UNIVERSITÉ DE FRIBOURG
UNVERSITÄT FREIBURG

SCIENCES DU MOUVEMENT ET DU SPORT
BEWEGUNGS- UND SPORTWISSENSCHAFTEN

Comparaison de méthodes d'apprentissage pour la prédiction de performance en course de montagne

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

déposé par

Christelle Crettol

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

en collaboration avec la
Haute école fédérale de sport de Macolin

Référent
Dr Thibaut Le Naour

Fribourg, juin 2019

Table des matières

Résumé 3
1 Introduction 4
1.1 Apprentissage automatique - « Machine Learning » 4
1.2 Programme de test des différents apprentissages 7
1.3 Principaux algorithmes 8
1.4 Objectif du travail 14
2 Méthode 15
2.1 Participants 15
2.2 Récolte des données 15
2.3 Application d'extraction, filtration et exportations des données 15
2.4 Algorithme sur Python 17
3 Résultats 21
3.1 Comparaison des régressions pour le modèle individuel 21
3.2 Comparaison des régressions pour la prédiction de course 22
3.3 Disparité de performance dans les jeux de données 31
4 Discussion 32
4.1 Les données d'entraînement permettent-elles de pouvoir prédire une performance en course de montagne ? 32
4.2 Comment évaluer et améliorer les prédictions ? 35
4.3 Forces, limitations et perspectives 36
5 Conclusion 37
Bibliographie 38
Annexe 39
Remerciements 48

Résumé

Connaissance. Il est très intéressant pour des coureurs sur route, de fond ou demi-fond de prédire leurs performances avant les courses afin de mieux planifier celles-ci. On peut les réaliser en utilisant le VO2max (consommation d'oxygène) ou encore la VMA (vitesse maximale d'aérobie). Notre travail a pour objectif d'étudier les prédictions de performance en course de montagne basées sur des données d'entraînement et de les évaluer au moyen d'algorithmes issus de l'apprentissage automatique.

Méthode. Les données d'entraînements de 52 sujets de tous âges et tous niveaux en course à pied ont été extraites, filtrées puis utilisées dans Python (langage de programmation). Des algorithmes d'apprentissage automatique tels que la régression linéaire (LIN), la régression polynomiale (POLY) et le MLP régresseur ont été choisis pour établir des prédictions individuelles. Les valeurs prédites par les algorithmes et celles enregistrées en entraînement ont pu être comparées. De plus, une quinzaine de sujets ont participé à la course Saillon-Ovronnaz et ont permis de vérifier dans la pratique des prédictions. Une autre approche visant à prédire ces mêmes temps de course (i.e. sur la course Saillon-Ovronnaz) basée sur des tests physiques a également été proposée par un autre étudiant.

Résultats. La moyenne des erreurs relatives a été calculée pour trois méthodes (i.e. POLY, MLP, LIN) à partir de deux jeux de données (des entraînements sélectionnés aléatoirement dans les données de départ ainsi que sur celles « terrain » récoltées à partir d'une course ; POLY : entr. 35.75% et course 35.18%, MLP : entr. 9.45% et course 51.39% et LIN : entr. 7.22% et course 24.16%). Trois cas ont retenu notre attention. Le sujet 6 obtenait un pourcentage d'erreur relative pour la prédiction de course de 3.95% pour POLY contre 52.10% pour MLP. Le sujet 16, quant à lui recueillait une meilleure prédiction de course avec MLP soit une erreur relative de 3.03% (POLY 43.82% et LIN 15.48). Le sujet 14 possédait un taux d'erreur relative très élevé (POLY 69.05%, MLP 79.64%, LIN 75.55%) bien que la course de montagne était son terrain d'entraînement. Ces trois sujets démontrent une forte variabilité des modèles et une sensibilité importante au bruit. La méthode par les tests physiques est plus concluante avec 6.25% d'erreur relative pour la prédiction de course.

Conclusion. Conformément aux hypothèses de recherche, les algorithmes ont conduit à une prédiction de performance en course de montagne. Cependant les modèles utilisés démontraient une grande variabilité des données. D'une part, des solutions pour éliminer les données non pertinentes devraient être recherchées et d'autre part, les valeurs minimums devraient être prises en considération afin d'améliorer les prédictions.

1 Introduction

La prédiction de performance est souvent réalisée lors de courses d'endurance sur route telles que les 10 kilomètres, les semi-marathons et les marathons. Des prédictions basées sur différentes mesures physiologiques couplées à des descripteurs du contexte de l'épreuve sont aussi proposées. Un autre prédicteur comme l'économie de fonctionnement (RE) peut être suggéré. Cette dernière est définie comme la demande d'énergie pour une vitesse donnée en mesurant la consommation d'oxygène (VO2) à un régime permanent (Saunders, Pyne, Telford & Hawley, 2004). D'autres prédicteurs peuvent également être utilisés comme la consommation maximale d'oxygène (VO2max) ou encore la vitesse maximale aérobie (VMA). Depuis plusieurs années, la course de montagne et le trail jouissent d'une popularité grandissante, il serait donc intéressant de pouvoir prédire les performances de ces deux disciplines. De nos jours, la mise en place de telles prédictions reste encore incertaine. En effet, beaucoup de paramètres différents sont à prendre en compte, comme le dénivelé ou encore le relief du terrain plus ou moins technique. Il est donc impossible d'utiliser les mêmes modèles de prédiction que sur un marathon sur route et sans dénivelé. Par exemple, pour une course comme Sierre-Zinal (31 kilomètres pour 2200 mètres de dénivelé positif et 1100 mètres de dénivelé négatif), les modèles du marathon de Lausanne (42 kilomètres 195 pour 185 mètres positifs et 218 négatifs) seraient inapplicables. De tels paramètres ont donc une réelle influence sur la performance. Il sera alors intéressant de pouvoir réfléchir sur la façon de mettre en place de telles prédictions. Une possibilité serait d'utiliser les données d'entraînement de plusieurs athlètes pour pouvoir définir des algorithmes d'apprentissage qui mettraient en relation les paramètres de dénivelé, de distance et de temps afin d'estimer une performance en fonction du terrain parcouru. Une autre possibilité d'établir des prédictions, serait de tester les sujets avec des tests physiques spécifiques.

1.1 Apprentissage automatique - « Machine Learning »

Depuis les années 1950, des chercheurs travaillent sur l'intelligence artificielle en se basant sur les modèles et méthodes de la logique (Haton, 2000). Les objectifs sont, non seulement la compréhension de l'intelligence propre des individus mais également la capacité de construire des entités intelligentes (Russel & Norvig, 2010). Le « Machine Learning » ou apprentissage automatique est un domaine important de l'intelligence artificielle. Il se base sur des approches statistiques, probabilistes et d'optimisation afin de permettre aux ordinateurs d'apprendre directement à partir de données. L'apprentissage automatique peut donc être utilisé dans le domaine de la course à pied. En effet, la prédiction de performance intéresse depuis toujours la

population sportive. L'athlète s'entraîne toujours davantage pour obtenir de meilleures performances et son intérêt grandit face aux prédictions de celles-ci en compétition. Il est très intéressant de connaître pour un marathonien quel sera son temps de course afin de pouvoir gérer au mieux son effort ou encore ses ravitaillements. D'un point de vue simplifié, l'apprentissage automatique se compose de deux phases : la phase d'apprentissage et celle de test. La première consiste en la conception d'un modèle à partir de données disponibles. La seconde phase permet de tester le modèle élaboré sur de nouvelles données, excellent moyen pour obtenir un retour sur sa qualité. L'apprentissage peut être qualifié de différentes manière en fonction des données: supervisé, non-supervisé, semi-supervisé ou encore par renforcement. Deux méthodes d'apprentissage peuvent être appliquées : la classification et la régression. Si les données sont discrètes, la méthode de classification sera utilisée ; dans le cas contraire, si elles sont continues, la méthode de régression sera appliquée. Par exemple, si l'on veut prédire le poids d'une personne en utilisant sa taille, la méthode de régression serait appliquée car elle met en lien des variables numériques. La première étape de la méthode élaborera un modèle avec les données taille-poids et la seconde établira des prédictions. L'évaluation du modèle pourra s'établir grâce aux erreurs commises (poids prédit – poids réel). La dernière étape consistera à optimiser la méthode afin de trouver la meilleure prédiction possible.

1.1.1 Applications. Il existe de nombreuses applications de l'apprentissage automatique dans la vie de tous les jours. Lorsqu'un acheteur veut estimer le prix d'une maison en fonction de ses places de parking ou de sa superficie alors l'intelligence artificielle est utilisée. Elle comparera le bien estimé avec d'autres biens immobiliers similaires. L'apprentissage automatique est donc très utilisé pour recommander des produits ; « Netflix », entreprise proposant des films ou épisodes sur internet, ou encore « Amazon », entreprise soumettant livres et autres produits, utilisent ce modèle pour conseiller des films ou vendre d'autres articles aux clients en fonction de leurs précédentes recherches réalisées en ligne. Dans le milieu bancaire, l'apprentissage automatique sert à détecter les comportements frauduleux et anormaux des usagers en se basant sur leurs transactions antérieures. « Siri », application de commande vocale, est un exemple concret de l'utilisation de l'apprentissage automatique. Ses algorithmes utilisent la voix pour les traduire en texte. Le domaine médical en a aussi recours pour diagnostiquer des maladies. Les algorithmes se basent sur les données médicales d'une personne et peuvent alors prédire si la maladie est présente. En effet, un problème peut être diagnostiqué avant qu'il se produise, comme dans le cas d'une crise cardiaque.

1.1.2 L'apprentissage supervisé. Utilisé dans ce travail, il équivaut au type d'apprentissage automatique le plus courant. Lorsqu'un phénomène est observé, deux variables ressortent toujours : les variables prédictives dites aussi explicatives et les variables cibles. Les variables prédictives sont les variables de bases sur lesquelles les prédictions sont établies. Elles se notent sous la forme d'un vecteur $x = (x_1, \ldots, x_m)$ à m composantes, m correspondant au nombre d'observations. Dans le cas d'un marathon ou d'une course de montagne, les variables prédictives sont entre autres la distance, le dénivelé positif, la température extérieure ou encore l'âge du coureur. La variable cible, notée *y*, quant à elle, équivaut à la variable que l'on souhaite prédire. Dans notre exemple, il s'agirait du temps réalisé lors d'une course. La variable cible peut s'écrire comme une fonction du vecteur x plus une erreur, toutes deux inconnues.

$$y = F(x) + \epsilon(x)$$

Un algorithme d'apprentissage automatique permettra de trouver une approximation de *F(x)* appelée *f(x)* par rapport à l'ensemble des observations. Elle équivaut à la fonction de prédiction.

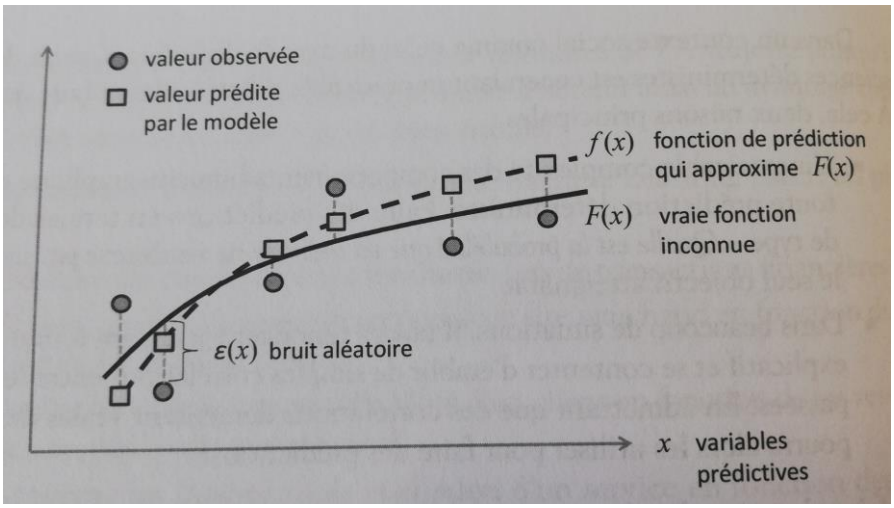


Figure 1. Représentation de la fonction de prédiction f en fonction de la vraie fonction F(x). La fonction ε(x) représente l'erreur (Lemberger, Batty, Morel & Raffaelli, 2017, p.112).

L'apprentissage automatique dans ce type de modèle, se définit alors comme étant un ensemble de structures mathématiques qui permet de construire une fonction f à partir d'observations. Il servira à trouver des corrélations entre les variables prédictives et les variables cibles. Dans notre exemple, le modèle permettra d'établir un lien entre le temps effectué et les différents paramètres propres à la course.

1.1.3 Apprentissage non-supervisé. Dans ce type d'apprentissage, les observations sont modélisées. Il n'y a pas de données *y* qui servent à établir une fonction de prédiction. A partir directement des observations, l'algorithme d'apprentissage automatique constitue un modèle qui améliore les observations et les classe par groupe en fonction de leurs caractéristiques. Il est très utilisé dans les campagnes publicitaires afin de cibler les comportements similaires des acheteurs par exemple.

1.1.4 Apprentissage semi-supervisé. Il se situe entre l'apprentissage supervisé et non-supervisé. Une partie des données seulement sert à l'élaboration de la fonction f. Il permet un gain considérable de temps si la banque de données est démesurée. En effet, grâce à ce type d'apprentissage, il suffit d'avoir certains y_i mais il ne nécessite pas de requérir beaucoup de travail pour trouver chacun d'entre eux. Il est employé, par exemple dans le cas de la classification d'images.

1.1.5 Apprentissage par renforcement. Il dépend de l'environnement et peut interagir avec celui-ci. Si ces actions sont positives alors il sera récompensé favorablement et dans le cas contraire, il recevra une récompense négative. Il est très exploité dans les jeux comme les échecs, par exemple.

1.2 Programme de test des différents apprentissages

Les données que l'on possède sont séparées en deux groupes : les données d'entraînement que l'on nomme *nTrain* et les données de test, *nTest*. Les données *nTrain* sont particulièrement utiles dans la phase d'apprentissage et permettent de trouver la fonction de prédiction. *nTest* sert de son côté, à comparer les prédictions et les valeurs cibles.

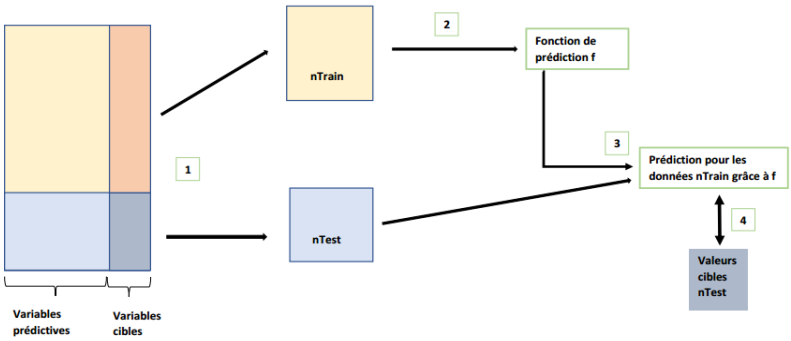


Figure 2. Représentation de la séparation des données : (1) Séparation *nTrain* et *nTest*, (2) Algorithme grâce à *nTrain* trouve la fonction de prédiction f, (3) Prédictions faites sur les données *nTest* grâce à *f*, (4) Comparaison des valeurs cibles de *nTest* et les prédictions.

Aucune règle ne définit combien de données sont utilisées dans le vecteur *nTrain*. Une répartition usuelle sépare les données en 80% pour l'entraînement et 20% pour les tests (Lemberger, Batty, Morel & Raffaelli, 2017). Dans ce travail, deux-tiers des données soit 67% sont utilisées dans le vecteur *nTrain* et le solde dans *nTest*.

1.3 Principaux algorithmes

Il existe énormément d'algorithmes utilisés pour l'apprentissage automatique. Dans notre cas, nous possédons m observations avec comme variable cible *y*.

1.3.1 La régression linéaire. La régression linéaire se définit comme une régression connue appartenant à l'apprentissage supervisé. Elle équivaut à un modèle paramétrique, ce qui signifie que la fonction de prédiction *f* possède une forme particulière et un nombre de paramètres spécifié à l'avance. Dans notre cas :

$$f(x) = a_1x_1 + a_2x_2 + ... + a_mx_m + b = ax + b.$$

Elle établit une relation linéaire entre une variable prédictive et la variable cible *y*. Les coefficients *a* et *b* doivent être minimisés pour diminuer les erreurs de prédictions des données d'apprentissage. Pour rappel, la linéarité explique un rapport proportionnel et constant entre deux variables. Il existe plusieurs manières d'approximer une régression linéaire : la méthode des moindres carrés, le maximum de vraisemblance ou encore l'interférence bayésienne. La méthode des moindres carrés, la plus utilisée, permet de définir l'erreur comme étant la somme des carrés des écarts entres les valeurs prédites par $f(x_i)$ et les valeurs cibles y_i. Il y a plusieurs domaines où l'apprentissage automatique utilise la régression linéaire : en économie, par exemple, lorsque l'on lie le prix d'un produit et sa demande.

Dans notre travail, on utilisera le modèle linéaire sous cette forme :

$$t = \alpha \frac{d}{Vfloat} + \beta \frac{D+}{VA} + \gamma \frac{D-}{VD}$$

Avec :

- *t = temps (seconde)*
- *d = distance (mètre)*
- *Vfloat = vitesse à plat (mètre/seconde)*
- *VA = vitesse ascensionnelle (mètre/seconde)*
- *VD = vitesse de descente (mètre/seconde)*
- *D+ = dénivelé positif (mètre)*
- *D- = dénivelé négatif (mètre)*
- *α , β, γ = coefficients*

Il existe de nombreux avantages à utiliser la régression linéaire. Pour l'élaboration des prédictions, il suffira simplement d'inverser la matrice basée sur les données d'apprentissage. De plus, le calcul de prédiction est très rapide car il ne dépend pas d'un modèle numérique complexe. La régression linéaire s'interprète facilement, autre avantage conséquent. Il existe cependant des inconvénients comme le fait que la linéarité entre deux variables n'est pas naturelle. Le modèle démontre également une grande variabilité aux valeurs anormales ce qui demande une grande attention sur le jeu de données afin de ne pas détériorer la prédiction.

1.3.2 La régression polynomiale. La régression polynomiale, quant à elle, observe et analyse la variation d'une variable explicative. Un polynôme, comme son nom l'indique additionne plusieurs monômes dont le degré décrit la puissance la plus élevée de ses monômes. Un polynôme de degré 1, graphiquement correspond à une droite, un degré 2 à une expression quadratique (sous forme de parabole) et pour finir, un degré 3 à une expression cubique. Il est donc possible de pouvoir comparer deux courbes en variant le degré du polynôme. Si nous ajoutons un degré au polynôme, il deviendra plus sensible aux données. La régression polynomiale permet de décrire une variable aléatoire en utilisant une fonction polynomiale d'une variable aléatoire explicative. On utilisera donc des données connues afin d'expliquer les données encore inconnues. Par exemple, lorsque l'on prend le couple (X_i,Y_i), correspondant au i^{ème} couple de variables aléatoires. Y_i s'écrit comme la somme d'un polynôme en X_i et un reste ε.

$$Y_i = f(X_i) = P_n(X_i) + \epsilon_i \text{ avec } P_n(X_i) = a_n x^n + a_{n-1}x^{n-1} + ... + a_0$$

P_n (X_i) est le polynôme en X_i, α_i sont les coefficients et x les composantes de X_i.

Nous avons utilisé dans notre cas, une régression polynomiale de degré 2. Ce qui veut dire que l'on a deux variables prédictives X_1 et X_2 et que l'on se retrouvera avec un modèle de cette forme :

$$P_2(Xi) = ax_1 + \beta x_2 + \gamma x_1^2 + \delta x_2^2 + \epsilon.$$

Dans le cas d'un polynôme de degré d, on cherchera une fonction de prédiction de la forme :

$$Y = \beta_{00} + \sum_{j=1}^{n} \beta_{1j}x_j + \sum_{j=1}^{n} \beta_{2j}x_j^2 + ... + \sum_{j=1}^{n} \beta_{nj}x_j^n$$

Les variables d'entrée se transforment donc grâce au polynôme. Il s'agit d'une régression linéaire sur les *n x d* variables : $x_1, x_2, ..., x_n, x_1^2, x_2^2, ..., x_n^2, ..., x_1^d, x_2^d, ..., x_n^d$.

Le but alors de notre régression se résume à trouver un modèle prédictif. Nous recherchons donc une fonction de prédiction qui calculera le temps que les sujets établiront sur différents parcours en fonction de leurs données d'entraînement.

1.3.3 Le perceptron multi-couches. Appelée MLP, la régression perceptron multi-couches équivaut à une régression non-linéaire. Elle est divisée en réseau de neurones qui font passer les informations d'une couche d'entrée à une couche de sortie. Le réseau de neurones, base pour créer un modèle de prédiction, va permettre de décomposer une illustration en plusieurs séquences. De manière concrète, des similarités doivent être trouvées entre cette dernière et un tableau grâce à un réseau de neurones. Chaque neurone possèdera alors un petit carré du tableau et lorsqu'un neurone détecte une ressemblance entre l'illustration et le bagage qu'il possède, il s'activera. Il faudra alors de nombreux neurones pour créer l'image finale. C'est donc en associant le travail de plusieurs neurones que l'on obtiendra l'image.

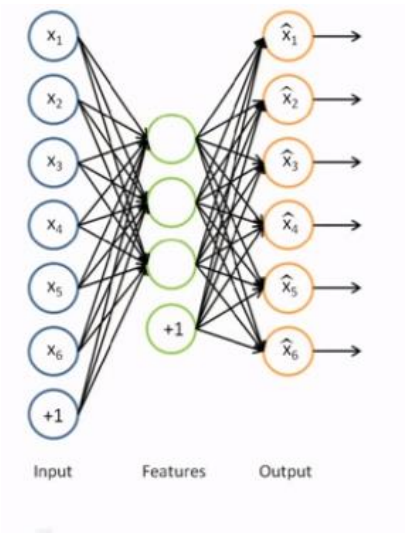


Figure 3. Construction de réseaux de neurones lors de l'apprentissage profond (Ng, 2019).

Un exemple de réseau est donné par la figure ci-dessus. L'image d'entrée donnée par les ronds bleus, représente une image connue. Les ronds orange équivalent à l'image de sortie. Il existe donc une formation de réseau permettant la prédiction de cette image. Elle est représentée sous la forme d'une matrice. Les ronds verts sont quant à eux des encodeurs capables de prendre des images de taille 6 pixels et de les encoder avec seulement 3 pixels. Le système force ainsi le réseau de neurones à représenter le monde en un nombre restreint de pixels. Il suffit alors d'enlever la couche de sortie et de la remplacer par un vecteur caractéristique $a = (a_1, a_2, a_3)$. Il existe alors un stock de caractéristiques, représenté par le vecteur a tel qu'il est possible de pouvoir en tout temps le recycler à d'autres moments. Par exemple, si nous apprenons en Afrique à définir un animal dangereux à ses dents pointues, lorsque nous partons au Canada et tombons sur un animal inconnu aux dents aiguisées, nous saurons qu'il est dangereux. Dans ce

cas, les caractéristiques apprises Afrique (i.e. le vecteur *a* représente les caractéristiques typiques des dents) sont utilisées à nouveau dans un autre endroit et ainsi recyclées.

En résumé, MLP est un modèle multicouches composé de :

- Une couche de i cellules d'entrée.
- Une couche de j neurones cachés d'activation.
- Une couche de sortie à k neurones d'activation.

$\left.\begin{matrix} \\ \\ \end{matrix}\right\}$ *i x j* connexions

$\left.\begin{matrix} \\ \\ \end{matrix}\right\}$ *j x k* connexions

Malheureusement, désavantage conséquent, nous ne pouvons pas connaître les dimensions du réseau MLP.

Les sessions suivantes seront moins abordées dans le détail car ce travail ne portera pas sur ce type de régressions. Néanmoins, elles restent très utilisées dans le cas de l'apprentissage automatique et donc il est important de les mentionner.

1.3.4 Les k plus proches voisins. C'est un cas d'apprentissage supervisé et non paramétrique. Cet algorithme se base sur la notion de regroupement d'éléments voisin entre eux. La notion de voisinage repose sur une métrique de comparaison spécifique. Une dimension de distance dans l'espace est mise en place afin de pouvoir rassembler les points les plus semblables. Elle correspond à un modèle de classification où chaque observation dépend des k points les plus proches afin de les classer. L'avantage de cet algorithme ne suppose absolument rien à propos de la distribution des données au début et que la méthode est relativement simple à utiliser. Malgré tout, ce modèle est très sensible aux valeurs aberrantes et très couteux pour un jeu de données conséquent. Dans notre cas, un problème subsisterait car il demanderait une attention très précise sur chaque donnée d'entraînement et serait coûteux en nettoyage de données.

1.3.5 La classification naïve bayésienne. Elle appartient aussi à un apprentissage supervisé. Elle est dotée d'un filtre qui va trier les données et les répartir dans plusieurs catégories distinctes en supposant que les variables sont indépendantes entre elles. Par exemple, dans le cas binaire (i.e. répartition en deux catégories distinctes) et en prenant un contexte de reconnaissance d'image, lorsque l'on prend le filtre fleur, le modèle prédira si l'image correspond à la catégorie fleur ou non-fleur. L'estimation est plus facile si le modèle possède à l'avance un échantillon dont il connaît l'appartenance aux catégories. Dans notre vie actuelle, il correspond au modèle appliqué pour trier les E-mails partant dans les spams et ceux arrivant dans la boîte de réception. Il équivaut à un algorithme simple et très efficace. Cependant, il reste inutilisable en cas d'indépendance entre les variables explicatives conditionnées sur les classes.

1.3.6 La régression logistique. Modèle de classification linéaire où la particularité est que la variable cible ne peut prendre seulement que deux valeurs : 1 et 0. Une fonction *S(X)* est utilisée et appelée fonction de score correspondant à un modèle linéaire employant les variables prédictives et des coefficients i_1, i_2, … i_n. Le but se résumant à trouver des coefficients afin que cette fonction soit positive lorsque la probabilité de la donnée d'appartenir au groupe 1 soit grande et négative et que celle d'appartenir au groupe 0 soit aussi grande. La régression logistique est notamment employée lorsque l'on veut prédire le taux de mortalité d'une maladie infectieuse en fonction des caractéristiques des patients. La rapidité de sa classification et la simplicité équivalent aux grands avantages de cette régression. Malheureusement, la complexité de l'optimisation des coefficients peut entraîner une longue phase d'apprentissage.

1.3.7 Les arbres de décisions. Modèles de classification et de régression, ils fonctionnent en segment où chaque nœud qui les sépare correspond à une question relative aux valeurs des variables prédictives. Durant la phase d'apprentissage, les bonnes questions devront être posées pour que toutes les observations se trouvent classées. Un exemple dans le domaine de la course équivaudrait à pouvoir classer les coureurs en différentes catégories en fonction de leurs entraînements. Dans la figure 4, par exemple, si le sujet court plus de trois fois par semaine alors il s'engagera dans la branche de gauche et au contraire s'il s'entraîne moins de trois fois, il se dirigera sur celle de droite et ainsi de suite.

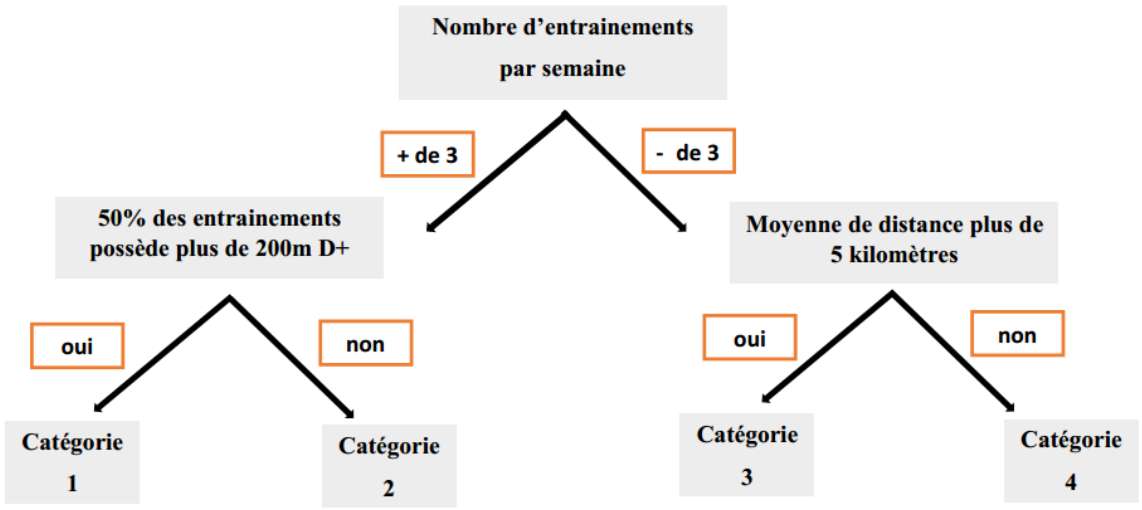


Figure 4. Construction d'un arbre de décision pour le domaine de la course.

L'avantage premier de ce type d'algorithme se résume à ce que les variables soient quantitatives comme le temps ou la distance ou alors qualitatives comme le niveau des coureurs. L'algorithme est robuste vis-à-vis du manque de données et de ce fait, le modèle donnera lieu à des solutions équivalentes. En revanche, le jeu de données doit être conséquent afin que l'arbre amène à de bonnes prédictions.

1.3.8 Les forêts aléatoires. Elles équivalent à la forme améliorée des arbres de décisions. En effet, cet algorithme a pour but d'éliminer les inconvénients de ce dernier. La première étape consiste à faire un « bootstrap », c'est-à-dire à créer des nouveaux échantillons de même taille en établissant un tirage sans remise des observations décrites au moyen des variables prédictives. Le bootstrap est souvent utilisé en statistiques pour estimer un paramètre en fonction de son estimation (Azencott, 2018). Une partie de ces variables seront choisies aléatoirement afin d'effectuer la meilleure segmentation. L'algorithme combinera alors un grand nombre d'arbres de décisions pour en créer un plus puissant. Lorsque l'on bénéficiera d'une nouvelle observation, elle passera par tous les arbres et la classe majoritaire sera sélectionnée afin que l'observation y soit classée. Elle correspond au meilleur modèle actuellement de prédiction en termes de précision. Cependant, en tant qu'algorithme très complexe, une librairie existante est vivement souhaitée.

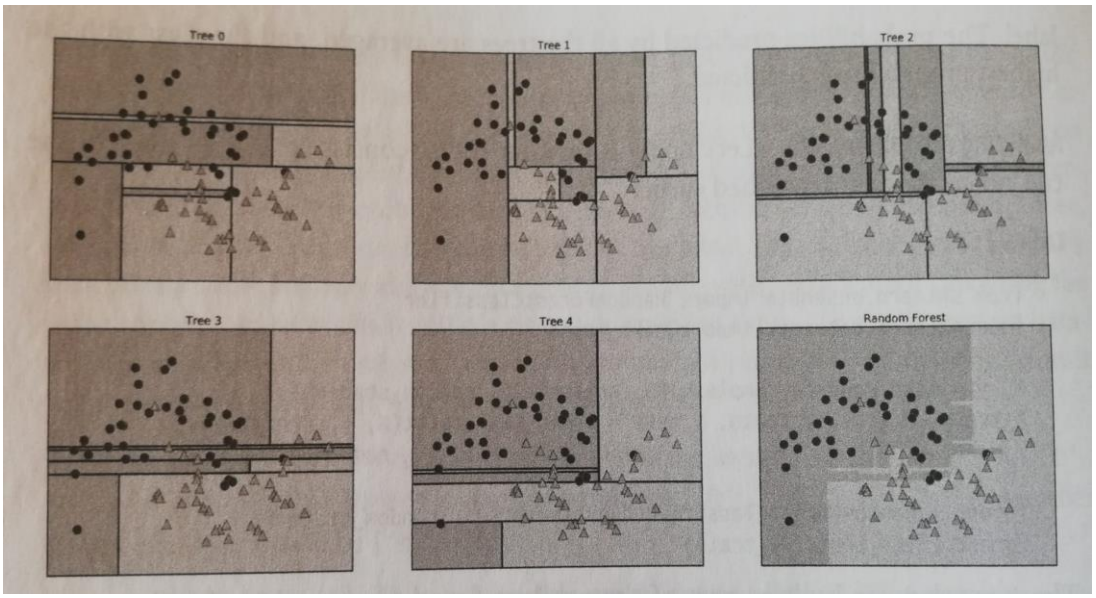


Figure 5. Représentation de forêt aléatoire avec un ensemble de cinq arbres de décision (Müller & Guido, 2017, p.88).

1.3.9 Autres régressions. Il existe encore de nombreux autres modèles comme la régression non paramétrique, l'algorithme des k-moyennes, les machines à vecteurs de support …

Dans ce travail de master, il est question de trouver une relation entre une variable cible comme le temps et plusieurs variables explicatives, ici le dénivelé positif, la pente et la distance effectuées en entraînement. Dans le modèle linéaire, les vitesses à plat, ascensionnelles et de descente seront également des variables explicatives. Le but ultime étant de pouvoir prédire le temps d'une course en fonction des données d'entraînement des sujets. La régression polynomiale, la régression MLP ainsi que le modèle linéaire ont été choisis afin de pouvoir comparer les méthodes en fonction des données d'entraînements.

1.4 Objectif du travail

L'objectif de ce travail est de savoir si, grâce aux données d'entraînements en course à pied, il est possible de prédire une performance en course de montagne. C'est pourquoi, nous paramétrons puis testons des algorithmes d'apprentissage automatique programmés sur Python à partir des données d'entraînement. A cet effet, nous nous sommes posés les questions suivantes :

a) Les données d'entraînement permettent-elles de pouvoir prédire une performance en course de montagne de la même manière que l'on prédit une performance pour des courses sans dénivelé ?

b) Comment évaluer la prédiction et comment l'améliorer en cas de résultats ?

2 Méthode

2.1 Participants

Par le biais d'un flyer mis sur les réseaux sociaux, avec l'aide du Trail du Mouret et du Trail des patrouilleurs à Montana, 52 sujets se sont intéressés à notre travail et prirent part à notre étude en nous contactant directement. Ils correspondent à des sportifs de tous niveaux et de tous âges (21 à 62 ans), dont la course est pour certains, un quotidien alors que pour d'autres, une activité du dimanche. Tous étaient très motivés et nous avons entretenu une relation par e-mail durant tout le long de notre travail.

2.2 Récolte des données

Les données d'entraînements enregistrées sur les montres connectées ont été récupérées pour établir notre jeu de données. Les sujets se sont occupés d'extraire leurs propres données sur le site internet correspondant à chaque montre (flow.polar.com pour les montres Polar, connect.garmin.com pour les Garmin et movescount.com pour les Suunto). Un champ devait être sélectionné pour obtenir seulement les données de course à pied, celles qui nous intéressaient. Ils devaient ensuite télécharger chaque entraînement individuellement en document « CSV » ou « TCX ». Une année complète de données était requise afin de recueillir un panel d'entraînement sur toute saison (car lors des mois d'hiver le taux d'entraînement diminuait) et permettait d'obtenir une représentation significative. Pour les sujets très entraînés, plus de 300 entraînements ont été extraits.

2.3 Application d'extraction, filtration et exportations des données

Nous avons créé des applications différentes en fonction des montres utilisées afin d'extraire les données pour le travail. Les montres « Garmin » et « Suunto » sont basées sur le format « GPX » permettant une géolocalisation en tout temps. Elles décrivent le positionnement du coureur en fonction des points de cheminement, de l'itinéraire ou encore de la trace. Les points de cheminement correspondent principalement à l'altitude en mètre, au temps en seconde et à l'orientation « GPS ». L'itinéraire évoque un lien entre la position du coureur et un itinéraire type sur une page web. Finalement la trace représente les coordonnées « GPS » du coureur données par des satellites. Les montres « Polar » sont quant à elles au format « CSV », format texte ouvert qui représente les données à l'image d'un tableau de valeurs.

2.3.1 Erreurs commises par les montres. Une des difficultés principales se résume au fait qu'une montre commet régulièrement des erreurs « GPS ». Il existe de nombreux désavantages comme le manque de précision du système de positionnement par satellite (précision de 3.35 mètres à 95% du temps), les différences de fréquence de la mesure (chaque seconde, deux secondes…), le décrochage du signal « GPS », l'enclenchement lors du départ, les virages alors que les GPS sont plus sensibles aux lignes droites, les tunnels, les arbres et les immeubles ou encore les nuages. Tous ces problèmes qui surviennent lors de la capture « GPS » rendent de moins en moins précis les données d'entraînement.

2.3.2 Conversion et réarrangement. Le temps, la distance, la pente, la fréquence de capture ont dû être convertis et réarrangés afin d'obtenir un jeu de données commun aux différents types de montres. La conversion du système de temps fut un point très important. En effet, les montres « Garmin » donnent les entraînements sous forme de date alors que celles « Polar », par exemple, l'établissent en seconde. Le pourcentage de pente a aussi été réarrangé et il a fallu calculer la distance dans certains cas. Les montres « Suunto » donnent une latitude et une longitude pour chaque étape de l'entraînement qu'il fallait alors convertir en distance. La fréquence de captation des données a aussi été adaptée car chaque montre se différenciait lors de l'enregistrement de la donnée allant d'une seconde à cinq secondes.

2.3.3 Filtration et exportation des données. Lorsque le réarrangement était exécuté, la filtration a pu être faite. Les données possédaient de nombreux problèmes d'altimétrie comme parfois, le décrochage de 8 mètres en 1 seconde, vraisemblablement dû à la synchronisation de la montre. Un point important de la filtration résidait dans le fait que certains sujets pratiquaient des entraînements en intervalles, ce qui pouvait fausser nos prédictions. L'étape suivante consistait à l'exportation des paramètres qui nous intéressaient grâce à des applications. Les données brutes représentaient le temps, en date ou en secondes, la position « GPS » ou la latitude/longitude, la distance parcourue en mètres, l'altitude en mètres, la fréquence cardiaque en puls/minute et la cadence en pas/minute. Nous exportions alors les données de dénivelé positif et négatif en mètres, de pente positive et négative en pourcentages, de vitesse à plat, ascensionnelle et de descente en mètres/minutes, de distance en mètres et pour finir, de temps en secondes.

2.3.4 Fichiers. Le fichier pour les modèles polynomial et MLP était constitué des données suivantes : dénivelé positif, pente positive, distance et temps, alors que ceux pour le modèle

linéaire contenaient le dénivelé positif et négatif, la vitesse à plat, ascensionnelle et de descente ainsi que la distance. Pour ce modèle, la transformation devait encore s'établir afin d'obtenir finalement comme première colonne : la distance/vitesse à plat (d/Vfloat), la deuxième, le dénivelé positif/vitesse ascensionnelle (D+/VA), la troisième, le dénivelé négatif/vitesse de descente (D-/VD) et finalement la quatrième, le temps.

2.4 Algorithme sur Python

Pour établir les algorithmes, le programme Python fut choisi et employé car il est intuitif et accessible. Ce travail est divisé en deux parties : celle où la prédiction de performance s'établit individuellement à partir des données d'entraînement et l'autre partie correspond à la prédiction de course, auquel certains des sujets prendraient part le 27 avril 2019.

2.4.1 Algorithme individuel. Un code générique, facile à utiliser, devait être construit afin que chaque modèle soit appliqué individuellement. Pour cette méthode, le fichier correspondant à chaque sujet été chargé au début du code et ensuite nous avons sélectionner les 2/3 des données pour les mettre dans le premier vecteur *nTrain*. Cette opération permettait de trouver un lien entre les différentes colonnes de la matrice de base *X*.

$$X= (D + (m), p+ (\%), d (m))$$

Avec :

- *Première colonne : le dénivelé positif (mètre)*
- *Deuxième colonne : la pente positive (pourcent)*
- *Troisième colonne : la distance (mètre)*

Elle correspondait à matrice de taille *mx3* où m est le nombre de données d'entraînement par sujet. Une autre matrice entrait aussi en jeu :

$$Y= (t (s))$$

Avec comme colonne le temps (seconde).

La matrice du temps, *Y*, est de taille *m x 1*.

Par exemple, pour le sujet 4 :

- La matrice *X* est de taille *242 x 3*.
- Le vecteur *nTrain* dans ce cas-là équivalait à : *nTrain= 161*. En résumé, il y avait 161 données pour permettre à l'algorithme de trouver un modèle afin de prédire le temps des 81 données restantes. Il était alors possible de comparer le y prédit et le y calculé pour les 81 derniers entraînements.

Modèle polynomial. Pour chaque modèle, un code différent a été employé pour sortir les yPredicted. Pour le modèle polynomial, le code model.predict était utilisé.

```
polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(XTrain)
model = LinearRegression()
model.fit(x_poly, yTrain)
yPredicted = model.predict(polynomial_features.fit_transform(XTest))
print("yPredicted")
print(yPredicted)
```

Modèle MLP. Pour ce modèle, le code utilisé pour obtenir le yPredicted était mlp.predict.

```
mlp = MLPRegressor(hidden_layer_sizes=500, max_iter=300, shuffle=False, ran-
dom_state=10, solver='adam', activation='relu')
mlp.fit(XTrain, yTrain)
yPredicted = mlp.predict(XTest)
print("yPredicted")
print(yPredicted)
```

Modèle linéaire. Le code employé dans ce modèle pour obtenir la prédiction de temps était model.predict.

```
model = LinearRegression()
model.fit(XTrain, yTrain)
print("coefficient")
print(model.coef_)
yPredicted = model.predict(XTest)
```

Il était alors possible de trouver la moyenne, la médiane et l'écart type absolu ainsi que la moyenne, la médiane et l'écart type relatif des erreurs pour chacun des modèles.

2.4.2 Prédiction course. La course partant de Saillon avec 466 mètres d'altitude et arrivant à Ovronnaz à 1235 mètres a été choisie afin de pouvoir valider ou non, nos prédictions. La course se déroulait le 27 avril 2019 par temps couvert avec une température d'environ 8 degrés. D'une distance de 9 kilomètres avec 840 mètres de dénivelé, Saillon-Ovronnaz est la première course de montagne valaisanne 2019. Elle se divise en 25% de sentiers, 50% de routes carrossables et 25% de routes goudronnées. Le parcours est relativement exigeant que ce soit physiquement ou mentalement. Deux ravitaillements sont à la disposition des coureurs durant leur effort.

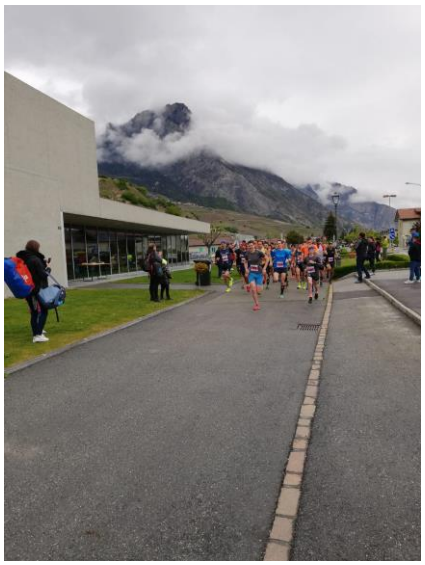


Figure 6. Départ de la course Saillon-Ovronnaz le 27 avril 2019.

Quinze de nos sujets prirent part à la course ce jour-là, portant tous sur leur dossard, l'inscription « Unité M », nom d'équipe donné pour l'occasion reliant « université » et « travail de master ».

Pour la prédiction de course, nous avons ajouté quelques lignes de codes à l'algorithme individuel pour ressortir la performance estimée lors de Saillon-Ovronnaz. Elle était calculée pour chaque sujet qui y participait. Nous avons prédit le temps de course pour les trois modèles : MLP, polynomial et linéaire. Dans notre cas, la matrice *nTrain* possédait toutes les données d'entraînement moins une c'est-à-dire *nTrain= m-1*. Nous devions enlever une donnée afin que les matrices *XTest* et *yTest* ne soient pas des matrices identités. Pour les modèles polynomial et MLP, un nouveau fichier appelé « terrain » était créé avec les données relatives à la course soit : dénivelé positif 840 mètres, pente moyenne 9% et distance 9230 mètres. Pour le modèle linéaire, un nouveau fichier « terrainii » fut établi, où ii correspondait aux initiales de chaque sujet. Il se composait de trois colonnes. La première était la distance de la course divisée par la moyenne de vitesse à plat pour chaque sujet *i* soit d/moyVFloat$_i$ (La moyenne de VFloat étant calculée avec les données d'entraînements), la deuxième colonne était D+/moyVA$_i$ et la troisième D-/moyVD$_i$ (moyVA correspondant à la moyenne de la vitesse ascensionnelle en entraînement et moyVD la moyenne de la vitesse de descente en entraînement). La donnée yCourse-Predicted était ainsi calculée grâce à l'algorithme et correspondait à une donnée temps. La prédiction avait été établie avant le jour j pour chaque coureur.

2.4.3 Travail parallèle de Jérôme Crettaz. Jérôme Crettaz effectuait une recherche parallèle consistant en l'élaboration d'une prédiction grâce au test « trail » de la Clinique romande de réadaptation (CRR). Le test se réalisait en laboratoire en utilisant la machine de spirométrie

ainsi que la prise de lactate. La première étape pour l'évaluation des prédictions fut la transposition de la distance de course, soit 9'230 mètres et son dénivelé 840 mètres en distance équivalente à plat : distance équivalente à plat (km) = distance (km) + dénivelé positif cumulé (m)/100 (Saugy et al., 2013).

$$\text{Distance équivalente à plat} = 9.23 + \frac{840}{100} = 17.63\text{km}$$

D'autres formules existaient également pour trouver la distance à plat ; Jérôme Crettaz de son côté, décida d'arrondir la distance à plat à 15km. Puis, le VO2 max de chaque sujet a été exporté des tests physiques afin de prédire pour chaque sujet un temps de course grâce à la figure ci-dessous.

Tableau	VMA VO2max	10'000m s	15'000m s	20'000m s
12	38	3480	5400	7320
13	42	3180	4980	6780
13.5	44	3060	4800	6540
14	46	2940	4620	6300
14.5	48	2850	4470	
15	50	2760	4320	5880
15.5	52	2670	4185	
16	54	2580	4050	5520
16.5	56	2505	3922.5	
17	58	2430	3795	5160
17.5	60	2365	3687.5	
18	62	2300	3580	4860
18.5	64	2235	3480	
19	66	2170	3380	4590
19.5	68	2105	3280	
20	70	2040	3180	4320
20.5	72	1985	3092.5	
21	74	1930	3005	4080
21.5	76	1880	2920	
22	78	1830	2835	3840
23	82	1740	2700	3660
24	86	1650	2565	3480

Figure 7. Utilisation du tableau de Gindre afin de prédire un temps pour 15km avec le V02max (Gindre, 2013, p.139).

3 Résultats

3.1 Comparaison des régressions pour le modèle individuel

Une comparaison des modèles MLP et polynomial a été effectuée pour chaque sujet. Le y prédit, a été calculé et comparé au y donné. Le y prédit, dépend des deux régressions à disposition : polynomial et MLP. La moyenne, la médiane et l'écart-type des erreurs absolues et relatives ont alors pu être évalués en fonction des régressions disponibles. Les résultats sont présentés dans la figure 8. La méthode fonctionnant le mieux pour chaque sujet dans ce modèle a été répertoriée dans le *tableau 1*.

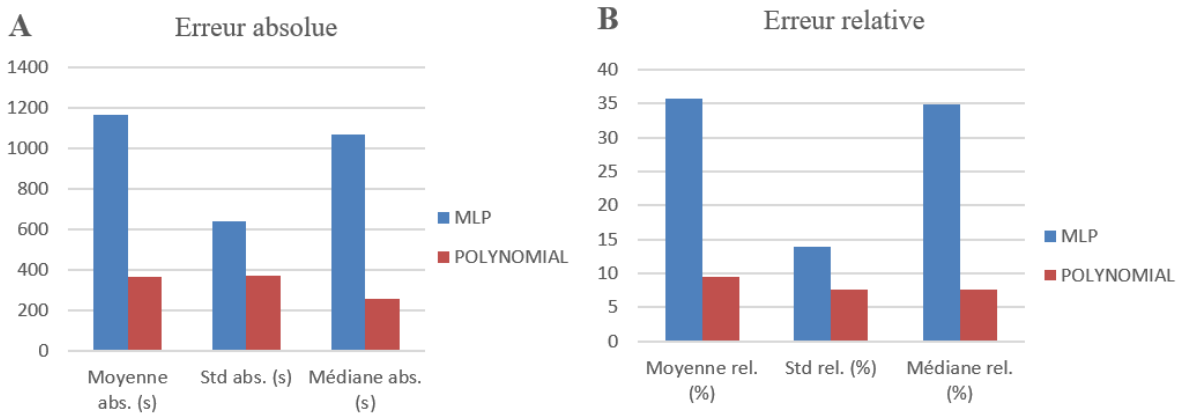


Figure 8. Comparaison des moyennes des erreurs. A: Erreur absolue des régressions polynomiale et MLP en utilisant la méthode individuelle. B: Erreur relative des régressions polynomiale et MLP en utilisant la méthode individuelle.

Tableau 1

Sélection de la méthode optimale pour chaque sujet

	Nombre de sujets	Pourcentage
Polynomial	50	96.15 %
MLP	2	3.85%

3.1.1 Méthode linéaire pour le modèle individuel. Pour chaque sujet, le y prédit, grâce au modèle linéaire a été calculé et comparé au y donné. La moyenne, la médiane et l'écart-type des erreurs absolues et relatives ont alors pu être évalués en fonction de ce modèle. Les résultats sont présentés dans la figure 9.

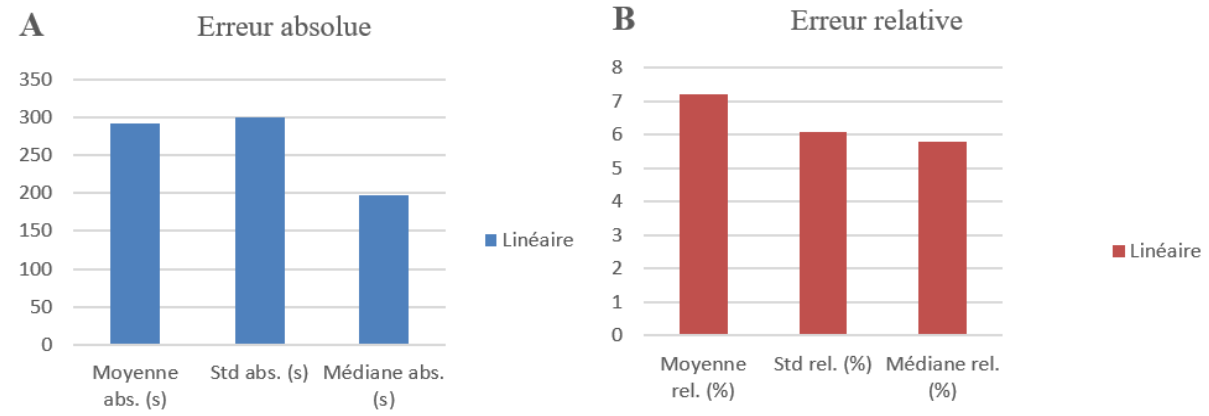


Figure 9. Comparaison des moyennes des erreurs. A: Erreur absolue du modèle linéaire en utilisant la méthode individuelle. B: Erreur relative du modèle linéaire en utilisant la méthode individuelle.

3.2 Comparaison des régressions pour la prédiction de course

3.2.1 Comparaison des modèles MLP et polynomial pour la prédiction de course. Pour chaque sujet, un *y* prédit pour la course, a été calculé puis comparé au temps exécuté durant la course Saillon-Ovronnaz. Les erreurs absolues et relatives ont pu être déterminées et les résultats sont présentés dans la figure 10. Le *tableau 2* rassemble les prédictions établies avec chacun des modèles pour tout sujet. Les erreurs relatives des différents sujets ont également été analysées et triées de manière croissante. Le *tableau 3* résume alors l'ensemble de ces résultats.

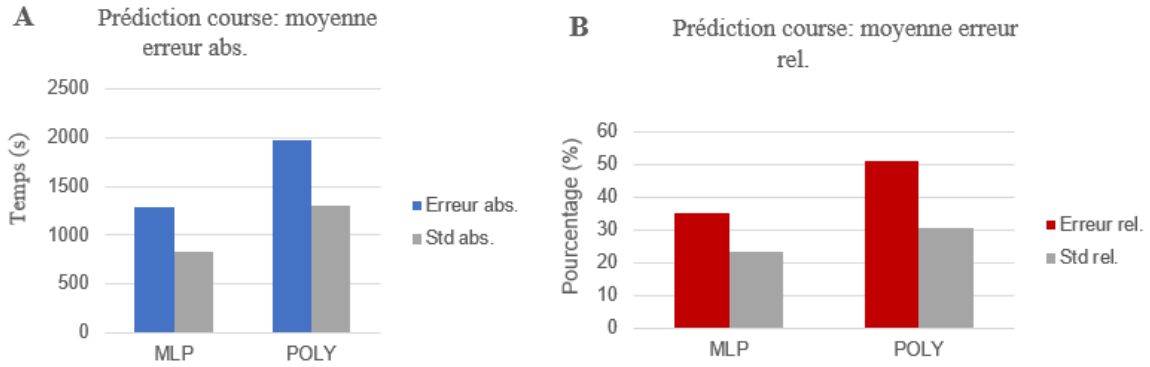


Figure 10. Comparaison des moyennes des erreurs. A: Erreur absolue des régressions MLP et polynomial pour la prédiction de course. B: Erreur relative des régressions MLP et polynomial pour la prédiction de course.

Tableau 2

Comparaison du pourcentage des erreurs relatives lors de la prédiction de course en fonction des sujets

Sujet	Temps fait (s)	Temps prédit MLP (s)	Temps prédit POLY (s)	Erreur relative MLP (%)	Erreur relative POLY (%)
S1	4262	4678	8218	9.79	92.87
S2	3568	4115	4465	15.68	25.54
S3	3874	4184	3337	8.40	13.56
S4	3073	4408	743	43.70	75.78
S5	4569	5363	554	21.46	17.46
S6	3524	5333	3645	52.10	3.95
S7	4641	5729	8351	23.67	80.27
S8	5653	6992	3716	23.81	34.20
S9	4507	7594	9395	68.49	108.45
S10	5559	5530	5962	55.72	67.88
S11	3707	4220	5890	14.43	59.69
S12	3360	4377	1057	41.71	65.78
S13	3064	5259	2836	71.67	7.43
S14	3364	6043	5687	79.64	69.05
S15	3025	3922	4732	29.67	56.44
S16	3445	3549	5000	3.03	43.82

Note. Pour chaque sujet, la méthode dont la prédiction est la meilleure a été mise en évidence.

Tableau 3

Comparaison du pourcentage d'erreurs relatives lors de la prédiction de course en fonction des sujets

Nombres de sujets	Erreur relative (%)	Détails		
		Sujet	Erreur relative (%)	Méthode
2	0-5%	S16	3.03	MLP
		S6	3.95	POLY
3	5-10%	S13	7.43	POLY
		S3	8.40	MLP
		S1	9.79	MLP
1	10-15%	S11	14.43	MLP
2	15-20%	S2	15.68	MLP
		S5	17.46	POLY
2	20-25%	S7	23.67	MLP
		S8	23.81	MLP
3	25-50%	S15	29.67	MLP
		S4	43.70	MLP
		S12	41.71	MLP
3	>50%	S10	55.72	MLP
		S9	68.49	MLP
		S14	69.05	POLY

Note. Pour chaque sujet, seule l'erreur relative de la meilleure méthode est présentée.

Cas 1 où la prédiction a bien fonctionné. S6, coureur de montagne, est né en 1989. 127 données d'entraînements existent pour ce sujet. La moyenne du dénivelé positif aux entraînements se monte à 583.85 mètres, celle de la pente positive de 11.33% et celle de distance de 10'825 mètres. Les temps prédits (i.e. y prédit), avec les deux régressions ainsi que le temps exécuté durant la course, sont représentés dans le *tableau 4*. L'erreur absolue et relative ont pu ainsi être calculée comme le représentent la figure 11. Le modèle individuel de S6 est représenté dans la figure 12.

Tableau 4

Comparaison des temps prédits (i.e. y prédit) des différentes régressions avec le temps établi (i.e. y donné) grâce au résultat de la course Saillon-Ovronnaz pour le sujet S6

Sujet	Temps prédit MLP (s)	Temps prédit Polynomial (s)	Temps course (s)
S6	5333	3645	3524

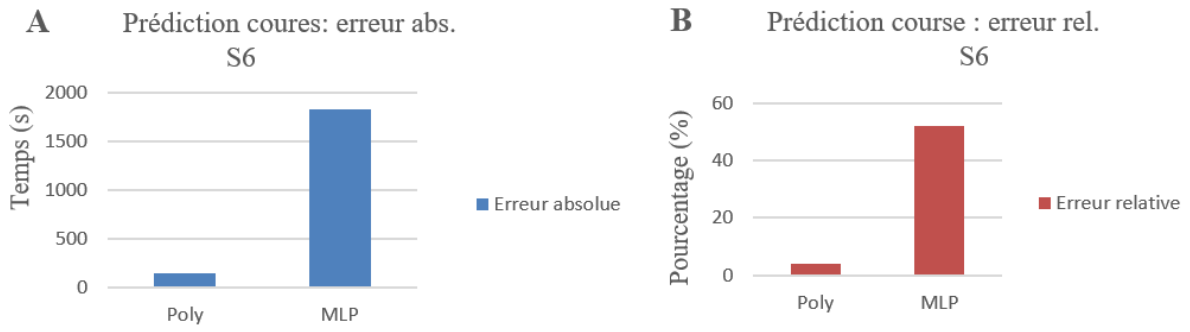


Figure 11. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial pour la prédiction de course pour le sujet S6. B: Erreur relative des régressions MLP et polynomial pour la prédiction de course pour le sujet S6.

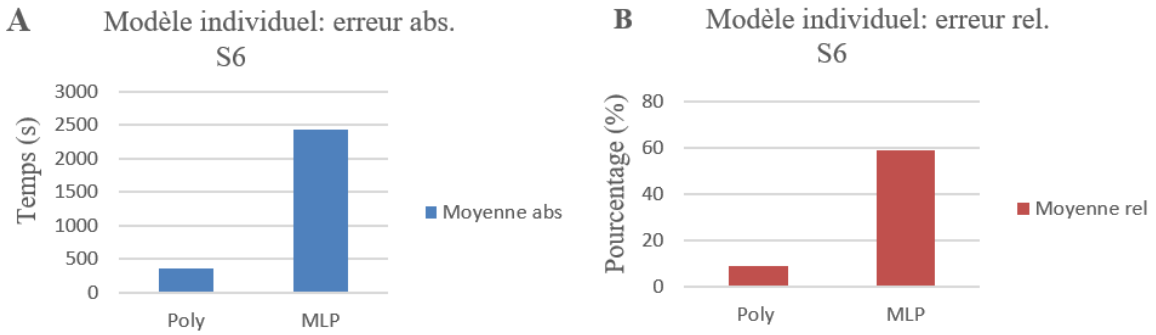


Figure 12. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial par le modèle individuel pour le sujet S6. B: Erreur relative des régressions MLP et polynomial par le modèle individuel pour le sujet S6.

Cas 2 où la prédiction a bien fonctionné. S16 est un coureur de montagne, né en 1971. Nous possédions 55 données correspondant à ses entraînements. En les analysant, nous avons recueilli sa moyenne de dénivelé positif, soit 321.209 mètres, de pente positive, soit 4.4% et la distance, soit 11’500 mètres. La comparaison des temps prédits et du temps établi durant sa course est disponible dans le *tableau 5*. Les erreurs absolues et relatives de S16 ont été calculées

pour les deux régressions MLP et polynomiale. La figure 13 résume alors l'ensemble des résultats. Le modèle individuel du sujet S16 figure dans la figure 14.

Tableau 5

Comparaison des temps prédits (i.e y prédit) des différentes régressions avec le temps établi (i.e. y donné) grâce au résultat de la course Saillon-Ovronnaz pour le sujet S16

Sujet	Temps prédit MLP (s)	Temps prédit Polynomial (s)	Temps course (s)
S16	3549	5000	3445

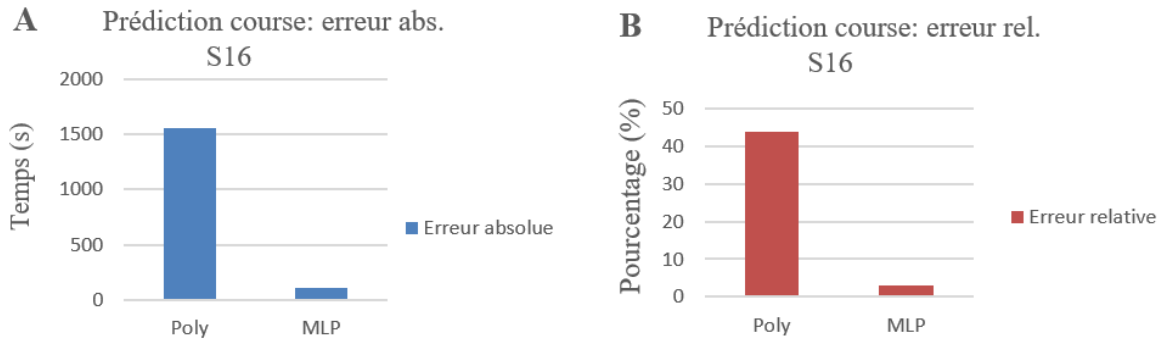


Figure 13. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial pour la prédiction de course pour le sujet S16. B: Erreur relative des régressions MLP et polynomial pour la prédiction de course pour le sujet S16.

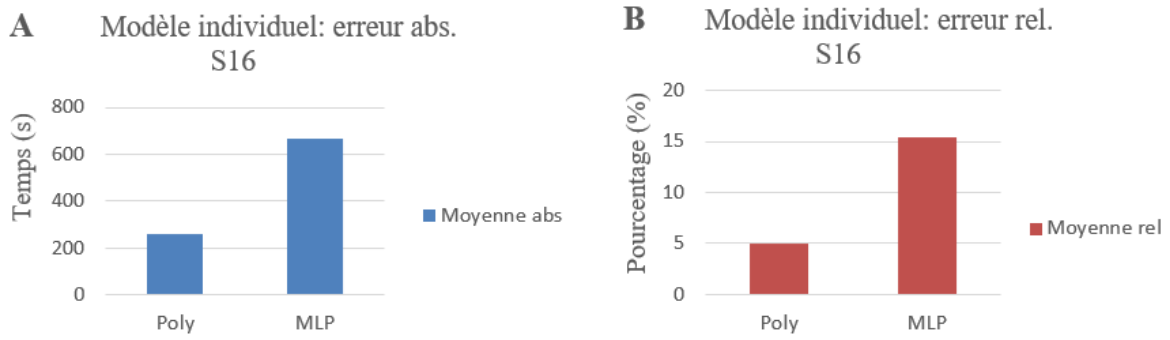


Figure 14. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial par le modèle individuel pour le sujet S16. B: Erreur relative des régressions MLP et polynomial par le modèle individuel pour le sujet S16.

Cas 3 où la prédiction n'est pas concluante. Le sujet S14, né en 1995, est également un coureur de montagne. 168 données d'entraînement ont été mises à notre disposition pour l'analyse avec un dénivelé moyen en entraînement de 472.99 mètres, de pente positive de 10.6% et de distance de 8829 mètres. Comme dans les deux cas précédents, les temps prédits et le temps établi durant la course sont représentés dans le *tableau 6*. Les erreurs absolue et relative de S14 sont calculées et représentées dans la figure 15 ainsi que le modèle individuel dans la figure 16.

Tableau 6

Comparaison des temps prédits (i.e. y prédit) des différentes régressions avec le temps établi (i.e y donné) grâce au résultat de la course Saillon-Ovronnaz pour le sujet S14

Sujet	Temps prédit MLP (s)	Temps prédit Polynomial (s)	Temps course (s)
S14	6043	5687	3364

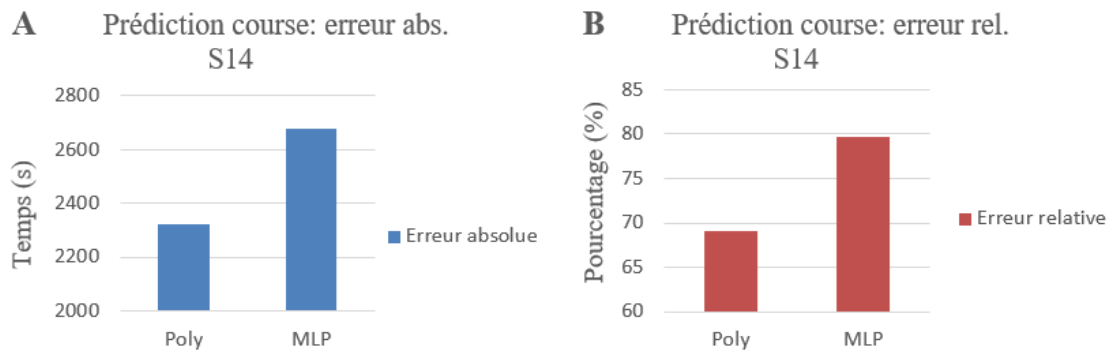
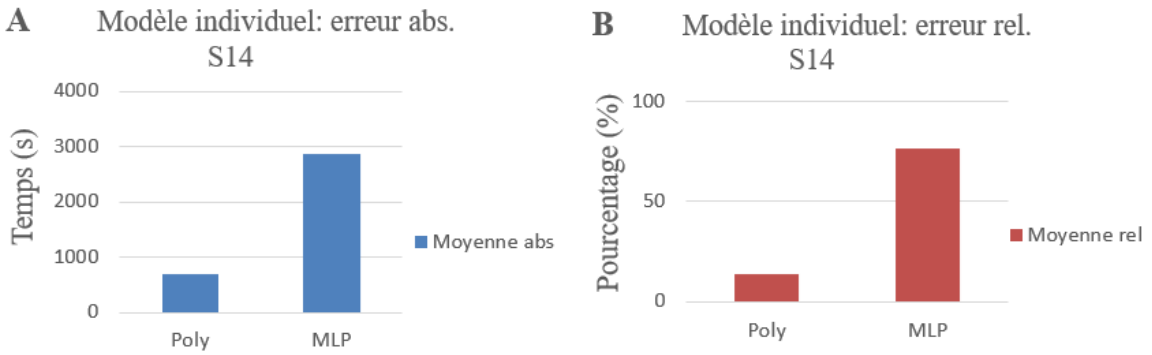


Figure 15. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial pour la prédiction de course pour le sujet S14. B: Erreur relative des régressions MLP et polynomial pour la prédiction de course pour le sujet S14.



Figures 16. Comparaison des erreurs. A: Erreur absolue des régressions MLP et polynomial par le modèle individuel pour le sujet S14. B: Erreur relative des régressions MLP et polynomial par le modèle individuel pour le sujet S14.

3.2.2 Utilisation du modèle linéaire pour la prédiction de course. Le modèle linéaire a été appliqué pour chaque sujet afin de prédire leur temps à la course Saillon-Ovronnaz. La moyenne des erreurs absolue et relative a pu être calculée avec l'algorithme et est présentée dans la figure 17. Pour chaque sujet, le *y* prédit grâce à ce modèle a été comparé au *y* établi durant la course voir *tableau 7*. Le *tableau 8* représente les erreurs relatives triées dans un ordre croissant.

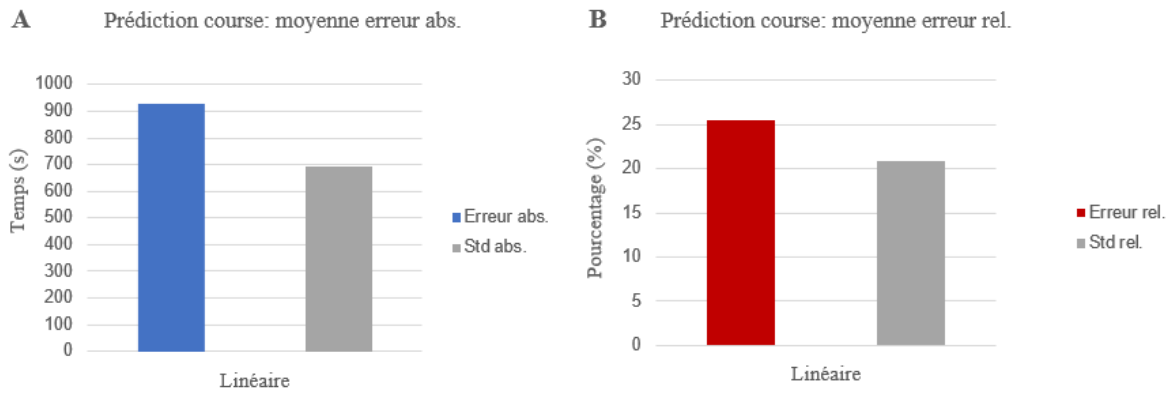


Figure 17. Comparaison des erreurs. A: Erreur absolue du modèle linéaire pour la prédiction de course. B: Erreur relative du modèle linéaire pour la prédiction de course.

Tableau 7

Temps prédit et pourcentage d'erreurs pour chaque sujet pour la prédiction de course avec le modèle linéaire

Sujet	Temps fait (s)	Temps prédit modèle linéaire (s)	Erreur relative modèle linéaire (%)
S1	4262	3139	26.33
S2	3568	3472	2.39
S3	3874	2272	41.15
S4	3073	2920	4.82
S5	4569	3888	14.84
S8	5653	5384	4.69
S9	4507	3646	19.10
S10	5559	4761	34.07
S12	3360	4699	52.15
S13	3064	3565	16.36
S14	3364	5906	75.55
S16	3445	2911	15.48

Note. Pour les sujets possédant une montre de la marque « Suunto », les données pour le modèle linéaire n'ont pas pu être extraites.

Tableau 8

Comparaison du pourcentage des erreurs relatives lors de la prédiction de course en fonction des sujets pour le modèle linéaire

Nombres de sujets	Erreur relative (%)	Détails	
		Sujet	Erreur relative (%)
3	0-5%	S2	2.39
		S8	4.69
		S4	4.82
0	5-10%		
1	10-15%	S5	14.84
3	15-20%	S16	15.48
		S13	16.36
		S9	19.10
0	20-25%		
3	25-50%	S1	26.33
		S10	34.07
		S3	41.15
2	>50%	S12	52.15
		S14	75.55

3.2.3 Prédiction par test physique. Un test physique ont été effectués sur les sujets (travail de master de Jérôme Crettaz, 2019). Les prédictions pour la course Saillon-Ovronnaz ont pu être établies pour chaque sujet et les résultats sont représentés dans le *tableau 9*. Les moyennes absolues et relatives des erreurs sont représentées dans la figure 18.

Tableau 9

Temps prédit et pourcentage d'erreurs pour chaque sujet pour la prédiction en utilisant la méthode « test physique »

Sujet	Temps établi (s)	Temps prédit par test physique (s)	Erreurs relatives test physique (%)
S1	4262	4320	1.36
S2	3568	3795	6.36
S3	3874	3687	4.83
S4	3073	3092	0.62
S5	4569	4620	1.12
S6	3524	3480	1.25
S7	4641	4620	0.45
S8	5653	4800	15.09
S10	5559	3580	35.60
S11	3707	3687	0.54
S14	3364	3380	0.48

Note. Les prédictions ont été établies seulement pour les sujets ayant participé au test physique

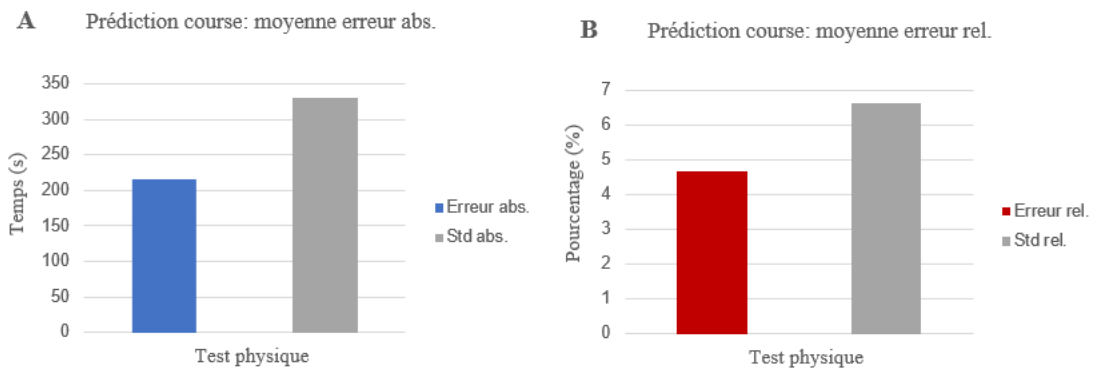


Figure 18. Comparaison des moyennes des erreurs. A: Erreur absolue pour la méthode « test physique ». B: Erreur relative pour la méthode « test physique ».

3.3 Disparité de performance dans les jeux de données

Les résultats présentent des prédictions avec un pourcentage d'erreurs relatives parfois conséquent, qui est expliqué principalement par le bruit causé par la variabilité dans le jeu de données. Dans le *tableau 10* figure un exemple évident de disparité.

Tableau 10

Exemple de disparité du jeu de données pour un sujet avec un jeu de données appartenant à nTrain, les temps réels exécutés en entraînement (i.e. données à prédire) et les temps prédits

Jeu de données d'apprentissage (s)	1077.8	949.8	12.882	-11.416	11757.8	11611
Données à prédire (s)	1028.2	1022	13.1263	-4.823	10806.7	6447
Données prédites (s)						11900

4 Discussion

Les résultats ont démontré des prédictions plus ou moins bonnes pour les sujets. Beaucoup de méthodes sont développées pour des courses d'endurance telles que les 10 kilomètres ou les marathons. Elles restent cependant incertaines pour les courses de montagne et donc les résultats concluants sur de tels parcours sont intéressants et nécessitent une attention particulière.

4.1 Les données d'entraînement permettent-elles de pouvoir prédire une performance en course de montagne ?

Les prédictions d'une performance en course de montagne peuvent être effectuées grâce aux méthodes réalisées avec les modèles polynomiale, MLP et linéaire. Cependant elles restent très sensibles au bruit.

4.1.1 Modèle individuel. L'erreur moyenne absolue du modèle polynomial était de 363.32 secondes et pour le modèle MLP de 1165.31 secondes. Pour le modèle linéaire, avec une base de données différentes, l'erreur absolue se montait à 293.01 secondes. Pour la moyenne des erreurs relatives, on trouvait 9.45% pour la régression polynomiale, 35,77% pour le modèle MLP et 7.22% pour le modèle linéaire. Nous constatons qu'il est alors possible de prédire une performance en course de montagne avec ces trois modèles et plus particulièrement avec le modèle linéaire ou polynomial qui nous donnent un taux plus faible d'erreur. Chez la majorité des sujets, 96.15%, le modèle polynomial donne de meilleure prédiction que la méthode MLP. Ce qui peut nous faire penser que la méthode MLP est plus sensible au bruit et possède donc une plus grande variabilité. Le modèle linéaire octroie des résultats avec une moyenne d'erreurs absolues de 293.07 secondes, soit 4.88 minutes ce qui le place dans les meilleurs modèles testés. Cependant, comme mentionné précédemment, les variables explicatives et cibles ne possèdent pas forcément une relation linéaire ce qui rend son utilisation compliquée.

4.1.2 Prédiction de la course. Les moyennes des erreurs absolues et relatives pour la prédiction ont été calculées et se montent à 2817.51 secondes pour le modèle polynomial contre 1290.65 secondes pour le modèle MLP. Il existe déjà, à ce niveau-là, une différence avec la méthode individuelle. Ici, le modèle MLP est plus performant dans la prédiction pour la course Saillon-Ovronnaz avec une moyenne d'erreurs relatives de 35.18% contre 51.39 % pour le modèle polynomial. Si l'on observe les six personnes qui possèdent moins de 15% d'erreurs relatives, le modèle MLP est plus performant chez quatre d'entre elles. Mais il est important de

remarquer que pour chacun des modèles, nous trouvions un sujet chez qui la prédiction était meilleure.

Sujet 6. Pour le sujet 6, la méthode polynomiale montre des résultats plus précis (erreur absolue 138.66 secondes et erreur relative 3.95%) que le modèle MLP (erreur absolue 1827 secondes et erreur relative 52.1%). C'est également le cas pour sa méthode individuelle où là aussi, le modèle polynomial sortait des prédictions plus précises.

Sujet 16. Pour S16, il est intéressant d'observer que le modèle MLP, contrairement au sujet précédent, donne une meilleure prédiction pour la course (erreur absolue 104.26 secondes et erreur relative 3.03%). De manière générale, on peut constater que la méthode MLP fonctionnait le moins au vu des pourcentages d'erreurs. S16 est un coureur de montagne qui s'entraîne en majorité sur des terrains similaires à la course Saillon-Ovronnaz, ce qui pourrait expliquer le taux relativement bas d'erreur relative soit 3.03% pour le modèle MLP contre 43.82% pour le modèle polynomial. Nous pouvons relever peu de variabilité dans les données d'entraînements et pratiquement aucune valeur aberrante ce qui engendre un faible bruit.

Chez les sujets possédant moins de 10% d'erreur, soit S16 et S6, il serait intéressant de connaître s'ils avaient le même profil et la même fréquence d'entraînement, le même âge afin de comprendre le faible taux d'erreur.

Sujet 14. Les résultats de S14 nous ont grandement étonné, comme S16, est un sujet s'entrainant majoritairement en présence de dénivelé et de pente. Nous nous attendions à de bonnes prédictions alors que les erreurs relatives des trois modèles sont conséquentes (69.05% polynomial, 79.64 MLP et 75.5% linéaire). Pour le modèle linéaire, seules 11 données étaient disponibles ; ce nombre peu conséquent pourrait expliquer ce taux d'erreur. Pour les deux autres modèles, le jeu était constitué de 168 données. La présence de valeurs absurdes est possible et pourrait fausser les prédictions surtout si elles se trouvent dans la partie sélectionnée pour le nTrain. Il faudrait analyser entraînement par entraînement afin de les retrouver et donc réduire le bruit pour obtenir de meilleures prédictions.

Bruit. Le nombre de données pouvait donc biaiser les prédictions et de nombreux paramètres augmentaient également le bruit comme par exemple si la course se déroulait en début de saison avec des sujets pas encore bien entraînés, une météo pluvieuse du jour, une température peu élevée ou encore si certains sujets s'entrainaient généralement sur du plat.

Test physique. En utilisant les prédictions par les tests physiques, nous pouvons observer des erreurs relatives ((valeur prédite – valeur effectuée) / valeur effectuée) majoritairement faibles (moyenne erreur absolue 315.91 secondes et moyenne erreur relative 6.15%). Neuf sujets sur onze possèdent une erreur relative plus petite que 10% ce qui démontre la puissance de cette

méthode de prédiction. Les sujets 10 et 8 détiennent les moins bonnes prédictions dans les tests physiques (erreur relative de S10, 35.60 et celle de S8, 15.09%). Dans les autres modèles, les sujets 10 et 8 obtenaient des erreurs relatives assez élevées à l'exception de S8 avec le modèle linéaire (S8 : linéaire 4.69%, MLP 23.81%, polynomial 34.20% et S10 : linéaire 34.07%, MLP 55.72%, polynomial 67.88%).

S8 est un coureur occasionnel mais très motivé. Sa prédiction aux tests physiques pourrait être influencée s'il s'entraîne occasionnellement mais pas de manière intensive. Il n'est donc pas habitué à aller dans ses retranchements et courir jusqu'à épuisement.

Le sujet 10, au contraire, grand habitué des courses de montagne s'entraîne quotidiennement. Ses grandes erreurs relatives dans les trois modèles et dans les tests physiques posent de nombreuses questions. 81 données étaient mises à notre disposition pour l'analyse et sa moyenne de dénivelé positif en entraînement de 706.19 mètres, de 7.8% de pente et 10'444.80 mètres de distance laissaient penser que l'on obtiendrait de bonnes prédictions grâce aux trois algorithmes. Lors du test physique, le sujet a couru jusqu'à épuisement avec toute son énergie. Il est donc très étrange d'obtenir de tels résultats.

Pour les algorithmes, quelques données pouvaient être moins cohérentes et ont pu biaiser nos prédictions. Le test physique donne de meilleurs résultats mais est moins pratique que l'analyse des données d'entraînements car la prise de temps est assez conséquente pour l'effectuer. Il existe différents bruits dans l'exécution du test physique comme la forme physique du jour du test du coureur, l'inaccoutumance au tapis de course ou encore la pression d'effectuer de tels tests. Il est également difficile de comparer les prédictions obtenues durant les tests physiques avec celles acquises grâce aux algorithmes d'apprentissage automatique. En effet, la majorité des coureurs présents au test était constitué d'excellents coureurs de montagne alors que mon panel de sujets était très diversifié. Il aurait été préférable de tester en laboratoire, un échantillon varié de sujets, autant avec de très bons coureurs que des coureurs occasionnels. Le panel des tests physiques était de neuf personnes représentant un échantillon relativement faible. Les résultats cependant concluants peuvent être repris et améliorés en augmentant le nombre et les niveaux des sujets présents aux tests physiques. Cependant, il existe un lien étroit entre le test physique et les modèles de régression. En effet, pour l'obtention de prédictions précises, il est possible d'associer les tests physiques et les modèles de régression en mettant en relation par exemple : la V02max et la VA obtenue lors des tests avec la régression polynomiale ou linéaire.

4.2 Comment évaluer et améliorer les prédictions ?

L'apprentissage automatique permet d'évaluer les prédictions directement à partir des entraînements sélectionnés aléatoirement dans le jeu de données de départ. En effet, la phase dite de test, évalue et présente les erreurs obtenues entre les valeurs prédites et celles réelles établies en entraînement. Un nouveau moyen supplémentaire a donc permis d'évaluer les prédictions grâce à la course Saillon-Ovronnaz. Ces dernières ont pu être établies au préalable puis comparées au temps réalisé le jour de la course. Nous obtenons donc plus de données dans la phase d'apprentissage ce qui augmente les précisions des prédictions.

Plusieurs facteurs peuvent améliorer les résultats afin d'obtenir de meilleures prédictions. Le problème majeur provient du bruit des données. En effet, un sujet qui effectue un parcours uniquement à la course ou en marchant va influencer les données et donc augmenter leur variabilité. Par exemple, dans le cas présenté dans le *tableau l0*, l'erreur de prédiction est dû à la donnée non pertinente dans le jeu de données d'apprentissage (i.e. 11'611 secondes) qui induit une très mauvaise prédiction du temps (i.e. 11'900 secondes au lieu de 6'447 secondes soit une erreur absolue de 5'453 secondes). Les données devraient être nettoyées et triées attentivement afin d'obtenir un jeu de données sans valeurs absurdes et en éliminant ce type de problème. D'autres problèmes subviennent également comme le nombre d'entraînements, le profil des entraînements vs le profil de la course sélectionnée ou encore le nombre de paramètres pris en compte dans la matrice X. Un plus grand nombre d'entraînements pour obtenir un jeu de données toujours plus élevé entraîne par conséquent une plus grande phase d'apprentissage pour autant que les valeurs soient cohérentes entre elles. Idéalement les sujets devraient s'entraîner uniquement avec du dénivelé et se rapprocher le plus, du profil de leurs entraînements avec celui de la course. Une autre manière d'améliorer les prédictions serait de les tester sur d'autres courses mais à différentes périodes de la saison afin d'éliminer tous facteurs parasites comme la fatigue, les paramètres météorologiques, la motivation… Le nombre de paramètres pris en compte dans la matrice X peut aussi être augmenté afin de tenir compte de tous les facteurs disponibles lors d'une course comme par exemple le relief du terrain. D'autre part, nous avons pris en considération peu de paramètres au regard de l'ensemble des composantes des courses de montagne comme l'influence de la météo, du type de terrain, du dénivelé négatif, de la forme actuelle de l'athlète, des difficultés techniques… Il serait alors possible d'agrandir la matrice X en lui rajoutant des colonnes pour ces nouveaux paramètres et d'effectuer de nouvelles prédictions afin de constater si elles se précisent davantage.

4.3 Forces, limitations et perspectives

L'échantillon constitue une des forces principales de ce travail de master. En effet 52 sujets ont participé à l'étude ce qui équivaut à un panel conséquent. De plus, la variété des sujets représente aussi un élément très positif. Une partie des coureurs sont très entraînés avec plus de quatre entraînements par semaine, d'autres le sont moins, avec une ou deux fois. Et enfin certains sujets comme les triathlètes s'entraînent dans d'autres disciplines et courent occasionnellement. Cependant, ils courent tous, au moins depuis une année et ils ont donc tous une certaine expérience dans ce sport.

Les limites de ce travail peuvent expliquer la différence de nos prédictions. En effet, nous constatons que le jeu de données possédait une grande variabilité. Après avoir analysé les résultats obtenus avec les trois méthodes (MLP, polynomial et linéaire), nous remarquons que la filtration et l'exportation doivent être exécutés de manière plus précise et attentive pour obtenir des temps prédits plus proches les temps établis. Le nettoyage des données permettrait de recueillir des valeurs plus pertinentes. Une limite supplémentaire émane du fait que nous n'avons pas sélectionné les sujets en fonction du nombre de données d'entraînement enregistré par leur montre. Certaines personnes possédaient seulement 9 entraînements et d'autres plus d'une centaine. Obtenir des prédictions fiables devenait compliqué avec un sujet avec un faible nombre d'entraînement car il pouvait être fatigué ou en moins grande forme plutôt qu'avec un avec 150 entraînements. Il aurait fallu sélectionner seulement les sujets qui possédaient un nombre minimal d'entraînement pour obtenir des prédictions plus précises. Cependant, nous ne sommes pas en mesure actuellement de définir ce nombre afin d'obtenir des résultats concluants. Comme mentionné plus haut, les variables explicatives ne sont pas assez représentatives pour une course de montagne, il faudrait donc augmenter le nombre de colonnes de celle-ci.

Il subsiste de nombreuses ombres concernant ces prédictions et il serait intéressant de trouver un moyen de prédire ce genre de performance. D'autres algorithmes d'apprentissage automatique seraient susceptibles de donner de meilleures prédictions. Plusieurs modèles devraient être testés afin de trouver un temps prédit plus proche de la réalité. Pour de futures études, une attention plus importante sur le nettoyage des données devrait être consacrée afin d'améliorer encore plus les prédictions. L'intelligence artificielle se développe actuellement et devrait être bénéfique pour la course à pied. L'avantage d'utiliser l'apprentissage automatique se résume au gain de temps d'énergie et financier (coût de Fr. 275.- pour un test VO2 et lactate à la Suva à Sion) pour les coureurs en s'économisant d'effectuer des tests dans un laboratoire.

5 Conclusion

Ce travail s'est intéressé à la prédiction de performances en course de montagne en utilisant des algorithmes issus de l'apprentissage automatique. En premier lieu, le modèle individuel donnait des erreurs relatives pour les différents algorithmes, soit 9.45% pour le modèle polynomial, 35.74% pour le modèle MLP et 7.22% pour le modèle linéaire. D'autre part, l'évaluation des prédictions avec la course Saillon-Ovronnaz du 27 avril 2019, a démontré des résultats divisés. D'excellentes prédictions sont présentées pour certains sujets comme S6 qui possédait une erreur absolue de 365.18 secondes avec le modèle polynomial et le sujet 16 qui, de son côté, obtenait une erreur absolue de 104.26 secondes avec le modèle MLP. Certains sujets présentaient des temps prédits trop élevés par rapport à leur performance de course et des erreurs relatives au-dessus de 50%., comme le sujet 14, pourtant grand coureur de montagne. Le modèle linéaire fonctionnait également pour certains sujets mais pas pour la totalité. En méthode individuel, le modèle linéaire amenait seulement 7.22% d'erreurs relatives ce qui laissait penser que les prédictions pour la course seraient précises pour la majorité des sujets. La variabilité des algorithmes était à souligner. Les prédictions de course au moyen des tests physiques ont présenté de très bons résultats, soit 9 sujets sur 11 possédaient une erreur relative en dessous de 10% et 8 en dessous de 5%. Les tests restent néanmoins contraignants au niveau du temps et coûteux. L'ensemble de ces résultats laissait donc à penser que la prédiction de course de montagne pouvait être élaborée grâce à des algorithmes cependant elle reste encore plus fiable en pratiquant des tests physiques. En définitive, la prédiction en course de montagne engendre l'analyse et la prise en compte de nombreux paramètres supplémentaires tels que le relief du terrain, le dénivelé positif et négatif ou encore le pourcentage de pente dans les algorithmes afin d'obtenir à l'avenir des prédictions plus fiables et plus précises.

Bibliographie

Azencott, C.-A. (2018). *Introduction au Machine Learning*. Saint-Just-la-Pendue : Imprimerie CHIRAT.

Gindre, C. (2013). *Courir en harmonie.* Chavéria: Volodalen.

Haton, J.-P. (2000). *L'intelligence artificielle* [Texte de la 263e conférence de l'Université de tous les savoirs donnée le 19 septembre 2000]. Accès à l'adresse https://streaming-canal-u.fmsh.fr/vod/media/canalu/documents/utls/190900.pdf

Lemberger, P., Batty, M., Morel, M., & Raffaëlli, J.-L. (2017). *Big Data et Machine Learning, Les concepts et les outils de la data science (*2ème édition). Saint-Just-la-Pendue : Imprimerie CHIRAT.

Müller, A.C. & Guido, S. (2017). *Introduction to Machine Learning with Python, a guide for data scientists.* Sebastopol: O'Reilly.

Ng, A. (2019). *AI For Everyone.* Accès à l'adresse https://www.coursera.org/learn/ai-for-everyone

Russel, S. & Norvig, P. (2010). *Intelligence artificielle* (3ème édition). Paris: Pearson Education France.

Saugy, J., Place, N., Millet, G.Y., Degache, F., Schena, F., & Millet, G. (2013). Alterations of Neuromuscular Function after the World's Most Challenging Mountain Ultra-Marathon. Plos One, n°8(6). doi: 10.1371/journal.pone.0065596.

Saunders, P.U., Pyne, D. B., Telford, R.D. & Hawley J.A. (2004). Factors Affecting Running Economy in Trained Distance Runners. *Sports Medicine, volume* (34), 465-485.

Annexe

Flyer de recrutement

Code méthode individuelle

```
from sklearn.datasets import make_friedman2
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel, Exponentiation,
RBF, ConstantKernel, Matern, RationalQuadratic, ExpSineSquared
import numpy as np
from sklearn import preprocessing
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as pl
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import csv
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.externals.six.moves import cStringIO as StringIO
from sklearn.metrics import roc_auc_score
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy.sparse import csr_matrix
from sklearn.utils.testing import (assert_raises, assert_greater, assert_equal,
```

```
assert_false)

# ---------------------------------------------------------
# Load dataset separator = ","
# ---------------------------------------------------------
_numXDim_ = 3

filenames = ['doc.csv']
filename = filenames[0]
x = list(csv.reader(open(filename, 'rt'), delimiter=';', quoting=csv.QUOTE_NONE))
data = numpy.array(x).astype('float')

y = data[:, _numXDim_]
X = data[:, 0:_numXDim_]
# ---------------------------------------------------------
# Enlever valeurs extrêmes
# ---------------------------------------------------------
numLine = len(X)
print(numLine)
newX = [np.array(X[0])]
newy = numpy.zeros(0)
newy = np.array(y[0])
for i in range(1, numLine):
    Xtraininterm = np.delete(X, i, axis=0)
    ytraininterm = np.delete(y, i, axis=0)

    polynomial_features = PolynomialFeatures(degree=2)
    x_poly = polynomial_features.fit_transform(Xtraininterm)
    model = LinearRegression()
    model.fit(x_poly, ytraininterm)
    yPredictedinter = model.predict(polynomial_features.fit_transform(numpy.array([X[i]])))

    if y[i] / yPredictedinter > 0.8 and y[i] / yPredictedinter < 1.2 :
        newy = np.append(newy, y[i])
        newX = np.append(newX, np.array([X[i]]), axis=0)

y = newy
X = newX
print("new Dim")
print(len(X))
# ---------------------------------------------------------
# Sélection des échantillons test et train
# ---------------------------------------------------------
nsTrain = 0;
nTrain = 27;
yTrain = y[nsTrain:nTrain]
print(yTrain)
yTest = y[nTrain:len(X)]
print(yTest)
XTrain = X[nsTrain:nTrain, :]
```

```
XTest = X[nTrain:len(X), :]

for i in range(1, len(filenames)):
    filename = filenames[i]
    x = list(csv.reader(open(filename, 'rt'), delimiter=';', quoting=csv.QUOTE_NONE))
    data = numpy.array(x).astype('float')
    print(data.shape)

    y_ = data[:, _numXDim_]
    X_ = data[:, 0:_numXDim_]

# ---------------------------------------------------------
# Enlever valeurs extrêmes
# ---------------------------------------------------------
numLine = len(X_)
newX = [np.array(X_[0])]
newy = numpy.zeros(0)
newy = np.array(y_[0])
for i in range(1, numLine):
        Xtraininterm = np.delete(X_, i, axis=0)
        ytraininterm = np.delete(y_, i, axis=0)

        polynomial_features = PolynomialFeatures(degree=2)
        x_poly = polynomial_features.fit_transform(Xtraininterm)
        model = LinearRegression()
        model.fit(x_poly, ytraininterm)
        yPredictedinter = model.predict(polynomial_features.fit_transform(numpy.a
ray([X_[i]])))

        if y_[i] / yPredictedinter > 0.8 and y_[i] / yPredictedinter < 1.2:
            newy = np.append(newy, y_[i])
            newX = np.append(newX, np.array([X_[i]]), axis=0)

y_ = newy
X_ = newX
print("new Dim")
print(len(X))

# ---------------------------------------------------------
# Sélection des échantillons test et train
# ---------------------------------------------------------
    y = np.concatenate((y, y_), axis=0)
    print(y.shape)

    X = np.concatenate((X, X_), axis=0)

    yTrain = np.concatenate((yTrain, y_[nsTrain:nTrain]), axis=0)
    yTest = np.concatenate((yTest, y_[nTrain:len(X)]), axis=0)
    XTrain = np.concatenate((XTrain, X_[nsTrain:nTrain, :]), axis=0)
    XTest = np.concatenate((XTest, X_[nTrain:len(X), :]), axis=0)
```

```
# ----------------------------------------------------------
# PLOT 3D point cloud
# ----------------------------------------------------------
a = np.array([1,2,3])
a = np.concatenate((a, [4]))
print(a)

# ----------------------------------------------------------
# Sélection du modèle
# ----------------------------------------------------------
#----------------------------------------------------------
# Modèle polynomial
# ----------------------------------------------------------
polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(XTrain)
model = LinearRegression()
model.fit(x_poly, yTrain)
yPredicted = model.predict(polynomial_features.fit_transform(XTest))
print("yPredicted")
print(yPredicted)

#----------------------------------------------------------
# Modèle MLP
# ----------------------------------------------------------
# mlp = MLPRegressor(hidden_layer_sizes=500,
#                    max_iter=300, shuffle=False, random_state=10, solver='adam',
#                    activation='relu')
# mlp.fit(XTrain, yTrain)
# yPredicted = mlp.predict(XTest)
# print("yPredicted")
# print(yPredicted)

# ----------------------------------------------------------
# Modèle linéaire
# ----------------------------------------------------------
# model = LinearRegression()
# model.fit(XTrain, yTrain)
#
# print("coefficient")
# print(model.coef_)
# yPredicted = model.predict(XTest)



# ----------------------------------------------------------
# Check prediction
# ----------------------------------------------------------
print(XTest);
```

```
diff = np.abs(yPredicted - yTest)
print("yTest");
print(yTest);
print("yPredicted");
print(yPredicted);
print("diff :")
print(yPredicted - yTest);
print("%");
print((diff / yTest * 100).astype(int));
print("Mean: " + str(np.mean(diff)) + ", std: " + str(np.std(diff))+ ", Median: " +
str(np.median(diff)))
print("Mean %: " + str(np.mean((diff / yTest * 100).astype(int))) + ", std %: " +
str(np.std((diff / yTest * 100).astype(int)))+ ", Median %: " + str(np.median((diff / yTest *
100).astype(int))))
resulta = np.column_stack((yTest, diff))

res = diff
yTestSorted = sorted(yTest)
sort_index = numpy.argsort(yTest)

for ida, a in enumerate(sort_index):
    res[ida] = diff[a]

# -------------------------------------------------------
# PLOT
# -------------------------------------------------------
pl.figure(2)
pl.clf()
pl.plot(resulta[:, 0], resulta[:, 1], 'r+', ms=20)
pl.show()
```

Code pour la prédiction de course

```
from sklearn.datasets import make_friedman2
from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import DotProduct, WhiteKernel, Exponentiation,
RBF, ConstantKernel, Matern, RationalQuadratic, ExpSineSquared
import numpy as np
from sklearn import preprocessing
from matplotlib import cm
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as pl
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import csv
import numpy
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import PolynomialFeatures
from sklearn.externals.six.moves import cStringIO as StringIO
from sklearn.metrics import roc_auc_score
```

```
from sklearn.neural_network import MLPClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.preprocessing import LabelBinarizer
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from scipy.sparse import csr_matrix
from sklearn.utils.testing import (assert_raises, assert_greater, assert_equal,
assert_false)

# ---------------------------------------------------------
# Load dataset separator = ","
# ---------------------------------------------------------
numXDim_ = 3
filenames = [doc.csv']
filename = filenames[0]
x = list(csv.reader(open(filename, 'rt'), delimiter=';', quoting=csv.QUOTE_NONE))
data = numpy.array(x).astype('float')
y = data[:, _numXDim_]
X = data[:, 0:_numXDim_]

# ---------------------------------------------------------
# Enlever valeurs extrêmes
# ---------------------------------------------------------
numLine = len(X)
print(numLine)
newX = [np.array(X[0])]
newy = numpy.zeros(0)
newy = np.array(y[0])
for i in range(1, numLine):
    Xtraininterm = np.delete(X, i, axis=0)
    ytraininterm = np.delete(y, i, axis=0)

    polynomial_features = PolynomialFeatures(degree=2)
    x_poly = polynomial_features.fit_transform(Xtraininterm)
    model = LinearRegression()
    model.fit(x_poly, ytraininterm)
    yPredictedinter = model.predict(polynomial_features.fit_transform(numpy.array([X[i]])))

    if y[i] / yPredictedinter > 0.8 and y[i] / yPredictedinter < 1.2 :
        newy = np.append(newy, y[i])
        newX = np.append(newX, np.array([X[i]]), axis=0)
y = newy
X = newX
print("new Dim")
print(len(X))

# ---------------------------------------------------------
# Sélection des échantillons test et train
# ---------------------------------------------------------
nsTrain = 0;
nTrain = 64;
```

```
yTrain = y[nsTrain:nTrain]
print(yTrain)
yTest = y[nTrain:len(X)]
print(yTest)
XTrain = X[nsTrain:nTrain, :]
XTest = X[nTrain:len(X), :]


for i in range(1, len(filenames)):
    filename = filenames[i]
    x = list(csv.reader(open(filename, 'rt'), delimiter=';', quoting=csv.QUOTE_NONE))
    data = numpy.array(x).astype('float')
    print(data.shape)

    y_ = data[:, _numXDim_]
    X_ = data[:, 0:_numXDim_]

# ---------------------------------------------------------
# Enlever valeurs extrêmes
# ---------------------------------------------------------
numLine = len(X_)
newX = [np.array(X_[0])]
newy = numpy.zeros(0)
newy = np.array(y_[0])
for i in range(1, numLine):
        Xtraininterm = np.delete(X_, i, axis=0)
        ytraininterm = np.delete(y_, i, axis=0)

        polynomial_features = PolynomialFeatures(degree=2)
        x_poly = polynomial_features.fit_transform(Xtraininterm)
        model = LinearRegression()
        model.fit(x_poly, ytraininterm)
        yPredictedinter = model.predict(polynomial_features.fit_transform(numpy.ar-
ray([X_[i]])))

        if y_[i] / yPredictedinter > 0.8 and y_[i] / yPredictedinter < 1.2:
            newy = np.append(newy, y_[i])
            newX = np.append(newX, np.array([X_[i]]), axis=0)
y_ = newy
X_ = newX
print("new Dim")
print(len(X))

# ---------------------------------------------------------
# Sélection des échantillons test et train
# ---------------------------------------------------------
y = np.concatenate((y, y_), axis=0)
print(y.shape)

X = np.concatenate((X, X_), axis=0)
```

```
yTrain = np.concatenate((yTrain, y_[nsTrain:nTrain]), axis=0)
yTest = np.concatenate((yTest, y_[nTrain:len(X)]), axis=0)
XTrain = np.concatenate((XTrain, X_[nsTrain:nTrain, :]), axis=0)
XTest = np.concatenate((XTest, X_[nTrain:len(X), :]), axis=0)


# ---------------------------------------------------------
# CHARGEMENT DES PARAMETRE DE TERRAIN
# ---------------------------------------------------------
t = list(csv.reader(open('terrain.csv', 'rt'), delimiter=';', quoting=csv.QUOTE_NONE))
dataCourseTest = numpy.array(t).astype('float')
courseTest = dataCourseTest[:, 0:_numXDim_]

# ---------------------------------------------------------
# PLOT 3D point cloud
# ---------------------------------------------------------
a = np.array([1,2,3])
a = np.concatenate((a, [4]))
print(a)

# ---------------------------------------------------------
# Sélection du modèle
# ---------------------------------------------------------
# ---------------------------------------------------------
# Modèle polynomial
# ---------------------------------------------------------
polynomial_features= PolynomialFeatures(degree=2)
x_poly = polynomial_features.fit_transform(XTrain)
model = LinearRegression()
model.fit (x_poly, yTrain)
yPredicted = model.predict(polynomial_features.fit_transform(XTest))

yCoursePredicted = model.predict(polynomial_features.fit_transform(courseTest))
print ("yCoursePredicted : " + str(yCoursePredicted) )

# ---------------------------------------------------------
# Modèle linéaire
# ---------------------------------------------------------
# model = LinearRegression()
# model.fit(XTrain, yTrain)
#
# print("coefficient")
# print(model.coef_)
# yPredicted = model.predict(XTest)
#
# yCoursePredicted = model.predict(courseTest)
# print ("yCoursePredicted : " + str(yCoursePredicted) )

#---------------------------------------------------------
```

```
# Modèle MLP
# ----------------------------------------------------------
# mlp = MLPRegressor(hidden_layer_sizes=500,
#                    max_iter=300, shuffle=False, random_state=10, solver='adam',
#                    activation='relu')
# mlp.fit(XTrain, yTrain)
# yPredicted = mlp.predict(XTest)
# yCoursePredicted = mlp.predict(courseTest)
# print ("yCoursePredicted : " + str(yCoursePredicted) )

# ----------------------------------------------------------
# check prediction
# ----------------------------------------------------------
print(XTest);
diff = np.abs(yPredicted - yTest)
print("yTest");
print(yTest);
print("yPredicted");
print(yPredicted);
print("diff :")
print(yPredicted - yTest);
print("%");
print((diff / yTest * 100).astype(int));
print("Mean: " + str(np.mean(diff)) + ", std: " + str(np.std(diff))+ ", Median: " +
str(np.median(diff)))
print("Mean %: " + str(np.mean((diff / yTest * 100).astype(int))) + ", std %: " +
str(np.std((diff / yTest * 100).astype(int)))+ ", Median %: " + str(np.median((diff / yTest *
100).astype(int))))
resulta = np.column_stack((yTest, diff))

res = diff
yTestSorted = sorted(yTest)
sort_index = numpy.argsort(yTest)

for ida, a in enumerate(sort_index):
    res[ida] = diff[a]

# ----------------------------------------------------------
# PLOT
# ----------------------------------------------------------
pl.figure(2)
pl.clf()
pl.plot(resulta[:, 0], resulta[:, 1], 'r+', ms=20)
pl.show()
```

Remerciements

Je remercie chaleureusement Thibaut Le Naour, pour son suivi, ses conseils, sa disponibilité notamment lors de la conception des algorithmes sur Python. Je suis reconnaissante envers toutes les personnes ayant faire une relecture attentive de mon travail ainsi que pour leur soutien sans faille durant les six derniers mois.