experts qui leur ont attribués une note de 0 à 9. Nous avons ainsi obtenu une base de données d'apprentissage pour le Machine Learning. Deux algorithmes de classification et de régression ont été choisis, les k-plus proches voisins et les arbres de décision. Nous avons également testé une méthode de régression linéaire et polynomiale ainsi qu'une méthode de deep-learning et de boosting du gradient. Tous ces algorithmes ont prédit les notes pour les données de test et une comparaison a été faite avec les vraies valeurs. Nous avons calculé le score de prédiction pour les méthodes de classification et le coefficient de détermination R^2 pour les méthodes de régression. L'erreur quadratique moyenne ainsi que l'erreur absolue moyenne ont également été calculées afin de pouvoir comparer les différentes méthodes d'apprentissage.

Résultats. Les méthodes de régression sont meilleures que les méthodes de classification. Par-dessus tout, la méthode de boosting du gradient est la méthode qui prédit les notes avec la plus grande justesse. C'est la seule méthode qui obtient une erreur moyenne absolue plus petite que 0,7 et un coefficient R^2 de 0.78. Nous avons pu définir qu'il existe suffisamment de cohérence dans les différents jugements des experts pour que l'algorithme d'apprentissage puisse généraliser les associations et prédire avec une précision acceptable les notes de nouveaux mouvements. Cependant, les résultats étant très diversifiés, nous n'avons pas pu définir précisément les caractéristiques importantes qui ont permises aux experts de distinguer qualitativement les mouvements entre eux. Plusieurs problèmes ont été relevés dans le jeu de données d'apprentissage tels que le manque d'homogénéité dans la répartition des données dans les classes et la rigueur dans le jugement des experts.

Conclusion. Le Machine Learning offre de très larges possibilités d'utilisations dans le domaine sportif. La récolte des données et la préparation des données d'apprentissage sont des étapes très importantes à ne pas sous-estimer pour que les résultats soient précis et explicables.

1 Introduction

Durant ces dernières années, nous remarquons des progrès phénoménaux dans le domaine informatique et technologique. Ainsi, la société actuelle s'oriente de plus en plus vers les nouvelles technologies et l'intelligence artificielle. En particulier, le Machine Learning (ML) ou apprentissage automatique, est une nouvelle branche de l'informatique qui s'intéresse aux algorithmes d'apprentissage. Cela consiste à la mise au point d'algorithmes qui, à partir de données d'apprentissage préalablement acquises, permettent de déterminer les caractéristiques de nouvelles données sans aucune intervention humaine.

Déjà en 1950, Arthur Samuel développe grâce au ML un programme qui joue au jeux des Dames et s'améliore par lui-même. Quelques années plus tard, le « Deep Blue », un superordinateur de la société IBM, bat le champion du monde d'échec. L'attrait pour le ML est lancé et les progrès dans le domaine informatique sont énormes. L'utilisation de tels algorithmes s'étend à de nombreux domaines, par exemple, dans les diagnostics médicaux pour prédire une rechute psychotique dans la schizophrénie (Fond et al., 2019) ou encore, pour estimer la cinématique scapulaire chez des patients afin de prévenir des blessures et d'adapter les traitements (Nicholson et al., 2019). Les systèmes immunitaires artificiels sont des outils puissants pour la reconnaissance de formes (Khelil & Benyettou, 2010) mais également utilisés pour la sécurité informatique et la détection de fraudes dans les entreprises. Le filtre anti-spam utilisé dès les années 1990 est une application du ML qui a touché un très large public et amélioré le quotidien de millions d'individus (Géron, 2017).

Selon Lemberger, Batty, Morel et Raffaëlli (2016), le ML est un « ensemble d'outils statistiques ou géométriques et d'algorithmes informatiques qui permettent d'automatiser la construction d'une fonction de prédiction f à partir d'un ensemble d'observation » (p.112). Grâce à ces outils, nous cherchons à faire une prédiction qui correspond à l'évaluation $f(x)=y$ de la fonction de prédiction f sur les variables prédictives d'une observation x. Le processus commence en premier lieu par la sélection d'un algorithme de ML. Puis, une fonction de prédiction f est produite grâce à l'entrainement de l'algorithme. Cet entrainement se base sur des données préalablement acquises qui constituent le jeu d'apprentissage. Enfin, une prédiction pour une nouvelle observation peut être faite à partir de cette fonction (Lemberger et al., 2016).

1.1 Les données d'apprentissage

Dans le ML, la récolte de données est une étape cruciale. Chaque observation d'un phénomène passé peut être décrite par deux types de variables :

- les variables prédictives : variables à partir desquelles les prédictions sont faites.
- les variables cibles : variables dont on souhaite prédire la valeur pour un événement qui n'a pas encore été observé.

La quantité de données à récolter est difficile à estimer. Toutefois, les données doivent représenter au mieux toutes les caractéristiques possibles présentes durant l'apprentissage. Le volume de données d'apprentissage croît en même temps que la précision souhaitée et le nombre de variables prédictives, mais également avec la complexité du modèle. La fonction de prédiction *f* permet d'approximer les valeurs cibles à partir des valeurs prédictives. Plus le modèle est précis, plus la prédiction est fiable (Lemberger et al., 2016).

Prenons un exemple simple pour illustrer ceci : Bruno a commencé le tennis dans un nouveau club et commence les compétitions. Les variables prédictives sont ses années d'expériences, son niveau de jeu, son niveau de santé, son âge, etc. Les variables cibles sont quant à elles le nombre de victoires, de défaites ou le meilleur classement obtenu par Bruno suite à son début d'entrainement. Plus nous avons d'informations sur sa personne et sur les autres membres de la fédération de tennis, plus les prédictions faites seront précises et fiables.

Dans n'importe quel apprentissage, on cherche à maximiser la précision de la prédiction. Pour cela, il est astucieux de créer 3 ensembles à partir des données récoltées. L'ensemble d'apprentissage est la base d'apprentissage pour l'algorithme et il nous permet d'ajuster le modèle. L'ensemble de validation sert à estimer l'erreur de prédiction pour le modèle sélectionné. On cherche ici à minimiser $|f(x) - y|$ soit la différence entre la variable prédite *f(x)* et la vraie valeur y. Enfin, l'ensemble de tests est utilisé pour évaluer l'erreur de généralisation du modèle final. Il n'y a pas de lois précises pour le choix des tailles des ensembles mais la séparation se fait généralement en 75-25-25% pour les données d'apprentissage, de validation et de test (Hastie, Tibshirani & Friedman, 2009).

1.2 Les types d'apprentissage

Les systèmes d'apprentissage peuvent être classés en fonction de la nature de la supervision acquise durant la phase d'entrainement.

On parle d'apprentissage supervisé lorsque les données d'entrainement fournies à l'algorithme sont préalablement classées. Le système suit alors le modèle des données qui ont été auparavant étiquetées par un expert. On parle de jeu d'entrainement étiqueté (Géron, 2017).

Les paires entrée/sortie permettent à l'algorithme de faire des liens entre les données acquises. Comme exemple, nous pouvons prendre les systèmes de détections de fraudes dans les transactions faites par carte de crédit ou la reconnaissance faciale sur les téléphones portables.

Un deuxième type d'apprentissage est l'apprentissage non-supervisé. Les données fournies sont uniquement les variables prédictives d'entrée. On ne fournit aucune donnée de sortie à l'algorithme (Müller & Guido, 2018).

L'objectif du premier apprentissage est de généraliser les associations que l'on a observées entre les variables explicatives (entrée) et les variables cibles (sortie) afin de pouvoir construire une fonction de prédiction f. Dans l'apprentissage non-supervisé, le but est que le système réussisse par lui-même à regrouper en catégories les données observées. C'est ensuite à l'humain d'interpréter les catégories identifiées (Lemberger et al,. 2016).

1.3 Les algorithmes de classification et de régression

Dans ce travail, nous avons essayé plusieurs modèles de traitements des données. En premier lieu, définissons un modèle de classification. Lors d'une classification, les variables traitées sont qualitatives. Elles se répartissent en deux groupes : les variables nominales qui se réfèrent à une classe définie par un nom et les variables ordinales qui sont des variables ordrées et comparables entre elles. Les variables sont réparties en « classes d'appartenance ». En deuxième lieu, intéressons-nous au modèle de régression. Contrairement à la classification, la variable prédictive, tout comme la variable cible, sont des variables quantitatives. On parle de données contenant des valeurs mesurables. Un modèle de régression linéaire multiple se définit par une fonction de prédiction *f* qui est une combinaison linéaire des variables prédictives et qui modélise la relation entre nos variables (x_i, y). Elle prend la forme suivante :

$$f(x) = \ y = a_0 + a_1x_1 + a_2x_2 + \ldots + a_nx_n,$$

où x_i sont les variables explicatives de y.

L'estimation des coefficients a_i se fait souvent grâce à la méthode des moindres carrés.

Nos données entrée/sortie (x_i, y_i) forment un nuage de points que la droite de régression doit prédire le plus précisément possible. On cherche à réduire les résidus $r = y - y_{prédit}$ (Goldfarb et Pardoux, 2011). Pour cela, nous allons utiliser la regression de la droite des moindres carrées . Il s'agit bien d'une droite qui minimise le carré des résidus que l'on définit ainsi :

$$\sum_{i=1}^{n}(y_i - y_{prédit})^2 = \sum_{i=1}^{n} r_i^2 \qquad (1)$$

Nous allons définir les différents algorithmes utilisés dans ce travail.

1.3.1 K-Nearest Neighbors. L'algorithme des K-Nearest Neighbors (K-NN ou des k-plus proches voisins) est une méthode permettant de faire de la classification et de la régression. L'algorithme stocke les différentes possibilités de valeurs et prédit une valeur numérique en fonction d'une mesure de similarité (par exemple : en fonction de la distance). En quelque sorte, il crée une carte de dimension égale au nombre d de variables décrivant les n observations $x^{(1)}, x^{(2)}, .. x^{(n)}$. Chaque point dans cet espace qui a d dimensions, équivaut à une observation $x^{(i)}$. Cette méthode cherche ensuite les k voisins les plus proches des données de test afin de pouvoir prédire une valeur à chacune de ces nouvelles données (Lemberger et al., 2016).

Prenons par exemple des paires qui constituent les données d'entrainement (x_i, y_i), $i = 1,2, \ldots n$ qui prennent valeur dans $\mathbb{R}^d$ (si on suppose avoir d paramètres pour une prédiction). Pour une variable cible *Xm*, l'algorithme *k*-NN prédit Y_m en cherchant les k valeurs voisines les plus proches de *Xm*. La moyenne de ces *k* valeurs nous donne la prédiction finale.

Les deux paramètres importants dans cet algorithme sont donc le nombre de voisins et la manière de mesurer la distance entre les points de données (Müller & Guido, 2018). Dans le cas de valeurs continues, le modèle utilisé est celui de régression basé sur les plus proches voisins. Les fonctions de distances les plus communes sont les suivantes :

- La distance euclidienne : $\sqrt{\sum_{i=1}^{k}(x_m - x_i)^{\,2}}$ (2)
- La distance de Manhattan : $\sum_{i=1}^{k}|x_m - x_i|$
- La distance de Minkovsky : $\sum_{i=1}^{k}(|x_m - x_i|^p)^{\frac{1}{p}}$

Notons que lorsque $p = 2$, la distance de Minkovsky égale la distance euclidienne. Le nombre *k* de voisins doit être choisi de manière à éviter un surapprentissage des données et cette valeur peut grandement varier d'un modèle à l'autre (Singh, 2018). Pour la définir, il est possible de calculer les valeurs obtenues pour différents *k* et de comparer les erreurs d'apprentissage et les erreurs sur l'ensemble de validation. Singh (2018) propose de prendre la valeur de *k* pour laquelle la racine de l'erreur quadratique moyenne (RMSE) est minimisée (formule donnée au chapitre 1.4).

1.3.2 Arbres de décision. Un autre modèle est l'arbre de décision. L'intérêt de cet algorithme résulte dans le fait qu'il peut effectuer des tâches de régression et de classification. Globalement, ils sont structurés par une série de questions (appelées tests ou critères de segmentation) qui servent à amener à une décision finale. La profondeur du modèle correspond au nombre de nœuds possibles par branche. A chaque nœud de l'arbre correspond une question

et chaque branche sortante désigne une réponse envisageable. L'algorithme étudie les tests possibles et décide des nœuds les plus pertinents pour avoir un maximum d'informations sur la variable cible. L'arbre de décision binaire est le résultat d'un processus de partitionnement récursif des données où on cherche à ranger correctement un ensemble $x^{(1)}, x^{(2)}, \dots, x^{(n)}$ de n observations dotées d'étiquettes $y^{(1)}, y^{(2)}, \dots, y^{(n)}$ (Lemberger et al., 2016).

L'une des qualités de cet algorithme est sa rapidité et le fait qu'il ne demande pas de recalibrage des données. C'est un modèle basique qui reste très visuel et facilement explicable (Müller & Guido, 2018).

Il est toutefois difficile de trouver un arbre optimal qui minimise le nombre de questions. De plus, il est important que les feuilles soient les plus homogènes possible. Cela signifie que les observations à ces nœuds n'appartiennent qu'à une seule classe. On recherche alors des critères de segmentations qui permettent au fur et à mesure du processus d'augmenter l'homogénéité des feuilles (Lemberger et al., 2016).

1.3.3 Apprentissage profond. Le Deep Learning (ou apprentissage profond) est un ensemble d'algorithmes automatiques. Parmi ceux-ci, les réseaux de neurones (RN) permettent à l'algorithme de résoudre de nombreuses tâches et d'avoir un apprentissage pour trouver une transformation f. En se basant sur des données d'entrée $x = (x_1, x_2, \dots x_n)$ et de sortie $y = (y_1, y_2, \dots y_n)$, elle occasionne une prédiction $f(x_i)$, la plus proche possible des valeurs observées y_i. La transformation f a lieu au moyen d'une suite de transformations linéaires et non-linéaires. Lorsque l'interconnexion entre neurones est définie par des couches successives avec des interconnexions limitées aux couches adjacentes, on désigne cette architecture comme un perceptron multicouche (MLP). Ainsi, chaque neurone d'une couche est relié à tous les neurones des autres couches adjacentes et ce lien est défini par un poids w_{ij}. Les résultats apparaissent sur la dernière couche nommée couche de sortie (Lemberger et al., 2016).

Cet algorithme, proposé par Rosenblatt (1958), est réalisé sur le modèle du neurone biologique. Il s'agit d'un modèle de classes multiples qui représente l'association entre une observation et une sortie réelle, grâce à une fonction multivariée (Massih-Reza, 2015). Certaines couches peuvent contenir des fonctions non-linéaires. On les nomme couches cachées.

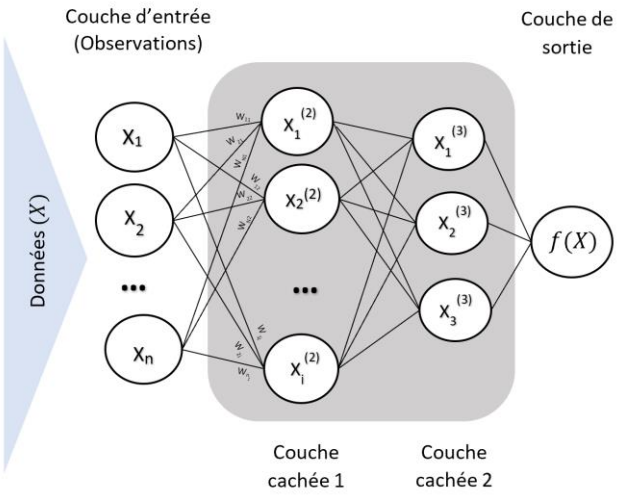


Figure 1. Représentation d'un MLP à 4 couches.

Pour revenir à un réseau simple, pour N neurones disposés dans une seule couche cachée, un perceptron est une fonction $f: \mathbb{R}^d \rightarrow \mathbb{R}$ de la forme :

$$f(x) = \sum_{i=1}^{N} \alpha_i \sigma(w_i^T x + b_i) ,$$

où $w_i \epsilon \mathbb{R}^d$ est le poids du neurone appliqué sur l'observation $x \epsilon \mathbb{R}^d$, σ est la fonction d'activation sigmoïdale[1] non-linéaire, $\alpha_i \epsilon \mathbb{R}$ est le poids du réseau appliqué sur chaque sortie des neurones dans la couche cachée et $b_i \epsilon \mathbb{R}$ est le biais pour le neurone i (Mcneela, s.d.).

Dans le cas d'un MLP, on procède récursivement couche par couche. Ainsi pour voir l'activation d'un neurone $x_i^{(n)}$ de la couche n :

1) On calcule la somme des activations d'un neurone $x_i^{(n-1)}$ présent sur la couche précédente, pondérée par les poids w_{ij}.
2) On applique la fonction d'activation $\sigma(x)$ à cette somme

(Lemberger et al., 2016).

Parmi ces réseaux de neurones, nous distinguons une sous-catégorie : les réseaux de neurones convolutifs qui sont spécialement conçus pour traiter des images en entrée. Ces algorithmes d'apprentissage profond ont été appliqués avec succès à plusieurs tâches de reconnaissance d'images, classification d'objets, d'animaux et de végétaux (Wagenaar, Okafor, Frencken & Wiering, 2017). Dans ce domaine, il existe un théorème intéressant, le théorème d'approximation universelle qui s'énonce ainsi dans l'ouvrage de Lemberger et al. (2016) :

[1] Une fonction sigmoïdale est une fonction $\sigma: \mathbb{R} \rightarrow \mathbb{R}$ telle que :

$$\sigma(t) = \frac{1}{1+e^{-x}},\ \sigma(t) \rightarrow \begin{cases} 1\ quand\ t \rightarrow \infty \\ 0\ quand\ t \rightarrow -\infty \end{cases}$$

> Toute fonction $f(x)$ peut être approximée avec une précision arbitraire par un RN formé d'une seule couche intermédiaire (trois couches si l'on compte l'entrée et la sortie) à condition toutefois d'utiliser un nombre suffisant de neurones dans la couche intermédiaire. (p. 138)

1.3.4 Boosting de gradient. Dans ce travail, nous allons également nous intéresser à un algorithme de boosting. Il s'agit d'une méthode qui « entraine des prédicteurs l'un après l'autre, chacun s'efforçant de corriger son prédécesseur » (Gérou, 2017 p.183). Parmi les plus connues, nous pouvons citer l'AdaBoost et le boosting du gradient. Nous nous baserons sur ce dernier dans ce travail. Assigné à une tâche de régression basée sur les arbres de décision, on le nomme le Gradient Boosted Regression Trees. Cette méthode qui progressivement construit un modèle additif, ajuste chaque nouvel arbre par rapport aux erreurs résiduelles du précédent (Géron, 2017). Cette méthode est « souvent le vainqueur dans les compétitions d'apprentissage automatique » (Müller & Guido, 2018, p.92).

1.4 Les qualités et dangers des algorithmes de ML

Selon Ted Dunning (congrès Big Data Paris, 2014, cité dans Lemberger et al., 2016), un algorithme de ML est jugé comme bon dans le domaine économique s'il regroupe les qualités suivantes : déployabilité, robustesse, transparence, adéquation aux compétences et proportionnalité. Ce dernier terme se rapporte directement à la quantité d'énergie et de temps investie pour l'optimisation de l'algorithme par rapport au gain apporté. Dans le domaine informatique, cela s'accorde à parler de performance de l'algorithme. Les chercheurs souhaitent mettre en avant des algorithmes qui « généralisent les associations apprises durant la phase d'entrainement à de nouvelles observations » (Lemberger et al., 2016, p.115) et cela de façon optimale.

Un premier danger est le surapprentissage (o*verfitting*) du ML. Cela se produit lorsqu'un modèle est trop complexe et que les données d'apprentissage ne nous donnent pas suffisamment d'informations. Le modèle ne peut pas généraliser l'apprentissage correctement pour de nouvelles données car il suit trop précisément les particularités de l'ensemble d'apprentissage. Au contraire, si l'apprentissage est basé sur un modèle trop simple qui ne permet pas de représenter tous les aspects et la variabilité du jeu d'apprentissage, on parle de sous-apprentissage (*underfitting*). Il faut donc trouver un juste milieu au niveau de la complexité de notre modèle de manière à éviter de tomber dans un des deux extrêmes (Müller & Guido, 2018).

Les difficultés d'un apprentissage automatique résident dans plusieurs points. La première difficulté est le nombre de données. La plupart des algorithmes ont besoin d'énormément de données pour travailler correctement. De plus, pour que les prédictions soient bonnes, il est important que ces données soient de bonne qualité et représentatives afin de les généraliser à de nouveaux cas. Il faut s'assurer que nos valeurs prédictives ne contiennent pas d'erreurs; supprimer les données aberrantes et faire attention au bruit dû à de mauvaises mesures (Géron, 2017) .

Pour analyser la précision des modèles, nous devons faire appel à plusieurs outils. Dans une méthode de régression, la mesure de vérification utilisée classiquement est la racine carrée de l'erreur quadratique moyenne (RMSE). Pour un nombre n de données d'observation, $x^{(i)}$ le vecteur des valeurs des variables pour la i^{ème} observation, $y^{(i)}$ la valeur de sortie pour cette observation et f la fonction de régression, la formule est définie ainsi :

$$RMSE(X,f) = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(f(x^{(i)}) - y^{(i)})^2}. \qquad (3)$$

C'est une façon de calculer la distance entre le vecteur des valeurs cibles et celui de prédictions afin d'avoir une idée de l'importance des erreurs de prédictions du système (Géron, 2017).

Géron (2017) souligne également que la RMSE est sensible aux valeurs s'écartant de la normale ainsi il propose d'utiliser dans ce cas l'erreur absolue moyenne (MAE) :

$$MAE(X,f) = \frac{1}{n}\sum_{i=1}^{n}|f\left(x^{(i)}\right) - y^{(i)}|. \qquad (4)$$

Pour des données continues, le score R^2 ou coefficient de détermination, est un bon moyen de vérifier la qualité d'une prédiction pour un modèle de régression (Müller & Guido, 2018). Il décrit la force de la relation linéaire entre les données prédites et les données d'observation correspondantes. Ce coefficient de détermination $R^2 \in [0,1]$ est défini par :

$$R^2 = 1 - \sum_{i=1}^{n}\frac{(y_{itest} - y_{iprédit})^2}{(y_{itest} - \overline{y})^2} \qquad \text{(Barnston, 1992)}$$

$$= \frac{\sum_{i=1}^{n}(y_{iprédit} - \overline{y})^2}{\sum_{i=1}^{n}(y_{itest} - \overline{y})^2} \qquad (5) \quad \text{(Goldfarb \& Pardoux, 2011)}$$

où $\overline{y}$ est la moyenne des mesures et n le nombre de mesures testées.

Il s'agit là de la proportion de la variation de Y expliquée par la droite des moindres carrés.

Si le score est égal à 1, la corrélation est linéairement parfaite. S'il n'y a pas de prédiction linéaire, alors le score sera 0 (Barnston, 1992). Il est alors intéressant de calculer le coefficient de corrélation $r \in [-1,1]$ pour définir l'intensité et la direction de la relation linéaire entre les variables quantitatives (Champely, 2004).

1.5 Le Machine Learning et le sport

Aujourd'hui, de nouvelles technologies sont mises au profit du sport avec de nouveaux équipements (montres connectées, traceurs GPS, chaussures et vêtements intelligents, etc.) mais aussi directement au niveau du jeu tactique et de l'analyse du mouvement (analyse virtuelle des positions, prise de décision, etc.). Ces outils nous permettent de mieux cibler les méthodes d'entrainement et d'améliorer la performance sportive qui dépend de nombreux facteurs physiques et mentaux. En s'intéressant à l'aspect cinématique du mouvement, l'entraineur peut apporter un feedback précis sur la technique du mouvement. D'ailleurs, l'analyse du geste se développe dans de nombreux domaines comme la reconnaissance, la synthèse, la segmentation ou encore l'évaluation (Morel, 2018). Il est souvent difficile de mettre en place des analyses précises de la cinématique d'un mouvement car cela demande du matériel coûteux (caméras, marqueurs, etc.) et est limité à un environnement de laboratoire. De nouvelles méthodes sans marqueurs permettant des prises dans des environnements réelles émergent grâce au ML. Des chercheurs finlandais ont mis en place une méthode qui permet de recréer en 2D la cinématique de la course sous l'eau d'un sportif, sans aucune pose de marqueurs. L'algorithme est entrainé par une base de mouvements d'environs 500 données qui ont été enregistrées sous l'eau par une caméra embarquée sous un plan sagittal. Il prédit la position des articulations et recrée la cinématique du geste par une processus de Deep Learning (Cronin, Rantalainen, Ahtiainen, Hynynen & Waller, 2019).

Dans le domaine médical sportif, des scientifiques ont mis en place un algorithme de Deep Learning qui permet de diagnostiquer une déchirure complète des ligaments croisés antérieurs (LCA) du genou à partir d'IRM. Cette blessure est très commune dans le domaine du sport et le retour au sport après cette blessure est souvent éprouvant. Un bon diagnostic de la blessure est essentiel pour pouvoir la traiter. Toutefois, il n'est pas toujours facile de poser un diagnostic précis et fiable à 100%. L'algorithme basé sur un réseau de neurones convolutifs détecte une déchirure complète de LCA avec une précision de 96% (Chang, Wong & Rasiej, 2019).

Plus concrètement, des algorithmes ont été mis en place pour améliorer la méthode d'entrainement. Les performances d'un sportif sont directement liées à son entrainement et à sa forme physique. Il est important de mettre en place des séances qui amènent des stimulis

correspondants au seuil de résistance physique et psychique, au degré de tolérance et aux besoins du sportif considéré individuellement (Weineck, 1997). Mezyk et Unold (2010) ont utilisé le ML combiné à un modèle de logique floue pour modéliser l'entrainement de nageurs par unité. Przednowek, Iskra, Wiktorowicz, Krzeszowski et Maszczyk (2017) ont également mis en place une planification des charges d'entraînement pour un coureur de 400m haies en utilisant des réseaux de neurones artificiels. 29 variables ont été utilisées pour caractériser le coureur (âge, BMI, résultats récents et résultats espérés) et l'entrainement (mètres en vitesse maximum, mètres en endurance aérobie, temps d'exercice technique à la course, etc.). Le modèle qui en réside peut assister l'entraineur pour planifier les entrainements.

Un autre travail de Rygula (2005) détermine les charges d'entrainement grâce à un réseau neuronal artificiel et souligne dans son travail l'importance de l'individualisation des entrainements.

La prédiction de performance est un domaine qui peut également être amené grâce au ML. Dans le domaine du foot, un algorithme de réseaux de neurones à convolution profond a été élaboré dans le but de prédire les occasions de marquer des buts à partir des données de positions des joueurs. Les chercheurs ont également essayé de faire des prédictions avec un algorithme K-NN mais la précision était moindre par rapport à l'algorithme de réseaux de neurones à convolution profond qui a atteint une précision moyenne d'environ 67% (Wagenaar et al., 2017).

1.5.1 Le tennis.

Dans ce travail, nous avons décidé de travailler sur le coup droit au tennis. Le tennis est un sport complet qui demande beaucoup de capacités physiques. La coordination entre les mouvements physiques et les perceptions visuelles dans l'espace est primordiale. Pour pouvoir anticiper les trajectoires de balle, par exemple lors du service, le travail d'analyse du mouvement du corps de l'adversaire est essentiel (Borrel, 2012). La maitrise du coup droit est un élément fondamental pour un pratiquant. Il est intéressant de s'attarder sur l'aspect technique du mouvement et sa cinématique. Nous prendrons le cas d'un droitier.

Selon Borrel (2012), le mouvement se décompose en 3 parties : la préparation, la frappe et l'accompagnement. Durant la phase de préparation, la ligne d'épaule est horizontale et perpendiculaire par rapport au filet. La raquette est amenée vers l'arrière en dessinant un arc de cercle vers le haut et la tête de la raquette est positionnée à la verticale. Les épaules font une rotation vers l'arrière, opposées aux appuis des pieds qui sont semi-ouverts (orientation d'environs 45° par rapport à la ligne du filet, pied gauche devant). L'appui arrière de la jambe droite soutient le poids du corps et le regard reste dirigé vers la balle. La frappe peut être liftée,

à plat ou coupée. Ces techniques sont un peu différentes l'une de l'autre et pour ce travail d'analyse, nous nous attarderons uniquement sur la frappe à plat qui est un coup droit sans rotation de la balle. Ce type de frappe produit une grande vitesse et une accélération de la balle lors de l'impact au sol (Borrel, 2012).

Durant la phase de frappe, la raquette et l'avant-bras forment une ligne horizontale. La tête de la raquette reste haute et ne descend pas sous le poignet. Le bras gauche sert à équilibrer le corps; il est donc écarté à l'opposé du bras droit. Durant cette phase, on retrouve la mise en tension de la ceinture abdominale par la rotation inverse du haut et bas du corps. La balle doit être frappée dans la zone située entre la hanche et l'épaule.

Pour la dernière partie du mouvement que constitue l'accompagnement de la balle, le coude droit reste décollé du buste dans la direction de la trajectoire de balle. Le corps est dévissé sous l'action du bras gauche qui part de côté et du buste qui fait une rotation vers la gauche. La ligne d'épaule suit presque parallèlement la direction de la balle frappée. La tête de la raquette est conduite au-dessus ou dessous de l'épaule gauche, ou au niveau de la hanche gauche. Enfin, l'angle entre la raquette et l'avant-bras diminue et le poignet revient en position neutre.

1.6 Objectif du travail

Le but de ce travail est de faire une utilisation ciblée du ML dans le domaine sportif. En premier lieu, il a été envisagé de créer une base de données conséquente du mouvement du coup droit au tennis afin d'avoir un panel complet de la cinématique du geste. Ensuite, les experts ont analysé les mouvements des sujets et les ont notés sur des critères qualitatifs. La dernière partie repose sur du Machine Learning. Ainsi, nous nous sommes posé les questions suivantes :

- Est-ce que l'algorithme d'apprentissage et le jugement humain suivent une même logique ?
- Quels sont les critères qui amènent à ce qu'un coup droit au tennis soit jugé comme bon par un expert ?

Il est envisagé de :

a) Créer des algorithmes d'apprentissage qui permettent, à partir de notre base de données cinématique du mouvement et des jugements d'experts, de prédire la qualité d'un nouveau mouvement grâce à diverses méthodes de ML.

b) Comparer les prédictions de divers algorithmes de régression et de classifications.

c) Trouver les corrélations présentes au sein d'une classe de mouvements et ainsi extraire les caractéristiques (variables indépendantes) expliquant le mouvement en lien avec les évaluations des experts grâce à un algorithme de DecisionTree.

Le travail suivant est structuré en 5 parties définies ci-dessous. La première partie a mis en évidence le contexte de la présente étude. Dans la deuxième partie, une recherche approfondie des différentes méthodes de prédiction sera présentée. Les algorithmes seront construits à partir de la base de données d'observation et de la libraire sklearn présente sur Python. Plusieurs aspects seront traités tels que la classification au moyen d'arbres de décision, la régression polynomiale ou encore une méthode basée sur les réseaux de neurones. Ces méthodes de prédiction implémentées avec le logiciel Python 3.7 seront également décrites. Les algorithmes à appliquer afin d'effectuer les prédictions seront présentés en annexe. Dans la troisième partie, nous présenterons les résultats de mise en application de nos différents algorithmes sur notre base de données de test. La quatrième partie contiendra une discussion des résultats des prédictions. Nous présenterons les forces et faiblesses de la présente étude en répondant aux questions formulées en début de travail. Finalement, la dernière partie contiendra un résumé des principaux résultats de cette étude tout en proposant des ouvertures à des travaux futurs dans ce domaine.

2 Méthode

Dans ce chapitre, nous allons expliquer toutes les étapes de l'expérience ainsi que les différents algorithmes utilisés.

2.1 Echantillonage

Pour acquérir nos données, nous avons fait passer 18 sujets pour un entrainement en coup droit (en collaboration avec le travail de Master de Lionel Lugon). Les sujets étaient tous des hommes de 18 à 35 ans, droitiers et n'ayant jamais pris de cours de tennis. Ils ont suivi 4 entrainements répartis sur 10 jours puis une rétention sept jours plus tard dans le cadre de l'apprentissage du mouvement. Les sujets ont tous donné leur consentement écrit. Au début de chaque entrainement, nous avons enregistré une série de 10 coups droits. Afin de réguler la distribution des balles, celles-ci partent d'un tube en polypropylène de 100 cm de long et 12,5 cm de diamètre. Le tube est fixé au plafond à 2.56 m de hauteur avec une légère pente de 8.96%. Les sujets doivent renvoyer la balle en coup droit, après que celle-ci ait fait un rebond. La position initiale du joueur est définie ainsi : debout, pieds à la largeur des hanches. Le sujet doit commencer face à l'écran à +/- 1m de la zone de rebond. La balle arrivant sur son coté droit, il doit faire un quart de tour pour préparer la frappe.

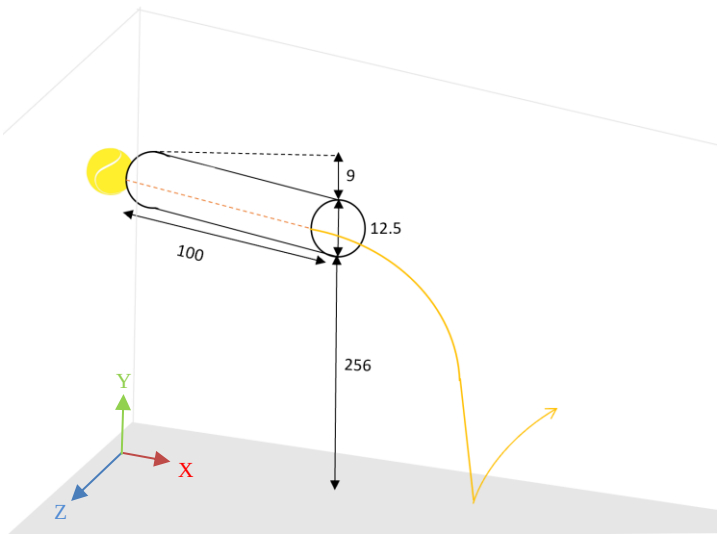


Figure 2. Schéma de l'installation pour la distribution des balles. Mesures données en centimètres (cm).

Les enregistrements ont été faits dans le laboratoire CopeLab à l'université de Fribourg. La salle noire d'une dimension de 876 cm de longueur et de 605 cm de largeur est équipée de 16 caméras infrarouges PRIME 17W comme illustré dans la figure 3. Avant chaque

enregistrement, une calibration des caméras a été faite et acceptée si elle était jugée comme « exceptionnelle » par le système. Chaque sujet est équipé d'une combinaison avec 41 capteurs réfléchissants (cf. figure 4) qui permettent de capturer la trajectoire virtuelle des marqueurs en 3D.

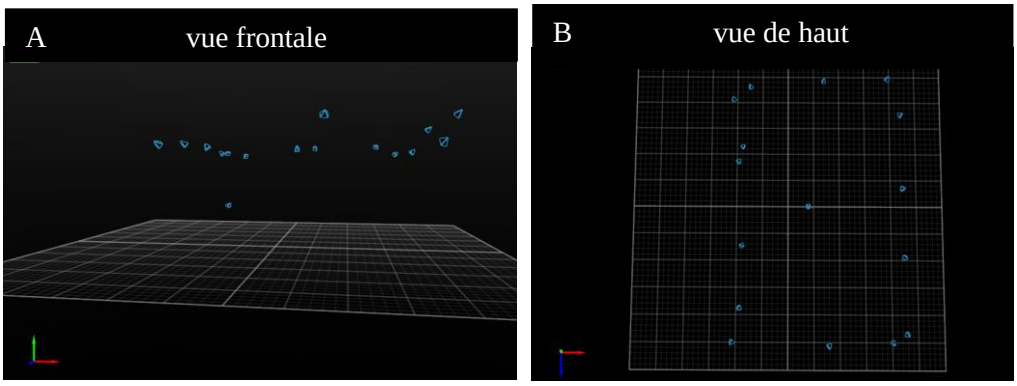


Figure 3. Positions des 12 caméras dans la salle d'expérience. A : Vision de face (direction inverse du sens de la frappe de la balle. B : Vision de haut.

La raquette (Babolat Pure Drive Wimbledon 2016) et les balles (Wilson US OPEN Extra Duty) étaient également équipées de marqueurs réfléchissants. Les squelettes guidés en 3D ont été reconstruits à l'aide du logiciel Motive 2.1.1. Les vidéos ont ensuite été séquencées pour chaque mouvement (+/-80 frames avant et après le contact de la raquette avec la balle).

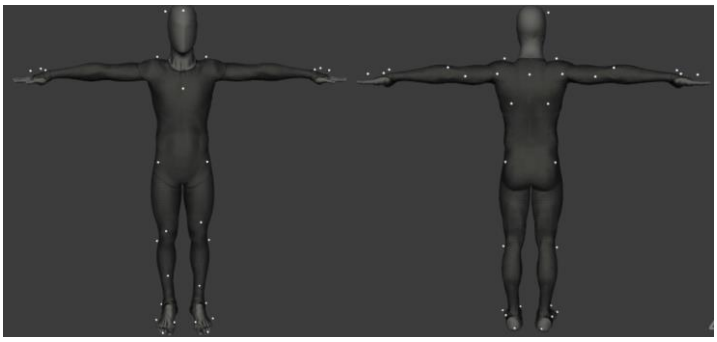


Figure 4. Disposition des 41 capteurs pour l'enregistrement 3D des mouvements par le logiciel Motive 2.1.1.

Plusieurs variables de l'action ont été récoltées. Premièrement, l'ensemble des rotations du squelette ont été capturées. Puis toutes les rotations des articulations autour des axes x, y et z ont été enregistrées avant/pendant/après le moment de la frappe dans un fichier .text. Les rotations « *avant contact* » (notation : *bImpact*) de la frappe correspondent au point mort de la raquette durant le mouvement de préparation vers l'arrière, c'est-à-dire à la dernière position avant qu'elle soit ramenée vers l'avant. Le moment d'impact (notation : *Impact)* correspond au

moment du contact entre la balle et la raquette. Les rotations « *après contact*» (notation: *aImpact)* sont la réflexion temporelle du point mort de la raquette lors de la préparation arrière du mouvement. De plus, certains angles correspondent aux feedbacks donnés lors de l'entrainement (cf. Travail de master de Lionel Lugon (2019)) . L'angle « *appui* » est l'angle formé entre l'axe des deux pieds et la projection de la raquette au sol. Ensuite, l'angle « *poignet* » est l'angle présent entre la prise de main de la raquette et l'avant bras. Enfin, l'angle « *coude* » mesure l'ouverture de l'articulation du coude.

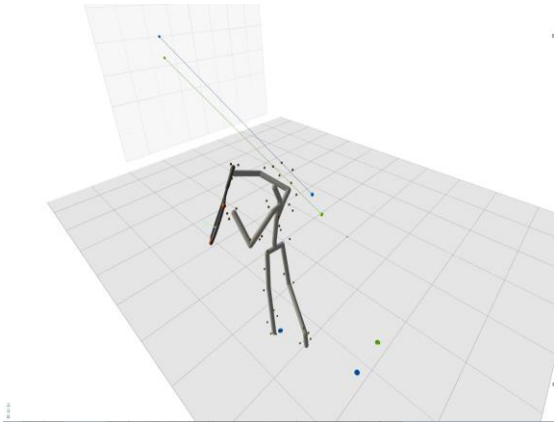


Figure 5. Représentation de la reconstitution du squelette par le logiciel Motive 2.1.1 lors d'une frappe en coup droit.

Les mouvements ont été visionnés par quatre experts de tennis (entraineurs certifiés j&S et/ou joueurs ayant au moins un niveau R4) et ont été jugés qualitativement grâce à une note octroyée par les experts allant de 0 à 9 selon le barème suivant :

- Mouvement parfait : 9
- Très bon mouvement : 7-8
- Bon mouvement : 5-6
- Mouvement satisfaisant : 3-4
- Mouvement insatisfaisant : 0-2

2.2 Implémentation des algorithmes

L'ensemble des données est composé de 559 mouvements. Afin d'avoir la plus grande diversité possible de données, 50 mouvements d'experts ont été ajoutés à la base de données d'observations des sujets. Les données acquises ont été réparties en deux groupes. Le premier consiste en des données d'apprentissage (ou données d'entrainement) pour la phase d'apprentissage de notre modèle. Le deuxième contient les données de test qui seront utilisées pour déterminer la qualité de l'algorithme. La répartition usuelle citée dans l'ouvrage de

Lemberger et al. (2016) et dans celui de Géron (2017) est de 80%-20%. Dans notre cas, cela représente 419 données pour le jeu d'apprentissage et 140 données de test. La répartition entre les deux ensembles a été faite de manière aléatoire, tout en s'assurant que chaque classe était représentée au moins une fois dans chaque ensemble (dénombrage avec la fonction sum. La même répartition aléatoire a été gardée dans les différents algorithmes. Afin de pouvoir traiter nos données, nous avons fait la moyenne des notes données par les experts pour chaque mouvement (data.moyenne). Dans le cas de l'apprentissage par classification, nous avons arrondis les moyennes au nombre entier le plus proche (data.arrondi). Pour la prédiction des données, nous avons utilisé le logiciel Python 3.7 et plus précisément la libraire Scikit-Learn, importante pour tout ce qui concerne le ML.

Nous ne détaillerons pas toutes les implémentations. Toutefois, la liste des scripts est disponible en annexe.

Tous les algorithmes sont créés sur un schéma similaire :

- Importation du fichier .csv des données d'observations avec panda.
- Définition des données d'entrée X comme étant la matrice des données d'angles des segments et articulations du corps.
- Définition des données de sortie Y comme étant le vecteur des moyennes des notes attribuées par les experts (moyenne arrondies ou non selon le mode de traitements des données).
- Répartition des données en un groupe test et un groupe d'apprentissage (faite à la main avec 20% de données de test et 80% données d'apprentissage)
- Prédiction de y grâce un l'algorithme d'apprentissage : .predict(X_test)
- Comparaison entre les données prédites/observées et calcul de l'erreur de la prédiction.

2.2.1 Algorithmes de classification. Nous avons choisi deux algorithmes pour la classification multiclasse. Pour ces traitements, nous avons utilisé le vecteur des moyennes arrondies.

1) KNeighborsClassifier : classification établie selon le vote des k-voisins les plus proches. Le paramètre le plus important est le choix du nombre k de voisins. Selon l'ouvrage de Singh (2018), nous avons basé notre choix selon la valeur RMSE. Cette erreur a été calculé pour $k \in [0, 30]$ avec une boucle for, puis nous avons sorti la valeur de k pour laquelle la RMSE était minimale. Nous avons également créé un graphique représentant les valeurs de la RMSE en fonction de k, dans le but d'avoir une idée de la tendance de

cette erreur. La métrique utilisée est celle de la distance euclidienne (metric='euclidean').

2) DecisionTreeClassifier : classification basée sur un arbre de décision. La profondeur maximale de l'arbre n'a pas été définie (max_depth = None) de manière à ce que les nœuds soient développés jusqu'à ce que les feuilles soient pures. La stratégie utilisée pour choisir la séparation de chaque nœud est définie comme étant la meilleure (splitter="best"). Enfin, par défaut, la fonction de mesure de qualité d'une séparation aux nœuds est choisie par la mesure d'impureté Gini :

$$G_i = 1 - \sum_{k=1}^{n} {p_{i,k}}^2 \quad (6)$$

avec $p_{i,k}$, le pourcentage d'observation pour la classe k parmi toutes les observations d'entrainement dans le $i^{ème}$ nœud.

Il est également possible d'utiliser la valeur d'entropie (criterion = "entropy") définie comme suit pour un $i^{ème}$ nœud :

$$H_i = - \sum_{\substack{k=1 \\ p_{i,k} \neq 0}}^{n} p_{i,k} \log(p_i, k)$$

Ces deux valeurs sont nulles lorsque les observations appartiennent à une seule classe; le nœud est pur. Lorsque nous avons le cas inverse, elles prennent la valeur 1 (Pedregosa et al., 2011). La question du choix de la mesure entre l'impureté Gini et l'entropie est traitée dans l'ouvrage de Géron (2017). Sur les résultats finaux, les arbres semblent similaires et l'impureté Gini est plus rapide à calculer (Raschka, 2019). C'est pourquoi nous avons fait le choix de garder cette mesure. Nous avons également travaillé sur la représentation graphique de l'arbre de décision avec graphviz. L'importance des différentes composantes du mouvement lors des choix à chaque nœud a été calculée grâce à feature_importances. Afin d'avoir un aspect plus visuel, nous avons créé un tableau en barre horizontale grâce plot.barh.

Selon Géron (2017), il était intéressant d'examiner la matrice de confusion (confusion_matrix()) pour analyser l'erreur d'un algorithme de classification. Cette matrice de confusion pour un classificateur multiclasse analyse le nombre d'éléments bien classés ; les lignes représentent les vraies classes et les colonnes les classes prédites. La représentation graphique de cette matrice est déterminée grâce à des niveaux de bleus (cmap=plt.cm.Blues). Les éléments de la diagonal représentent le nombre de points pour lesquels la classe prédite est

égale à la vraie classe. Plus les valeurs y sont élevées, plus les nuances de bleus sont foncées et donc plus les prédictions sont correctes. Cependant, les résultats d'une classe sont influencés par le nombre initial d'élément la représentant. Ainsi, il est préférable d'utiliser une version qui prend en compte le nombre d'élément par classes. Pour cela, il suffit de diviser les coefficients de la matrice par le nombre d'éléments de la classe correspondante (Géron, 2017). Nous avons ainsi pour chacun de ces deux algorithmes programmé la matrice de confusion qu'il sera intéressant d'analyser plus tard.

2.2.2 Algorithmes de régression. En ce qui concerne la régression, nous avons utilisé plusieurs méthodes.

1) neighbors.KNeighborsRegressor : régression basée sur la méthode des arbres de décision. Les paramètres restent les mêmes que pour la classification.
2) linearRegression : modèle de régression linéaire. Ce modèle s'apprête également à la régression polynomiale. Pour ceci, une première étape de transformation des données d'apprentissage grâce à PolynomialFeature est nécessaire afin de pouvoir ajuster le modèle de régression linéaire. Cette fonction génère une nouvelle matrice de nos données caractéristiques constituées de toutes les combinaisons polynomiales des données. Dans le cas d'un degree=2, un tableau contenant à la base 207 variables devient un nouveau tableau avec $\frac{(n+2)!}{n!2!} = \frac{(207+2)!}{207!2!} = 21736$ variables (formule tirée de Géron, 2017).

Pour analyser la précision de nos différents modèles, nous avons calculer 2 mesures. La première est l'erreur quadratique moyenne donnée par la fonction mean_squared_error. La seconde, qui concerne les algorithmes de classification, est la fonction accuracy_score définie comme suit :

$$accuracy(y_{test}, y_{prédit}) = \frac{1}{n}\sum_{i=1}^{n-1} \mathbb{I}(y_{test} = y_{prédit}), \quad (7)$$

où $\mathbb{I}(x)$ est la fonction indicatrice et n est le nombre de données. Le score renvoyé est compris dans [0,1] et calcule la précision de notre modèle en analysant le nombre de prédictions correctes (Pedregosa et al., 2011).

Pour les algorithmes de régression, nous avons défini la fonction .score. Nous avons ainsi obtenu le score R^2 ce qui nous permet d'avoir une idée de la précision moyenne sur les données de test et les données prédites.

2.2.3 Méthode de boosting. Afin de tester une méthode de boosting pour notre cas de régression linéaire avec l'algorithme d'arbres aléatoires, nous avons travaillé avec

GradientBoostingRegressor qui est un algorithme de boosting du gradient. Afin d'être au plus précis dans notre prédiction, nous avons évalué les erreurs quadratiques moyennes afin de connaitre le nombre d'arbres à implémenter dans notre algorithme grâce à staged_predict. Le boosting du gradient est assez robuste pour résister au surajustement. Ainsi un grand nombre d'arbres amènent la plupart du temps à de meilleures performances (Pedregosa et al, 2011). La fonction de perte à minimiser est la fonction de régression des moindres carrées (cf. formule (1)). Nous nous sommes restreint à des arbres simples ainsi la profondeur maximale d'un arbre est de deux avec un nombre d'arbres ajouté de 500 (n_estimators=500, max_depth=2). Müller et Guido (2018) assistent sur un deuxième paramètre important, le degré d'intensité ; degré « avec lequel chaque arbre essaie de corriger les erreurs des arbres précédents » (p.93). Afin d'éviter un modèle trop complexe, le degré a été fixé à 0.1 (learning_rate=0.1), « valeur faible (…), mais qui permet généralement aux prédictions de mieux se généraliser » (Müller & Guido, 2018, p.92).

2.2.4 Deep learning. Pour la dernière partie d'apprentissage avec du deep learning, nous avons utilisé MLPRegressor qui est un algorithme de régression utilisant le MLP. Aussi nommé réseau de neurones feed-forward, il s'agit là d'une généralisation des modèles linéaires, qui pour conclure à une prédiction, opère en de nombreuses étapes de traitement (Müller & Guido, 2018). Lemberger et al. (2016) soulignent le fait que les fonctions complexes f ont besoin d'un nombre important de neurones pour parvenir à approximer les valeurs correctement. Nous avons donc paramétré 500 couches cachées (grâce à hidden_layer_sizes). La méthode utilisée pour trouver la solution pour l'optimisation du poids est l'algorithme L-BFGS (Limited memory for the Broyden-Fletcher-Goldfarb-Shanno algorithm, solver='lbfgs'). C'est une approximation de l'algorithme BFGS de la famille des méthodes quasi-Newton qui utilise le volume limité de la mémoire de l'ordinateur. Cette méthode convergeant rapidement, elle est plus performante pour des échantillons de données contenant moins de 1000 observations (Pedregosa et al., 2011). Présentée dans l'ouvrage de Müllet et Guido (2018), nous avons choisi comme fonction d'activation non-linéaire pour les couches cachées, une unité de rectification linéaire (activation='relu'), qui renvoie $f(x) = max\ (0, x)$.

3 Résultats

Notons que tous les tests ont été réalisés directement sur le logiciel Python 3.7. Afin d'éviter des erreurs de propagation ou des assignations faussées, chaque algorithme a été programmé dans une fenêtre différente. Tous les scripts des algorithmes sont présentés dans l'annexe 1.

Notre premier algorithme de gestion des données nous a permis de nous assurer que la répartition aléatoire des données respectait une certaine rigueur au niveau de la représentation de chaque classe. Les résultats de la répartition des classes sont présentés dans le *tableau 1*.

Tableau 1
Répartition des données

	Données de test		Données d'apprentissage	
Note attribuée	Nombre de représentants	Pourcentage (%)	Nombre de représentants	Pourcentage (%)
1	1	0.89	2	0.45
2	5	4.46	18	4.03
3	20	17.85	83	18.57
4	23	20.54	105	23.49
5	23	20.54	81	18.12
6	23	20.54	80	17.89
7	7	6.25	30	6.71
8	2	1.79	6	1.34
9	8	7.14	42	9.40
Total	112	100	447	100
% Total*	20.04%		79.96%	

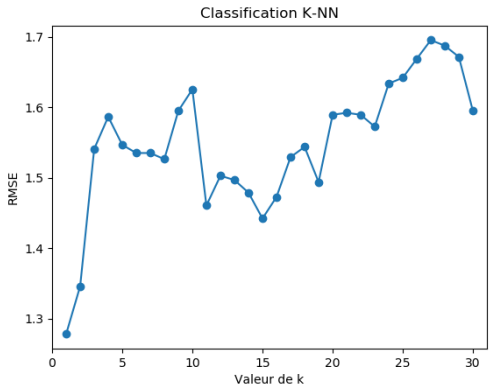
Note. Pour les notes attribuées, nous avons pris en considération les moyennes arrondies des notes des experts.
* Pourcentage du nombre totale des 559 données enregistrées.

3.1 KNeighborsClassifier

Avec cet algorithme, nous avons pu faire de la classification ainsi que de la régression. Pour ces deux méthodes, nous avons en premier recherché le nombre de voisins k qui permet d'avoir une RMSE le plus petit possible (formule (3)). Dans le cas de la classification K-NN, la boucle for nous permet de déterminer que la valeur de k, qui permet une RMSE minimale, est égale à 1. La valeur de la RMSE pour $k = 1$ est de 1.2782 comme indiqué dans le *tableau 2.*

Tableau 2

Classification K-NN : recherche de la valeur k

Représentation graphique des valeurs de la RMSE en fonction de la valeur de k	
Valeur RMSE minimale	1.27825215486952
Valeur de k	1

Note. Pour la méthode de classification, nous avons utilisé le vecteur des moyennes arrondies pour les données d'entrainement et les données de test. RMSE : racine de l'erreur quadratique moyenne. La formule de la RMSE se trouve dans l'équation (3).

La recherche de la valeur du *k* nous a permis de créer un algorithme K-NN plus précis. Nous avons obtenu une matrice de confusion (M1) donc les coefficients apparaissent dans la figure 6. Pour rappel, cette matrice compte le nombre de fois où les observations de la classe X ont été rangés dans la classe Y. Les colonnes de cette matrice sont les classes prédites alors que les lignes représentent les classes réelles. Les nombres présents sur la diagonale correspondent donc à toutes les données qui ont été correctement classées. Les nuances de bleus s'assombrissent en fonction du nombre d'éléments dans cette classe. Plus il y a d'éléments présents, plus la case correspondante est foncée.

A

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 2 & 9 & 6 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & 5 & 13 & 3 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 6 & 8 & 1 & 0 & 0 \\ 0 & 0 & 2 & 0 & 8 & 8 & 4 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

B

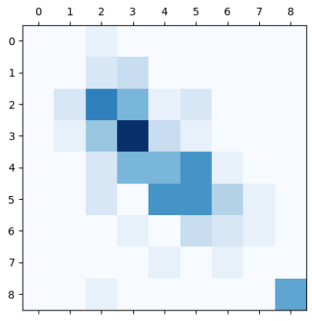


Figure 6. Matrice de confusion M1 pour l'algorithme de classification K-NN. A : coefficients de la matrice de confusion M1. La dimension de la matrice correspond au nombre de notes (classes) possibles lors de la classification. B : Représentation visuelle de la matrice M1. Les nuances de foncés sont proportionnelles au nombre de réprésentants par classe. Nombre total de données testées : 112.

La figure 7 présente les données de la matrice de confusion pondérée (M2). Les dimensions de la matrice M1 sont conservées mais chaque coefficient a_{ij} , $i,j \in [1,9] \subset \mathbb{N}$ de M1 est divisé par le nombre n_i de représentants de sa classe ce qui correspond au nombre d'élément sur la ligne i. Les coefficient b_{ij} de M2 sont définis ainsi :

$$b_{ij} = \frac{a_{ij}}{\sum_j a_{ij}} , \qquad i,j \in [1,9] \subset \mathbb{N} \tag{8}$$

avec n_i , le nombre de représentant pour chaque classe.

A

0	0	1	0	0	0	0	0	0
0	0	0.4	0.6	0	0	0	0	0
0	0.1	0.45	0.3	0.05	0.1	0	0	0
0	0.043	0.217	0.565	0.131	0.043	0	0	0
0	0	0.087	0.261	0.261	0.348	0.043	0	0
0	0	0.087	0	0.348	0.348	0.174	0.043	0
0	0	0	0.143	0	0.429	0.286	0.143	0
0	0	0	0	0.5	0	0.5	0	0
0	0	0.125	0	0	0	0	0	0.875

B

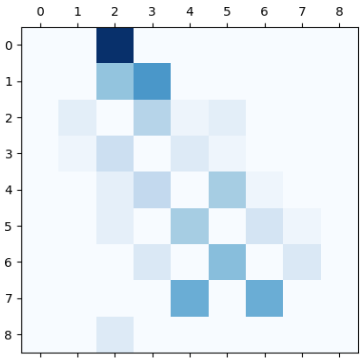


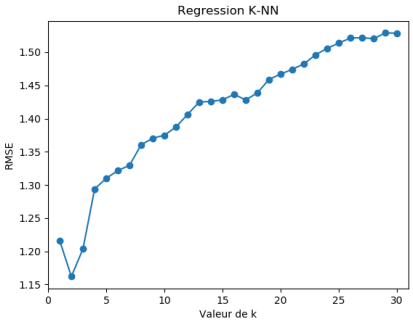
Figure 7. Matrice de confusion M2 pour l'algorithme de classification K-NN. A : coefficients de la matrice de confusion M1. La dimension de la matrice correspond au nombre de notes (classes) possibles lors de la classification. Les données des coefficients ont été arrondies au centième. Par soucis de clarté, les coefficients de la diagonale ont été mis à zéro de manière à avoir une meilleure représentation des erreurs. B : Représentation visuelle de la matrice M1. Les nuances de foncés sont proportionnelles au nombre de réprésentants par classe. Nombre total de données testées: 112.

3.2 KNeighborsRegressor

L'algorithme K-NN a également été utilisé pour la régression. Tout comme pour la classification, nous avons en premier lieu recherché la valeur de *k* puis nous avons appliqué l'algorithme à notre jeu de données de test. Soulignons que pour la régression, nous avons utilisé le vecteur des moyennes des notes non-arrondies. Le coefficient de détermination R^2 a également été calculé et vaut 0.5821 dans notre cas (calculé grâce à la formule (7)). Les résultats sont présentés ci-dessous dans le *tableau 3.*

Tableau 3

Classification K-NN : recherche de la valeur k

Représentation graphique des valeurs de la RMSE en fonction de la valeur de k	
Valeur RMSE minimale	1.1619070108170078
Valeur de k	2
Coefficient R^2	0.5820792389213734

Note. Pour la méthode de régression, nous avons utilisé le vecteur des moyennes non-arrondies pour les données d'entrainement et les données test. K-NN: k-plus proche voisin. RMSE : racine de l'erreur quadratique moyenne. La formule de la RMSE se trouve dans l'équation (3). Le coefficient R^2 est calculé à partir de la formule (5)**(5)**.

3.3 DecisionTreeClassifier

Pour cet algorithme basé sur les arbres de décision, nous avons remarqué que les données pouvaient varier légèrement, ainsi nous avons fixé random_state=1 afin d'obtenir des résultats constants. Le score de précision est de 0.3928. L'erreur quadratique moyenne vaut pour cet algorithme 1.2356. Nous avons également représenté les matrices de confusion (cf. figure 8 et 9).

A

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 2 & 11 & 4 & 3 & 0 & 0 & 0 & 0 \\ 0 & 2 & 5 & 6 & 7 & 3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 10 & 5 & 1 & 0 & 0 \\ 0 & 0 & 1 & 3 & 9 & 5 & 5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 7 \end{bmatrix}$$

B

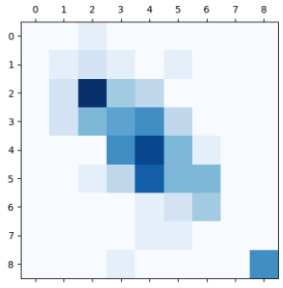


Figure 8. Matrice de confusion M1 pour l'algorithme DecisionTree. A : coefficients de la matrice de confusion M1. La dimension de la matrice correspond au nombre de notes (classes) possibles lors de la classification. B : Représentation visuelle de la matrice M1. Les nuances de foncé sont proportionnelles au nombre de réprésentants par classe. Nombre total de données testées : 112.

A

$$\begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0.4 & 0.2 & 0 & 0.2 & 0 & 0 & 0 \\ 0 & 0.1 & 0.55 & 0.42 & 0.15 & 0 & 0 & 0 & 0 \\ 0 & 0.087 & 0.217 & 0.261 & 0.304 & 0.130 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.304 & 0.435 & 0.217 & 0.043 & 0 & 0 \\ 0 & 0 & 0.043 & 0.130 & 0.391 & 0.217 & 0.217 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.143 & 0.286 & 0.571 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.125 & 0 & 0 & 0 & 0 & 0.875 \end{bmatrix}$$

B

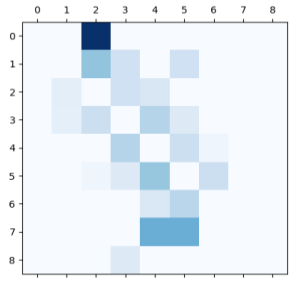


Figure 9. Matrice de confusion M2 pour l'algorithme de classification DecisionTree. A : coefficients de la matrice de confusion M1. La dimension de la matrice correspond au nombre de notes (classes) possibles lors de la classification. Les coefficients (calculés grâce à la formule (8)) ont été arrondis au centième. Par soucis de clarté, les coefficients de la diagonale ont été mis à zéro de manière à avoir une meilleure représentation des erreurs. B : Représentation visuelle de la matrice M1. Les nuances de foncés sont proportionnelles au nombre de réprésentants par classe. Nombre totale de données testées = 112.

Grâce à cet algorithme, nous avons fait une extraction de l'importance des caractéristiques. Les 5 caractéristiques les plus importantes sont présentées dans le *tableau 4.*

Tableau 4

Classement importances des caractéristiques DecisionTreeClassifier

		Taux d'importance	%
1.	aImpact_RightUpLeg_x	0.1067026200260916	10.67
2.	aImpact_appuis_angle	0.0659600983347930	6.6
3.	impact_Neck_z	0.0501212602976818	5.01
4.	impact_LeftToeBase_x	0.0465092520774130	4.65
5.	bImpact_leftShoulder_z	0.0355294214735607	3.55

Nous avons également ressorti le classement des caractéristiques rattachées aux angles des articulations ainsi que celles en rapport avec la raquette (cf. annexe 3: *tableau 9*). L'arbre de décision pour cette méthode est disponible dans l'annexe 2.

3.4 DecisionTreeRegression

Tout comme pour la méthode de classification à partir d'un algorithme d'arbre de décision, cet algorithme de régression nous a permis de souligner les cinq caractéristiques les plus importantes au niveau de l'apprentissage. Elles sont répertoriées dans le *tableau 5.*

Tableau 5

Classement importances des caractéristiques DecisionTreeRegression

		Taux d'importance	%
1.	aImpact_RightUpLeg_x	0.4332601948165229	43.32
2.	aImpact_appuis_angle	0.1549809613694461	15.49
3.	aImpact_Spine_y	0.0490235335527040	4.90
4.	impact_LeftToeBase_x	0.0415046693737312	4.15
5.	aImpact_RightLeg_z	0.0399152360208649	3.99

Grâce à la formule (5), nous avons calculé pour cet algorithme de régression un score R^2 de 0.52908. Son RMSE est de 1.2333. Dans le *tableau 10* de l'annexe 3, nous exposons les différentes importances de caractéristique comme déjà présentées pour la méthode de classification.

3.5 GradientBoostingRegressor

Pour cette méthode de boosting, nous avons en premier cherché le nombre d'arbre optimal. Après itérations, ce nombre s'élève à 465 (nombre maximum possible = 500) . Le coefficient de détermination R^2 vaut 0.78362 et la RMSE est de 0.8360. De plus, nous avons obtenu le classement des caractéristiques selon leur importance. Le *tableau 6* représente les cinq premières places de ce classement. Les autre pourcentages attribués aux caractéristiques d'angles et de raquette sont présentées dans le *tableau 11* en annexe.

Tableau 6

Classement importances des caractéristiques

		Taux d'importance	%
1.	aImpact_RightUpLeg_x	0.32483578703413396	32.48
2.	aImpact_appuis_angle	0.11725251267652063	11.73
3.	aImpact_Spine_y	0.10074180767773083	10.07
4.	aImpact_RightLeg_z	0.06189133925994384	6.19
5.	impact_RightHand_x	0.02459413477931358	2.46

Nous avons également décidé d'ajouter un algorithme pour déterminer les coefficients de corrélation entre les notes et certaines caractéristiques qui sont ressorties comme étant les plus importantes par GradientBoostingRegressor grâce à np.corrcoef. Nous obtenons les coefficients de corrélation r qui sont présentés dans le *tableau 7*.

Tableau 7

Coefficient de corrélation entre les notes attribuées et des caractéristiques

Caractéristiques choisies	r
aImpact_RightUpLeg_x	- 0.528815
aImpact_appuis_angle	0.4184342
aImpact_Spine_y	-0.5033415
aImpact_RightLeg_z	-0.3737418
impact_RightHand_x	-0.0889955

Note. r = coefficient de corrélation, $|r| \in [0,1]$. Le signe montre le sens de la corrélation. La corrélation est calculée pour la relation entre la caractéristique et la moyenne des notes non-arrondies correspondantes.

3.6 Régression et DeepLearning

Dans le cas de notre algorithme de régression linéaire LinearRegression, nous avons obtenu un score R^2 de 0.5125. La RMSE vaut 1.25487. La première étape de la régression polynomiale nous a permis de passer d'une matrice de dimension 447 x 207 à une matrice 447 x 21735 (grâce à PolynomialFeatures). La régression polynomiale a un coefficient de détermination de 0.6458 ainsi qu'une RMSE de 1.0697. Enfin, la méthode MLP obtient un score d'exactitude de 0.6896 et une RMSE de 1.0013.

Finalement, afin d'avoir une vision d'ensemble de tous nos algorithmes, le *tableau 8* récapitule les scores d'exactitude, la RMSE et l'erreur moyenne absolue (MAE) de toutes nos méthodes proposées.

Tableau 8

Récapitulatif des résultats pour les algorithmes utilisés

Classification

	Score d'exactitude	RMSE	MAE
KNeighborsClassifier	0.4017	1.2782	0.8482
DecisionTreeClassifier	0.3928	1.2356	0.8482

Régression

	Coefficient R^2	RMSE	MAE
KNeighborsRegressor	0.5821	1.1619	0.8203
DecisionTreeRegressor	0.5290	1.2333	0.8437
GradientBoostingRegressor	0.7836	0.8360	0.6053
LinearRegression	0.5125	1.2540	0.8493
RegressionPolynomial	0.6457	1.0697	0.8314
MLPRegressor	0.68961	1.0013	0.7759

Note. Score d'exactitude est calculé grâce à la fonction score (formule (7)) . Le coefficient R^2 grâce à la fonction r2_score. Notation : RMSE = racine de l'erreur quadratique moyenne, MAE = erreur moyenne absolue entre la prédiction et la valeur exacte a également été calculée grâce à la formule mean_absolute_error.

3.7 Représentations de squelettes

Afin de pouvoir optimiser notre discussion, nous avons fait des captures de mouvements en mettant différents aspects en avant dans les figures présentées ci-dessous.

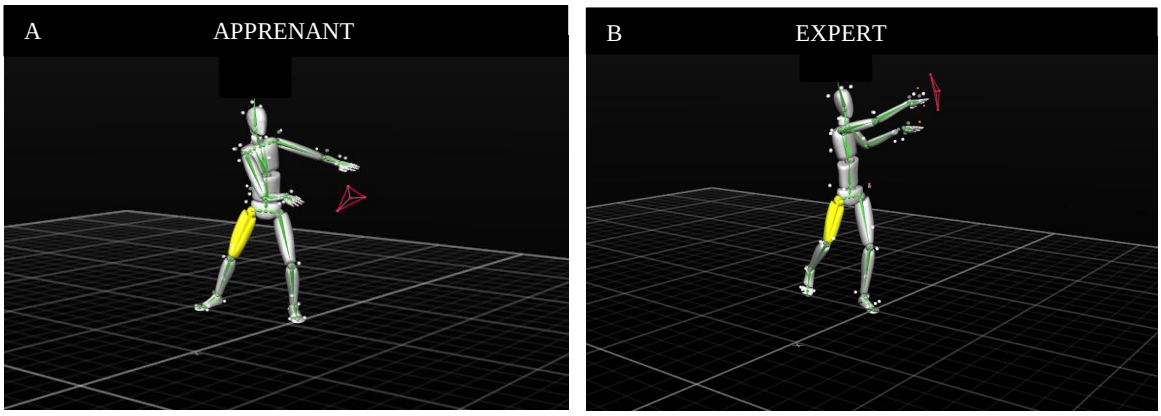


Figure 10. Comparaison au niveau de l'angle de la hanche (côté droit) après l'impact de la balle. A : Représentation du mouvement d'un apprenant ayant reçu une note de 2.75. B : Représentation du mouvement d'un expert ayant reçu une note de 9.

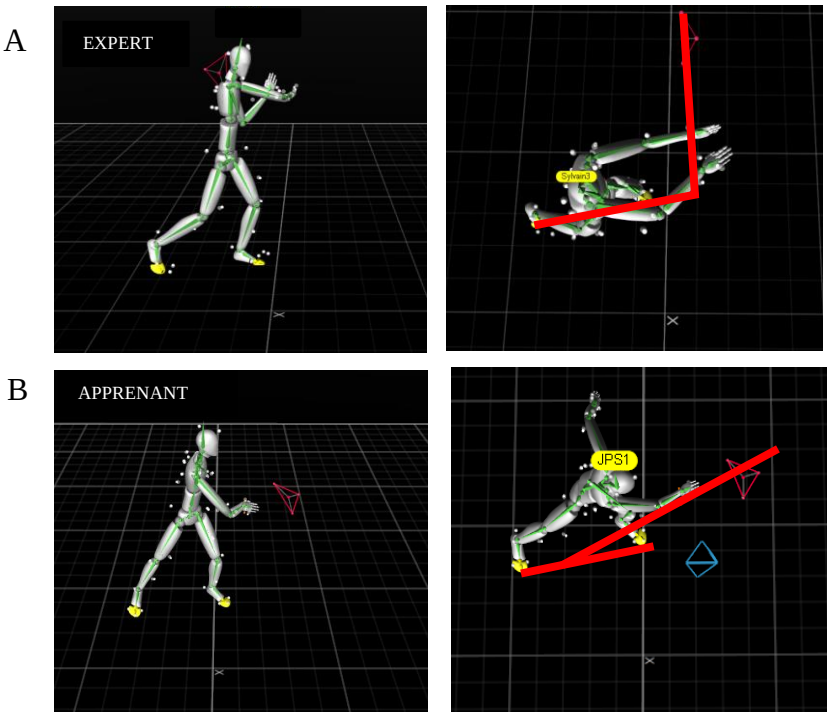


Figure 11. Représentation de l'axe « appuis » depuis une vue sagitale et une vue du haut. A : Représentation du mouvement d'un apprenant ayant reçu une note de 2.75. B : Représentation du mouvement d'un expert ayant reçu une note de 9.

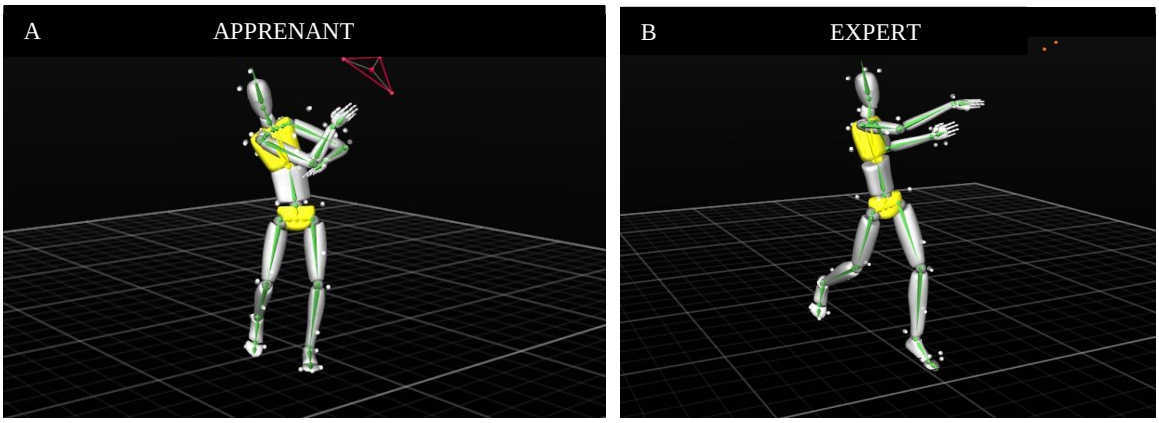


Figure 12. Représentation de la toursion du torse après l'impact avec la balle. A : Représentation du mouvement d'un apprenant ayant reçu une note de 2.5. B : Représentation du mouvement d'un expert ayant reçu une note de 9.

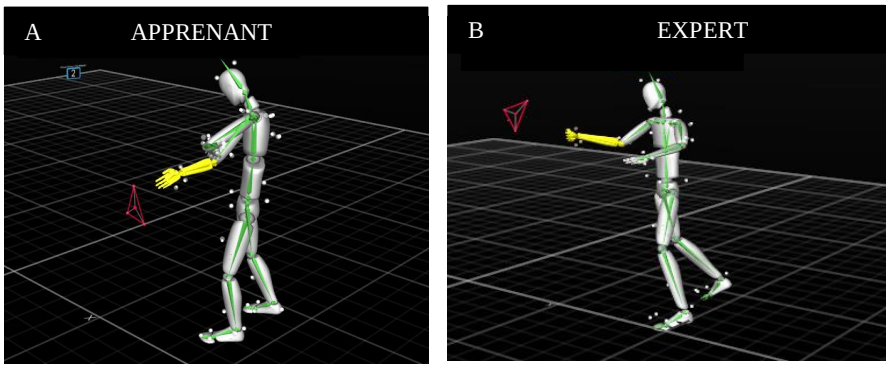


Figure 13. Représentation de la position de la main droite au moment de la frappe. A : Représentation du mouvement d'un apprenant ayant reçu une note de 2.25. B : Représentation du mouvement d'un expert ayant reçu une note de 9.

4 Discussion

Dans ce chapitre, nous allons discuter des différents résultats obtenus grâce à nos méthodes. Nous prendrons comme support les deux questions que nous nous étions posées afin d'analyser les forces et faiblesses des algorithmes et des méthodes proposées.

4.1 Est-ce que l'algorithme d'apprentissage et le jugement humain suivent une même logique ?

Afin de répondre à cette question, nous allons en premier lieu analyser nos algorithmes de classification. Les algorithmes de classification présentés dans ce travail sont KNeighborsClassifier et DecisionTreeClassifier. Les fonctions suivantes ont été évaluées par la fonction .score (cf. formule (7)) qui calcule la précision moyenne des données. Nous voyons que les résultats de ces deux algorithmes sont très semblables (score d'exactitude : 0.4017 et 0.3928). Les prédictions de l'algorithme K-NN sont dépendantes du nombre k de voisins qu'on lui implémente. Dans notre cas, nous avons choisi la valeur de k qui permet une RMSE minimale mais le résultat obtenu $k = 1$ semble petit, bien que dans la littérature de Müller et Guido (2018), il est proposé de s'arrêter à un nombre petit de voisins (entre 3 et 5). Avec autant de caractéristiques (n=207), nos données d'entrainement sont très dispersées et donc pour garder une précision convenable, l'algorithme choisi de prendre en considération le voisin le plus proche.

Sachant que nos notes vont de 0 à 9, l'erreur moyenne absolue de 0.842 correspond donc à placer une prédiction dans une classe inférieure ou supérieure à celle réelle. La matrice de confusion de l'algorithme K-NN (cf. figure 6) illustre bien cette tendance puisque la plupart des résultats se trouvent proches de la diagonale. Si l'on prend le nombre total de données testées (n=112), 45 données sont présentes sur la diagonale, ce qui équivaut à 40.17% de réussite. Enfin, si l'on analyse le nombre de données qui ont été sous/sur-évaluées d'une classe, nous obtenons 48 prédictions. 42.85% des données se trouvent dans la tendance. 10.96% des données ont été prédites avec une erreur de plus d'un point à la note, ce qui est conséquent pour un algorithme. L'erreur vient sûrement du fait que le nombre de représentants pour chaque classe n'était pas égal. Pour une prédiction plus précise, nous aurions aimé avoir une répartition homogène de nos 112 données dans nos 10 classes. Seulement, comme nous pouvons remarquer dans le *tableau 1*, les représentants pour les classes 1 et 8 sont très peu nombreux. Plus de 75% de nos données d'apprentissage sont groupées dans les notes 3, 4, 5 et 6, ce qui amène à un surapprentissage pour ces groupes et/ou un sousapprentissage pour les autres. La matrice de la

figure 7 permet de constater que les taux de réussite les plus hauts sont ceux pour les notes 3 (45%), 4 (56%) et 9 (87%). Ces classes, qui présentent certainement le plus d'homogénéité dans les mesures, ont pu être le plus correctement définies par l'algorithme. Pour l'algorithme K-NN, les classes plus communes ont tendance à dominer la prédiction pour un nouvel élément, car elles sont statistiquement plus fréquentes parmi les k plus proches voisins. Il aurait fallu un découpage plus homogènes des mouvements par classes. Nous constatons quand même une anomalie dans la matrice de confusion de la figure 6. Un mouvement ayant obtenu une note de 9 par les experts a été prédit comme un mouvement noté à 3. La raison peut venir d'un manque de cohérence et/ou de rigueur dans la notation des experts. Une autre possibilité serait une anomalie de l'algorithme sachant que l'algorithme K-NN ne fonctionne pas très bien pour les jeux de données clairsemées ou avec des caractéristiques trop nombreuses (Müller & Guido, 2018) .

Si l'on compare ces résultats avec ceux de la classification DecisionTree (cf. figure 8), nous remarquons que le pourcentage de valeurs présentes sur la diagonale (39.28%) reste très proche de celui du K-NN, de même pour la dispersion +/-1 autour de la diagonale (42.85%). L'anomalie persiste à nouveau avec une note prédite de 4 mais notée par les experts à 9. Nous pouvons donc exclure le problème de l'algorithme. Toutefois, nous remarquons une tendance pour cet algorithme à sous-estimer les notes prédites avec 36 notes sous-estimées sur 112 contre 32 surestimées. Le score d'exactitude pour ces deux méthodes de classification reste relativement bas avec 0.4017 pour K-NN et 0.3928 pour les arbres de décision.

Les méthodes de régression nous offrent des scores plus élevés au niveau de l'exactitude. La meilleure méthode est la GradientBoostingRegressor qui possède un coefficient R^2 de 0.7828. Ce modèle de régression explique donc bien les données puisque le coefficient de détermination est relativement haut. C'est également le seul algorithme qui permette d'obtenir une RMSE plus petite que 1 et une erreur moyenne absolue de 0.605. Le travail de Di Cellio Dias et al. (2018) souligne également les performances d'un tel algorithme dans le cas binaire et argumente son efficacité pour toutes autres applications. Il est l'algorithme le plus performant sur notre étude et celui a recommander pour un travail futur.

Ainsi, nous pouvons conclure que pour améliorer nos algorithmes, il faudrait une répartition plus homogène des mouvements dans les classes et plus de rigueur de la part des experts dans la notation. Une échelle précisant les différents aspects du mouvement serait intéressante pour diriger la notation des experts. Nous privilégerions les algorithmes de régression et en particulier celui du GradientBoosting qui nous donne des résultats excellents.

4.2 Quels sont les critères qui amènent à ce qu'un coup droit au tennis soit jugé comme bon par un expert ?

Afin de répondre à cette question, nous nous sommes basés sur l'importance des caractéristiques ressorties grâce à la fonction feature_importances. Nous avons défini cette fonction sur trois de nos algorithmes : DecisionTreeClassifier, DecisionTreeRegressor et GradientBoostingRegressor. Les résultats sont plus dispercés sauf deux caractéristiques qui ressortent sur chacun des trois classements (cf. *tableau 4-5-6*). Il s'agit de « *aImapact_rightUpLeg_x* » et « *aImpact_appuis_angle* ».

La première correspond à l'angle de la jambe droite au niveau de la hanche autour d'un axe x après l'impact. La relation entre l'angle et la note est modérée avec un coefficient de corrélation de -0.5288. Cette relation décroissante signifie que lorsque l'angle augmente, la note diminue et vice-versa. Toutefois comme les axes correspondent à un repère fixe au sol et que le sujet bouge dans la salle, il n'est pas possible de tirer des conclusions par rapport à l'angle à apporter pour avoir une meilleure note. Du fait que cette caractéristique ressort avec des taux haut d'importance variant de 10.67% avec le DecisionTreeClassifier à 43.32% par le GradientBooste, nous avons décidé de poursuivre l'analyse en étudiant directement deux squelettes (cf. figure 10).

Le phase finale du mouvement dépend de la direction de la frappe et de la rotation préalable faite par le corps. Pour un appui en ligne, lorsque la frappe est exécutée, le poids du corps est en grande partie transféré sur la jambe d'appui avant. Le point important repose sur ce transfert de poids sur la jambe gauche. Dans la figure 10, nous remarquons que sur le mouvement de l'apprenant, le poids reste sur les deux jambes et la hanche est maintenue ouverte. L'expert ferme la hanche droite avec une rotation médiale de la hanche. Soulignons toutefois que le mouvement des hanches n'est que la continuïté directe du relâchement de l'épaule et du tronc après la frappe et est dépendant du style de chacun.

La deuxième caractéristique mise en avant est l'angle entre les appuis et la projection de l'axe de la raquette au sol après l'impact (aImpact_appuis_angle). Cette caractéristique a un taux d'importance d'environ 12% entre les trois algorithmes. Le coefficient de corrélation entre cet angle et la note vaut 0.4184. Il s'agit donc d'une relation positive ; plus l'angle augmente, plus la note est bonne. Si l'on prend en compte l'angle extérieur, cette théorie est plausible. Le mouvement de l'expert a un angle « appuis » beaucoup plus grand que celui du mouvement de l'apprenant. Cela est dû à une position final de coup beaucoup plus fermée sur le coté gauche avec la raquette qui est ramenée vers l'épaule gauche ainsi que des appuis presque rectillignes

Enfin, le GradientBoosterRegression donnant de très bonnes valeurs d'exactitude, nous nous sommes attardés sur deux autres caractéristiques présentes dans le *tableau 7*. La première est « *aImpact_spine_y* ». Tout comme pour la rotation des hanches, il n'y a pas d'interprétation directe à donner toutefois en se basant sur les images de la figure 12, nous pouvons présenter le résultat suivant en rapport avec la torsion du tronc. Après la frappe, la raquette peut être ramenée à différentes hauteurs entre la hanche et l'épaule. La rotation du haut du corps est importante pour que la frappe puisse être puissante. A la fin du mouvement, l'axe épaule-hanche doit rester droit avec le bassin qui revient vers l'avant. Nous remarquons sur la figure 12 que le squelette de droite subit un déséquilibre au niveau du tronc vers la droite. Le coup droit est donc optimal si le tronc reste stable, ce qui n'est pas le cas du débutant.

Finalement, la dernière caractéristique mise en valeur est « *Impact_RightHand_x* ». Il nous semblait évident de devoir parler de la position de la main droite bien que cette caractéristique est mise en avant uniquement par l'algortithme GradientBooster. Tout comme l'angle d'avant, rien ne peut être défini directement à partir de cette valeur. Ainsi, nous allons analyser le cas présenté par la figure 13. Le coefficient de corrélation entre cette caractéristique et la note attribuée est négatif et très faible (r = -0.0889). ainsi nous constatons une relation inverse. Si l'on regarde la figure 13, nous pouvons constater que la zone de frappe est différente. Chez le sujet débutant, elle se situe au niveau de la cuisse alors qu'elle devrait se trouver entre le bassin et les épaules. Le poignet doit rester dans l'axe de l'avant-bras de manière à pouvoir transmettre plus de force à la balle. Cela soutient cette relation inverse de l'angle poignet-raquette qui doit être le plus faible possible pour un coup efficace. Les algorithmes ne mettent pas en avant l'importance de l'angle du poignet ni celle du coude bien que cela paraissent être des points très importants pour le mouvement. La position de la raquette a encore moins d'importance que nos caractéristiques précédentes. Les limites des algorithmes résident dans la position fixe de l'axe de base. Il aurait fallu aligner l'axe au centre de gravité de chaque joueur pour avoir des angles plus homogènes et ainsi permettre une comparaison ciblée des angles de chaque joueur. On constate également que dans l'arbre de décision (cf. Annexe 2), tous les coefficient de Gini sont nuls sur la dernière feuille. Cela signifie que les feuilles aux noeuds sont pures. Toutefois, les résultats ont peu de sens.

En conclusion, les forêts aléatoires sont des algorithmes accessibles mais qui ne permettent pas de bonnes prédictions pour des données éparses et qui possèdent un grand nombre de caractéristiques. Dans notre cas, il est impossible de ressortir mathématiquement les aspects esssentiels d'un bon mouvement au tennis.

5 Conclusion

Dans ce travail, nous avons utilisé des algorithmes mathématiques pour créer une prédiction à partir de données d'observations. Nous avons défini des algorithmes de classification et de régression. Les résultats des prédictions pour de nouvelles données ont été satisfaisantes grâce à l'algorithme GradientBoostingRegression. Ainsi, la notation des experts suivait une certain logique reproductible par notre algorithme avec une erreur moyenne absolue de 0.603. Cependant, il n'a pas été possible de définir les caractéristiques précises qui permettaient de classer les mouvements. Bien que les algorithmes de forêts aléatoires nous ont permis de faire ressortir certaines caractéristiques importantes pour la séparation des nœuds, aucune conclusion n'a pu être directement tirée. L'analyse de squelettes nous a permis d'émettre quelques hypothèses sur l'importance de certains angles avec l'aide du coefficient de corrélation.

Les limites de notre expérience résident en deux points principaux. Le premier point est la notation subjective des mouvements par des experts. Le tennis n'étant pas un sport basé sur l'esthétisme, la technique ne sera jamais directement jugé. C'est la performance et la finalité du mouvement qui est important. Ainsi, la notation de nos données était basées principalement sur l'expérience des experts et leur acuité visuelle. Il serait intéressant de mettre en pratique ces différents algorithmes dans un sport comme la gymnastique aux agrès.

Le deuxième point vient directement de l'enregistrement de nos données corporelles pour les sujets. Il serait intéressant d'enregistrer de nouvelles données et de se focaliser sur des trajectoires par exemple. De plus, nous n'avons pas utilisé tout le panel d'algorithmes possibles ni même un semblant des possibilités offertes par le deeplearning. C'est un domaine en pleine expansion et il y a encore énormément d'utilisations possibles en relation avec le sport .

Bibliographie

Barnston, A. G. (1992). Correspondence among the Correlation, RMSE, and Heidke Forecast Verification Measures; Refinement of the Heidke Score. *Weather and Forecasting*, *7*, 699–709.

Borrel, F. (2012). *Le tennis, comment ? Approche technique, biomécanique et pédagogique*. Paris: Vigot.

Champely, S. (2004). *Statistique vraiment appliquée au sport : Cours et exercices*. Bruxelles: De boeck.

Chang, P. D., Wong, T. T. & Rasiej, M. J. (2019). Deep Learning for Detection of Complete Anterior Cruciate Ligament Tear. *Journal of Digital Imaging*. https://doi.org/10.1007/s10278-019-00193-4.

Cronin, N. J., Rantalainen, T., Ahtiainen, J. P., Hynynen, E. & Waller, B. (2019). Markerless 2D kinematic analysis of underwater running: A deep learning approach. *Journal of Biomechanics*, *87*, 75–82. https://doi.org/10.1016/j.jbiomech.2019.02.021.

Di Cellio Dias, P. C., Experian, S., Forti, M., Bradesco & Witarsa, M. (2018). A Comparison of Gradient Boosting with Logistic Regression in Practical Cases. *SAS Institute*, 1–25. Accès à l'adresse https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1857-2018.pdf.

Fond, G., Bulzacka, E., Boucekine, M., Schürhoff, F., Berna, F., Godin, O.,…Llorca, P. M. (2019). Machine learning for predicting psychotic relapse at 2 years in schizophrenia in the national FACE-SZ cohort. *Progress in Neuro-Psychopharmacology and Biological Psychiatry*, *92*, 8–18. https://doi.org/10.1016/j.pnpbp.2018.12.005.

Géron, A. (2017). *Machine learning avec Scikit-Learn : mise en oeuvre et cas concrets*. Malakoff: Dunod.

Hastie, T., Tibshirani, R. & Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction (Second Edition)*. https://doi.org/10.1007/978-0-387-84858-7.

Khelil, H. & Benyettou, A. (2010). Application du système immunitaire artificiel ordinaire et amélioré pour la reconnaissance des caractères artificiels. *Nature et Technologie*, n°*02*/janvier, 9–13.

Lemberger, P., Batty, M., Morel, M. & Raffaëlli, J. (2016). *Big Data et Machine Learning -les concepts et les outils de la data science*. Malakoff : Dunod.

Massih-Reza, A. (2015). *Apprentissage machine- De la théorie à la pratique : Concepts fondamentaux en Machine Learning*. Eyrolles.

Mcneela, D. (s.d.). The universal approximation theorem for neural networks. Accès à l'adresse http://mcneela.github.io/machine_learning/2017/03/21/Universal-Approximation-Theorem.html.

Mezyk, E. & Unold, O. (2010). Machine learning approach to model sport training. *Computers in Human Behavior*, Vol. 27, pp. 1499–1506. https://doi.org/10.1016/j.chb.2010.10.014.

Morel, M. (2018). *Modélisation de séries temporelles multidimensionnelles. Application à l ' évaluation générique et automatique du geste sportif.* (Thèse de doctorat, Université Pierre et Marie Curie - Paris VI, France). Accès à l'adresse https://tel.archives-ouvertes.fr/tel-01703849.

Müller, A. & Guido, S. (2018). *Le machine learning avec Python : La bible des data scientists*. Paris: First.

Nicholson, K. F., Richardson, R. T., Van Roden, E. A. R., Quinton, R. G., Anzilotti, K. F. & Richards, J. G. (2019). Machine learning algorithms for predicting scapular kinematics. *Medical Engineering and Physics*, *65*, 39–45. https://doi.org/10.1016/j.medengphy.2019.01.005.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., … Duchesnay, E. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, *12*. Accès à l'adresse https://scikit-learn.org/stable/index.html.

Przednowek, K., Iskra, J., Wiktorowicz, K., Krzeszowski, T. & Maszczyk, A. (2017). Planning Training Loads for The 400 M Hurdles in Three-Month Mesocycles Using Artificial Neural Networks. *Journal of Human Kinetics*, *60*(1), 175–189. https://doi.org/10.1515/hukin-2017-0101.

Raschka, S. (2019). Why are implementations of decision tree algorithms usually binary and what are the advantages of the different impurity metrics?. Accès à l'adresse https://sebastianraschka.com/faq/docs/decision-tree-binary.html#why-are-implementations-of-decision-tree-algorithms-usually-bina.

Ryguła, I. (2005). Artificial neural networks as a tool of modeling of training loads. *Proceedings of the 2005 IEEE engineering in medicine and biology 27th annual conference*, Shanghai, China (pp. 1–4).

Singh, A. (2018). A Practical Introduction to K-Nearest Neighbors Algorithm for Regression (with Python code). Accès à https://www.analyticsvidhya.com/blog/2018/08/k-nearest-neighbor-introduction-regression-python/.

Wagenaar, M., Okafor, E., Frencken, W. & Wiering, M. (2017). Using Deep Convolutional Neural Networks to Predict Goal-Scoring Opportunities in Soccer. *International Conference on Pattern Recognition Applications and Methods (ICPRAM).*

Weineck, J. (1997). *Manuel d'entraînement : 4e Edition*. (Vigot, Ed.). Paris.

Annexes

Liste des Abréviations

BFGS	Limited memory for the Broyden-Fletcher-Goldfarb-Shanno algorithm
K-NN	K -nearest neighborhood (k-plus proches voisins)
GBRT	Gradient boosted Regression Trees
MAE	Erreur moyenne absolue
ML	Machine Learning
MLP	Multi Layer Perceptron (perceptron multicouche)
RMSE	Racine carré de l'erreur des moindres carrées
RN	Réseau de neurones

Liste des caractéristiques étudiées

Notation : Timing_partieducorps_Axes

Timing par rapport au contact de la balle avec la raquette :

Avant : bImpact

Pendant : impact

Après : aImpact

Axes : x , y , z

Parties du corps :

Hips	LeftArm	
Spine	RightArm	
LeftUpLeg	RightShoulder	
LeftLeg	RightArm	
RightLeg	LeftForeArm	
RightUpLeg	RightForeArm	
LeftFoot	LeftHand	
RightFoot	RightHand	
LeftToeBase	raquette	
RightToeBase	poignet_angle	(pas d'axe)
Head	appui_angle	(pas d'axe)
Neck	coude_angle	(pas d'axe)
LeftShoulder		

Annexe 1. Code sources des fonctions principales

Bibliothèques de bases

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

Importation des données

```
data_app = pd.read_csv("data_apprentissage.csv", sep=";", header=0)
nom_colonne = data_app.columns
caracteristiques = nom_colonne[0:207]
X_train = data_app[caracteristiques]
y_train = data_app.arrondi
data_test = pd.read_csv("data_test.csv", sep=";", header=0)
X_test = data_test[caracteristiques]
y_test = data_test.arrondi
neighbors_settings = range(1, 31)
```

Recherche du nombre de voisins pour l'algorithme KNN

```
# erreur pour différent k
rmse_val = []
exactitude_app = []
exactitude_test = []
print(list(neighbors_settings))

for K in list(neighbors_settings):
    model = neighbors.KNeighborsClassifier(n_neighbors=K)
    model.fit(X_train, y_train)
    pred = model.predict(X_test)
    error = sqrt(mean_squared_error(y_test, pred))
    exactitude_app.append(model.score(X_train, y_train))
    exactitude_test.append(model.score(X_test, y_test))
    rmse_val.append(error)
    print('la racine de lerreur quadratique moyenne valeur (RMSE) pour k=', K, 'is:', error)

print('La valeur de RMSE minimal est', min(rmse_val), 'et K pour des données de moyenne arrondies vaut', rmse_val.index(min(rmse_val))+1)

# créer un graphique
plt.plot(range(1, 31), rmse_val, "o-")
plt.xlim(0, 31)
plt.ylabel("RMSE")
plt.xlabel("Valeur de k")
plt.title("Classification K-NN")
plt.show()

# comparaison exactitude de l'appprentissage et du test par apport au score
plt.plot(neighbors_settings, exactitude_app, label="exactitude de l'apprentissage")
plt.plot(neighbors_settings, exactitude_test, label="exactitude du test")
plt.ylabel ("exactitude")
plt.xlabel("nombre de voisins")
plt.legend()
plt.show()

# même chose pour les valeurs de note moyenne mais avec y= data_app.mmoyenne
y2_train = data_app.moyenne
y2_test = data_test.moyenne
```

Algorithme KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import confusion_matrix


knn = KNeighborsClassifier(n_neighbors=1, metric='euclidean').fit(X_train, y_train)

y_pred = knn.predict(X_test)

# exactitude du test
print("Exactitude du test knn:{:.2f}", metrics.accuracy_score(y_pred, y_test))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))

# matrice de confusion
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
plt.matshow(conf_mat, cmap=plt.cm.Blues)
plt.ylabel("Note réelle")
plt.title("Note prédite")
plt.show()

# matrice de confusions pondérée
rows_sum = conf_mat.sum(axis=1, keepdims=True)
norm_conf_mat = conf_mat/rows_sum

print(norm_conf_mat)
np.fill_diagonal(norm_conf_mat, 0)
plt.matshow(norm_conf_mat, cmap=plt.cm.Blues)
plt.ylabel("Note réelle")
plt.title("Note prédite")
plt.show()
```

Algorithme KNeighborsRegressor

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import r2_score

reg = KNeighborsRegressor(n_neighbors=2, metric='euclidean').fit(X_train, y_train)
y_pred = reg.predict(X_test)

# Pour voir nos données prédites
print ("nombre de données test", len(list(y_test)))
print("nombre de données d'apprentissage:", len(list(y_train)))
print(y_pred)
print(list(y_test))
# Coefficient R2
print("coefficient R2 knn regression:", r2_score(y_test, y_pred, sample_weight=None, multioutput="uniform_average"))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

Algorithme DecisionTreeClassifier

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz

clf = DecisionTreeClassifier(max_depth=None, splitter="best", criterion='gini', random_state=1).fit(X_train, y_train)
y_pred = clf.predict(X_test)

# précision du modèle
print("nombre de donnée test", len(list(y_test)))
print("nombre de donnée d'entrainement", len(list(y_train)))
print("score de précision modèle arbre:", metrics.accuracy_score(y_test, y_pred))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))

#graphique de l'arbre de décision
export_graphviz(clf, out_file='arbre_de_décision.txt',
```

```
            filled=True,
            impurity=True,
            feature_names=caracteristiques,
            class_names=['1', '2', '3', '4', '5', '6', '7', '8', '9'])

# matrice de confusion cf. algorithme KNN
# pour voir l'importances des variables
print("importance des données:\n{}".format(clf.feature_importances_))
importances = clf.feature_importances_
print("Sorted Feature Importance:")
sorted_feature_importance = sorted(zip(importances, list(X_train)), reverse='True)
print(sorted_feature_importance)
feat_importances = pd.Series(importances, index=X_train.columns)
feat_importances.nlargest(207).plot.barh(color="blue")
plt.show()
classement_importance = sort(sorted_feature_importance)
print("les 5 données les plus importantes sont:", classement_importance[:5])

#importance des angles
nom = classement_importance[:, 1]
for angle in ('bImpact_poignet_angle',  'impact_poignet_angle', 'aImpact_poignet_angle',
          'bImpact_coude_angle', 'impact_coude_angle', 'aImpact_coude_angle',
          'bImpact_appuis_angle', 'impact_appuis_angle', 'aImpact_appuis_angle'):
   print("la place dans notre classement pour", angle, 'est', list(nom).index(angle)+1)
# le +1 car il commence à compter à 0!

# importance de la raquette
for raquette in ('bImpact_raquette_x', 'bImpact_raquette_y', 'bImpact_raquette_z',
          'impact_raquette_x', 'impact_raquette_y', 'impact_raquette_z',
          'aImpact_raquette_x', 'aImpact_raquette_y', 'aImpact_raquette_z'):
   print("la place dans notre classement pour", raquette, 'est', list(nom).index(raquette)+1)
```

Algorithme DecisionTreeRegressor

```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(max_depth=None, splitter="best", random_state=1).fit(X_train, y_train)
y_pred = dtr.predict(X_test)

export_graphviz(dtr,
            out_file='arbre_de_décision_regression.txt',
            filled=True,
            impurity=True,
            feature_names=caracteristiques,
            class_names=['1', '2', '3', '4', '5', '6', '7', '8', '9'])

# pour voir l'importances des variables cf. algorithm TreeClassificator
# précision du modèle
print("précision des données d'entrainement: {:.3f}".format(dtr.score(X_train, y_train)))
print("coefficient R2:", r2_score(y_test, y_pred, sample_weight=None, multioutput="uniform_average"))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

Algorithme GradientBoostingRegressor

```
from sklearn.ensemble import GradientBoostingRegressor

gbrt = GradientBoostingRegressor(n_estimators=500, max_depth=2, learning_rate=0.1).fit(X_train, y_train)
y_pred = gbrt.predict(X_test)

#pour trouver le nbr d'arbres pour avoir de meilleures performances
errors = [mean_squared_error(y_test, y_pred)
        for y_pred in gbrt.staged_predict(X_test)]
bst_n_estimator = np.argmin(errors)
gbrt_best = GradientBoostingRegressor(max_depth=2, n_estimators=bst_n_estimator, random_state=1)
gbrt_best.fit(X_train, y_train)
y_pred_best = gbrt_best.predict(X_test)
```

```
print("le  nrb d'arbres est " , bst_n_estimator)

# l'importances des variables (Cf. DecisionTreeClassificator)
# précision du modèle
print("coefficient R2:", r2_score(y_test, y_pred_best, sample_weight=None, multioutput="uniform_average"))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred_best)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

Algorithme LinearRegression

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score

model = LinearRegression.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('Coefficients du polynome:', model.coef_)
print("coefficient R2:", r2_score(y_test, y_pred, sample_weight=None, multioutput="uniform_average"))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

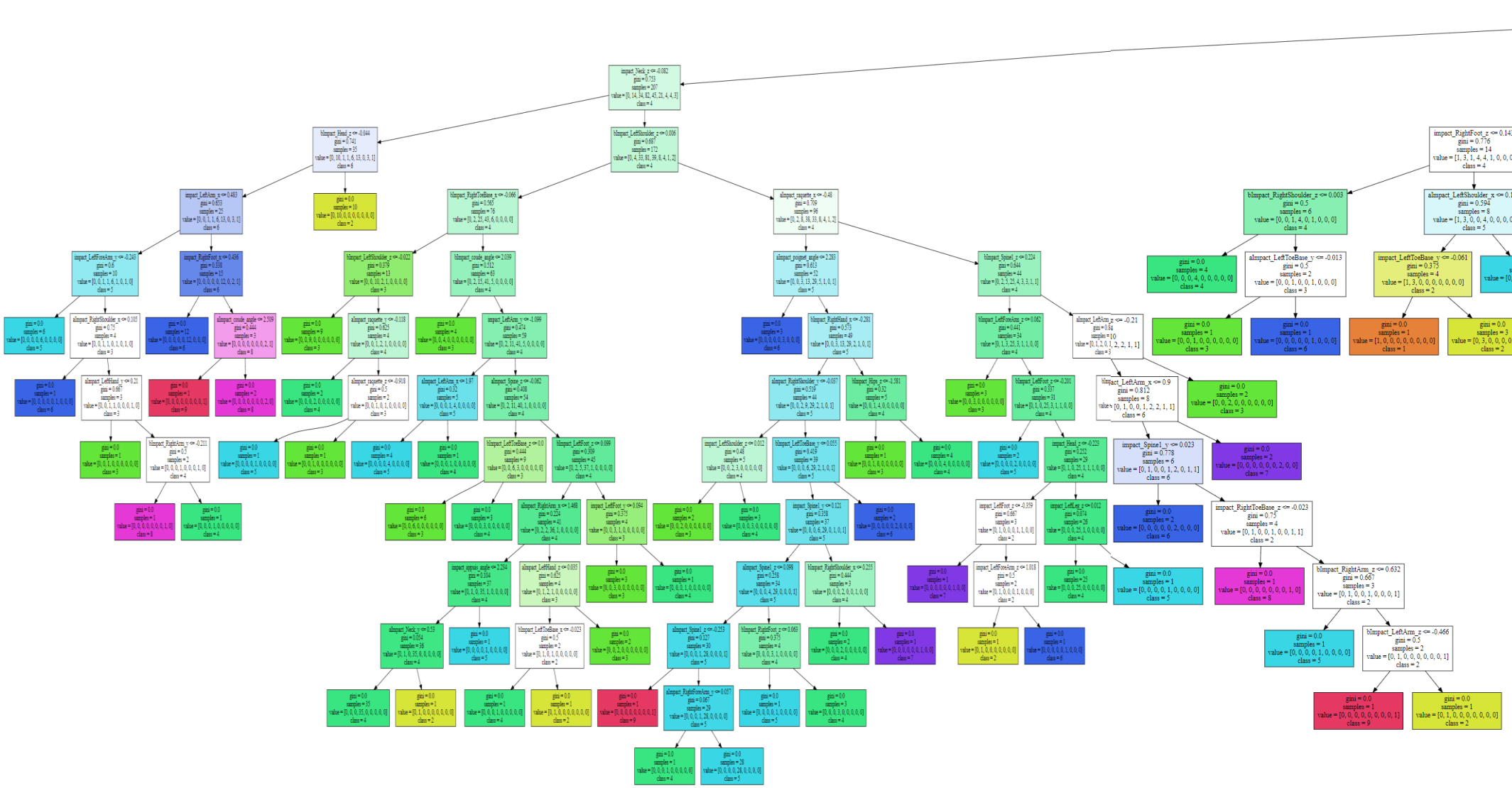
Algorithme de régression polynomial

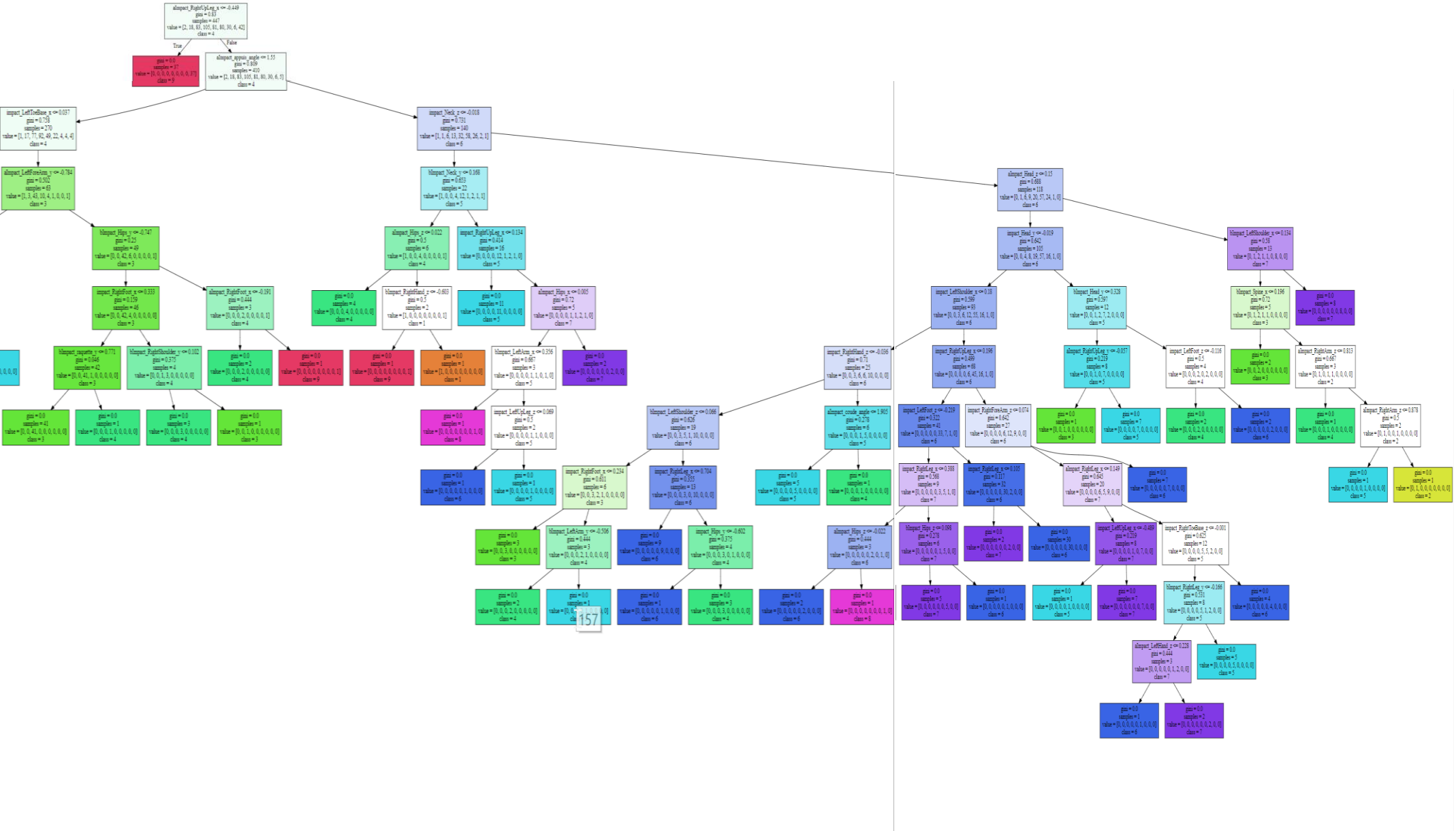
```
from sklearn.preprocessing import PolynomialFeatures

# transformer nos données
X_poly_train = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X_train)
print("grandeur nouvelle matrice: ", np.shape(X_poly_train))
X_poly_test = PolynomialFeatures(degree=2, include_bias=False).fit_transform(X_test)
print(np.shape(X_poly_test))

# créer un modèle et le remplir
model = LinearRegression().fit(X_poly_train, y_train)
y_pred = model.predict(X_poly_test)

# résultats
print('Coefficients du polynome:', model.coef_)
print("coefficient R2:", r2_score(y_test, y_pred, sample_weight=None, multioutput="uniform_average"))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

Algorithme MLPregressor

```
from sklearn.neural_network import MLPRegressor

mlp = MLPRegressor(hidden_layer_sizes=500,
                   max_iter=300, random_state=10, solver='lbfgs',
                   activation='relu').fit(X_train, y_train)

y_pred = mlp.predict(X_test)

# précision du modèle
print("précision modèle MLP:", mlp.score(X_test, y_test))
print("coefficient R2:", r2_score(y_test, y_pred, sample_weight=None, multioutput="uniform_average"))
print("racine de l'erreur quadratique moyenne :", sqrt(mean_squared_error(y_test, y_pred)))
print("erreur moyenne absolue", mean_absolute_error(y_test, y_pred))
```

Annexe 2. Arbre de décision pour DecisionTreeClassifier




bimpact_RightUpLeg_x <= -0.449
gini = 0.83
samples = 447
value = [2, 18, 85, 105, 81, 80, 30, 6, 42]
class = 4
True
False
gini = 0.0
samples = 37
value = [0, 0, 0, 0, 0, 0, 0, 0, 37]
class = 9
bimpact_appui_angle <= 1.55
gini = 0.809
samples = 410
value = [2, 18, 85, 105, 81, 80, 30, 6, 5]
class = 4
bimpact_LeftToeBase_x <= 0.037
gini = 0.758
samples = 270
value = [1, 17, 77, 92, 49, 22, 4, 4, 4]
class = 4
bimpact_Neck_x <= -0.018
gini = 0.791
samples = 140
value = [1, 1, 8, 13, 32, 58, 26, 2, 1]
class = 6
bimpact_LeftForeArm_y <= -0.784
gini = 0.502
samples = 63
value = [1, 3, 43, 10, 4, 1, 0, 0, 1]
class = 3
bimpact_Neck_y <= 0.168
gini = 0.653
samples = 22
value = [1, 0, 0, 4, 12, 1, 2, 1, 1]
class = 5
bimpact_Head_z <= 0.15
gini = 0.681
samples = 118
value = [0, 1, 8, 9, 20, 57, 24, 1, 0]
class = 6
bimpact_Hips_y <= -0.747
gini = 0.25
samples = 49
value = [0, 0, 42, 6, 0, 0, 0, 0, 1]
class = 3
bimpact_Hips_z <= 0.022
gini = 0.5
samples = 6
value = [1, 0, 0, 4, 0, 0, 0, 0, 1]
class = 4
bimpact_RightUpLeg_x <= 0.134
gini = 0.434
samples = 16
value = [0, 0, 0, 0, 12, 1, 2, 1, 0]
class = 5
bimpact_Head_y <= -0.019
gini = 0.642
samples = 105
value = [0, 0, 4, 1, 19, 57, 16, 1, 0]
class = 6
bimpact_LeftShoulder_x <= 0.154
gini = 0.58
samples = 13
value = [0, 1, 1, 1, 1, 0, 8, 0, 0]
class = 7
bimpact_RightFoot_x <= 0.333
gini = 0.159
samples = 46
value = [0, 0, 42, 4, 0, 0, 0, 0, 0]
class = 3
bimpact_RightFoot_x <= -0.191
gini = 0.444
samples = 3
value = [0, 0, 0, 2, 0, 0, 0, 0, 1]
class = 4
gini = 0.0
samples = 4
value = [0, 0, 0, 4, 0, 0, 0, 0, 0]
class = 4
bimpact_RightHand_z <= -0.603
gini = 0.5
samples = 2
value = [1, 0, 0, 0, 0, 0, 0, 0, 1]
class = 1
gini = 0.0
samples = 11
value = [0, 0, 0, 0, 11, 0, 0, 0, 0]
class = 5
bimpact_Hips_x <= 0.005
gini = 0.72
samples = 5
value = [0, 0, 0, 0, 1, 1, 2, 1, 0]
class = 7
bimpact_LeftShoulder_x <= 0.18
gini = 0.599
samples = 93
value = [0, 0, 3, 6, 12, 55, 16, 1, 0]
class = 6
bimpact_Head_y <= 0.328
gini = 0.597
samples = 12
value = [0, 0, 1, 1, 7, 2, 0, 0, 0]
class = 5
bimpact_Spine_x <= 0.196
gini = 0.72
samples = 5
value = [0, 1, 1, 1, 1, 0, 0, 0, 0]
class = 2
gini = 0.0
samples = 8
value = [0, 0, 0, 0, 0, 0, 8, 0, 0]
class = 7
bimpact_RightHand_z <= -0.098
gini = 0.71
samples = 25
value = [0, 0, 3, 6, 6, 10, 0, 0, 0]
class = 6
bimpact_RightUpLeg_x <= 0.196
gini = 0.499
samples = 68
value = [0, 0, 0, 0, 6, 45, 16, 1, 0]
class = 6
bimpact_RightUpLeg_y <= -0.057
gini = 0.219
samples = 8
value = [0, 0, 1, 0, 7, 0, 0, 0, 0]
class = 5
bimpact_LeftFoot_y <= -0.116
gini = 0.5
samples = 4
value = [0, 0, 0, 2, 0, 2, 0, 0, 0]
class = 4
gini = 0.0
samples = 2
value = [0, 0, 2, 0, 0, 0, 0, 0, 0]
class = 3
bimpact_RightArm_y <= 0.813
gini = 0.667
samples = 3
value = [0, 1, 0, 1, 1, 0, 0, 0, 0]
class = 2
bimpact_LeftShoulder_y <= 0.066
gini = 0.628
samples = 19
value = [0, 0, 3, 5, 1, 10, 0, 0, 0]
class = 6
bimpact_create_angle <= 1.905
gini = 0.278
samples = 6
value = [0, 0, 0, 1, 5, 0, 0, 0, 0]
class = 5
bimpact_LeftFoot_y <= -0.219
gini = 0.722
samples = 41
value = [0, 0, 0, 0, 0, 33, 7, 1, 0]
class = 6
bimpact_RightForeArm_y <= 0.074
gini = 0.642
samples = 27
value = [0, 0, 0, 0, 6, 12, 9, 0, 0]
class = 6
gini = 0.0
samples = 1
value = [0, 0, 1, 0, 0, 0, 0, 0, 0]
class = 3
gini = 0.0
samples = 2
value = [0, 0, 2, 0, 0, 0, 0, 0, 0]
class = 4
bimpact_RightArm_y <= 0.878
gini = 0.5
samples = 2
value = [0, 1, 0, 0, 1, 0, 0, 0, 0]
class = 2
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 1, 0, 0, 0, 0]
class = 5
gini = 0.0
samples = 1
value = [0, 1, 0, 0, 0, 0, 0, 0, 0]
class = 2
bimpact_LeftForeArm_y <= -0.784
gini = 0.502
samples = 63
value = [1, 3, 43, 10, 4, 1, 0, 0, 1]
class = 3
bimpact_Hips_y <= -0.747
gini = 0.25
samples = 49
value = [0, 0, 42, 6, 0, 0, 0, 0, 1]
class = 3
bimpact_RightFoot_x <= 0.333
gini = 0.159
samples = 46
value = [0, 0, 42, 4, 0, 0, 0, 0, 0]
class = 3
bimpact_RightFoot_x <= -0.191
gini = 0.444
samples = 3
value = [0, 0, 0, 2, 0, 0, 0, 0, 1]
class = 4
bimpact_neguette_y <= 0.773
gini = 0.046
samples = 42
value = [0, 0, 41, 1, 0, 0, 0, 0, 0]
class = 3
bimpact_RightShoulder_y <= 0.102
gini = 0.375
samples = 4
value = [0, 0, 1, 3, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 2
value = [0, 0, 0, 2, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 0, 0, 0, 1]
class = 9
gini = 0.0
samples = 41
value = [0, 0, 41, 0, 0, 0, 0, 0, 0]
class = 3
gini = 0.0
samples = 1
value = [0, 0, 0, 1, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 3
value = [0, 0, 0, 3, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 1
value = [0, 0, 1, 0, 0, 0, 0, 0, 0]
class = 3
bimpact_LeftArm_x <= 0.256
gini = 0.667
samples = 3
value = [0, 0, 0, 0, 1, 1, 0, 1, 0]
class = 5
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 0, 0, 0, 1]
class = 8
bimpact_LeftUpLeg_z <= 0.069
gini = 0.5
samples = 2
value = [0, 0, 0, 0, 1, 1, 0, 0, 0]
class = 5
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 0, 0, 0, 1]
class = 8
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 1, 0, 0, 0, 0]
class = 5
bimpact_RightFoot_x <= 0.254
gini = 0.611
samples = 6
value = [0, 0, 3, 2, 1, 0, 0, 0, 0]
class = 3
bimpact_FightLeg_x <= 0.704
gini = 0.355
samples = 13
value = [0, 0, 0, 3, 0, 10, 0, 0, 0]
class = 6
gini = 0.0
samples = 3
value = [0, 0, 3, 0, 0, 0, 0, 0, 0]
class = 3
bimpact_LeftArm_y <= -0.506
gini = 0.444
samples = 3
value = [0, 0, 0, 2, 1, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 9
value = [0, 0, 0, 0, 0, 9, 0, 0, 0]
class = 6
bimpact_Hips_y <= -0.602
gini = 0.375
samples = 4
value = [0, 0, 0, 3, 0, 1, 0, 0, 0]
class = 4
gini = 0.0
samples = 2
value = [0, 0, 0, 2, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 1, 0, 0, 0]
class = 6
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 1, 0, 0, 0]
class = 6
gini = 0.0
samples = 3
value = [0, 0, 0, 3, 0, 0, 0, 0, 0]
class = 4
gini = 0.0
samples = 5
value = [0, 0, 0, 0, 0, 5, 0, 0, 0]
class = 5
gini = 0.0
samples = 1
value = [0, 0, 0, 1, 0, 0, 0, 0, 0]
class = 4
bimpact_Hips_z <= -0.022
gini = 0.444
samples = 3
value = [0, 0, 0, 0, 2, 0, 1, 0, 0]
class = 5
gini = 0.0
samples = 2
value = [0, 0, 0, 0, 2, 0, 0, 0, 0]
class = 6
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 0, 1, 0, 0]
class = 8
bimpact_RightLeg_x <= 0.105
gini = 0.117
samples = 32
value = [0, 0, 0, 0, 0, 30, 2, 0, 0]
class = 6
bimpact_FightLeg_x <= 0.149
gini = 0.945
samples = 20
value = [0, 0, 0, 0, 6, 5, 9, 0, 0]
class = 7
gini = 0.0
samples = 7
value = [0, 0, 0, 0, 0, 0, 7, 0, 0]
class = 6
bimpact_RightLeg_x <= 0.388
gini = 0.568
samples = 9
value = [0, 0, 0, 0, 0, 3, 5, 1, 0]
class = 7
bimpact_RightLeg_x <= 0.105
gini = 0.117
samples = 32
value = [0, 0, 0, 0, 0, 30, 2, 0, 0]
class = 6
bimpact_Hips_y <= 0.089
gini = 0.278
samples = 6
value = [0, 0, 0, 0, 0, 1, 5, 0, 0]
class = 7
gini = 0.0
samples = 2
value = [0, 0, 0, 0, 0, 0, 2, 0, 0]
class = 7
gini = 0.0
samples = 30
value = [0, 0, 0, 0, 0, 30, 0, 0, 0]
class = 6
bimpact_LeftUpLeg_x <= -0.489
gini = 0.219
samples = 8
value = [0, 0, 0, 0, 1, 0, 7, 0, 0]
class = 7
bimpact_RightToeBase_x <= -0.001
gini = 0.625
samples = 12
value = [0, 0, 0, 0, 5, 5, 2, 0, 0]
class = 5
gini = 0.0
samples = 5
value = [0, 0, 0, 0, 5, 0, 0, 0, 0]
class = 7
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 1, 0, 0, 0]
class = 6
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 1, 0, 0, 0, 0]
class = 5
gini = 1.0
samples = 7
value = [0, 0, 0, 0, 0, 0, 7, 0, 0]
class = 7
bimpact_RightLeg_y <= -0.166
gini = 0.531
samples = 8
value = [0, 0, 0, 0, 5, 1, 2, 0, 0]
class = 5
gini = 0.0
samples = 4
value = [0, 0, 0, 0, 0, 4, 0, 0, 0]
class = 6
bimpact_LeftHand_z <= 0.228
gini = 0.444
samples = 3
value = [0, 0, 0, 0, 0, 1, 2, 0, 0]
class = 7
gini = 0.0
samples = 5
value = [0, 0, 0, 0, 5, 0, 0, 0, 0]
class = 5
gini = 0.0
samples = 1
value = [0, 0, 0, 0, 0, 1, 0, 0, 0]
class = 6
gini = 0.0
samples = 2
value = [0, 0, 0, 0, 0, 0, 2, 0, 0]
class = 7

Annexe 3 :

Tableau 9

Importances des caractéristiques DecisionTreeClassifier

		Classement	Taux d'importance	%
Angle Poignet				
	bImpact	129	0	0
	impact	87	0	0
	aImpact	33	0.010246364091742953	1.02
Angle coude				
	bImpact	25	0.011638318253029854	1.16
	impact	88	0	0
	aImpact	45	0.008082890311381145	0.81
Angle appui				
	bImpact	130	0	0
	impact	58	0.005101363705030845	0.51
	aImpatc	2	0.06596009833479305	6.6
Position raquette				
bImpact				
	x	128	0	0
	y	53	0.005260293694708362	0.53
	z	127	0	0
impact				
	x	86	0	0
	y	85	0	0
	z	84	0	0
aImpact				
	x	10	0.02106410949784608	2.11
	y	69	0.004041445155690573	0.4
	z	80	0.0026942967704603814	0.27

Note. Le classement a été fait en fonction du taux d'importance. Le nombre total de variables différentes est de 207 données. Les pourcentages ont été arrondis au centième.

Tableau 10

Importances des caractéristiques DecisionTreeRegressor

		Classement	Taux d'importance	%
Angle Poignet				
	bImpact	39	0.0015945117826612813	0.15
	impact	133	0	0
	aImpact	182	0	0
Angle coude				
	bImpact	76	0.0002545969430751292	0.02
	impact	116	2.624710753352627e-05	2.6 e-03
	aImpact	24	0.0031752000886882906	0.31
Angle appui				
	bImpact	84	0.0001607166638078878	0.01
	impact	59	0.0005764989690397903	0.05
	aImpatc	2	0.15498096136944617	15.49
Position raquette				
bImpact				
	x	128	0	0
	y	53	0.0027139315954373714	0.27
	z	127	3.937066130027822e-05	3.9 e-03
impact				
	x	86	0.0006696367314491494	0.06
	y	85	0.0017126237665620982	0.17
	z	84	3.2808884416901126e-05	3.2 e-03
aImpact				
	x	10	1.968533065013911e-05	1.9 e-03
	y	69	0. 00037172466044352	0.03
	z	80	0	0

Note. Le classement a été fait en fonction du taux d'importance. Le nombre total de variables différentes est de 207 données. Les pourcentages ont été arrondis au centième. Notation: e-05 $=10^{-5}$

Tableau 11

Importances des caractéristiques GradientBoostingRegressor

		Classement	Taux d'importance	%
Angle Poignet				
	bImpact	12	0.012911376370576428	1.29
	impact	149	0.00012592045298462037	0.01
	aImpact	145	0.0001437923267542227	0.01
Angle coude				
	bImpact	26	0.00485413118084679	0.48
	impact	182	1.778982838311369e-05	1.7e-03
	aImpact	133	0.00021513948218834222	0.02
Angle appui				
	bImpact	19	0.006880507074890423	0.69
	impact	74	0.0010309203656686827	0.10
	aImpatc	2	0.11725251267652063	11.72
Position raquette				
bImpact				
	x	81	0.0009128567464642771	0.09
	y	30	0.0038370861646365366	0.38
	z	127	0.0002712785719866979	0.02
impact				
	x	117	0.00034964001936170947	0.03
	y	107	0.0004562591130819939	0.04
	z	121	0.0003064584485188989	0.03
aImpact				
	x	17	0.007363114327226541	0.74
	y	7	0.022249044670892244	2.22
	z	146	0.0001431402493939934	0.01

Note. Le classement a été fait en fonction du taux d'importance. Le nombre total de variables différentes est de 207 données. Les pourcentages ont été arrondis au centième. Notation: e-05 =10^{-5}

Remerciements

J’adresse mes remerciements à toutes les personnes qui m’ont aidé de proche ou de loin dans la réalisation de ce travail de master.

Je remercie Thibaut Le Naour pour sa disponibilité et son aide pour la mise en pratique de cette expérience et surtout pour la résolution des divers soucis techniques.

Je remercie également ma famille pour la relecture de ce travail.

Je remercie Lionel Lugon pour cette sympathique collaboration et pour les précieux conseils qu’il a pu me donner dans le domaine du tennis.

Je remercie les différents experts et sujets qui ont pris le temps de s’investir pour ce travail.