Gym AI Tracker

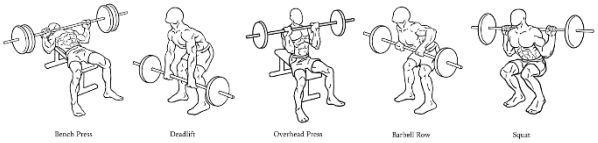
Train your algorithms for better performance!

CHAKIR Fatima Ez-zahra
fatimaezzahra.chakir@etu.uae.ac.ma
ENSAH

1. Introduction

- **Ojectif du projet**

Ce projet a pour objectif de créer des scripts Python pour traiter, visualiser et modéliser les données sensorielles des accéléromètres et des gyroscopes. L'objectif ultime est de développer un modèle d'apprentissage automatique capable de **classifier** les exercices dans la salle du sport et de compter les répétitions.



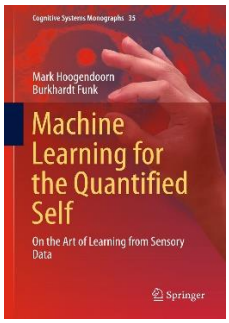
- **L'enjeu de données**

L'enjeu de données utilisé dans ce projet est collecté par des capteurs lors d'un entraînement en salle de sport, pendant des sessions où cinq participants effectuaient divers exercices avec une barre (Figure au-dessus). Donc on va se servir des données sensorielles pour procéder notre projet.

NB : Les exercices sont : **Bench, Dead, ohp, row, squat.**

- **Le Quantified self**

Le quantified self désigne toute personne engagée dans le suivi personnel de données biologiques, physiques, comportementales ou environnementales. Cette auto-surveillance est motivée par un objectif spécifique de l'individu, avec le désir d'agir en fonction des informations collectées, comme décrit par Hoogendoorn et Funk dans leur ouvrage ***"Machine Learning for the Quantified Self: On the art of learning from sensory data"*** (2018).



2. Traitement des données brutes

Chemin de Script
\Gym AI Tracker\src\data\make_dataset.ipynb

Dans cette partie, notre objectif principal est de gérer efficacement les fichiers CSV bruts en les lisant, les traitant et en les fusionnant dans un seul DataFrame. Pour ce faire, nous allons explorer plusieurs étapes cruciales :

- **Compréhension des données**

Chemin de dataset
\GymAITracker\data\raw\data\data

- **Données Accéléromètre** (G-force) (Figure1)**:**

Dans le contexte de notre projet, les données de l'accéléromètre sont mesurées en unités de force G, où un G-force unique sur notre planète Terre est équivalent à 9.8 m/s². Cette mesure, exprimée en "g," représente l'accélération gravitationnelle standard et sert de référence pour quantifier les forces appliquées sur l'axe de l'accéléromètre.

- **Données Gyroscope** (Degré/s)(Figure2):

Les données du gyroscope sont mesurées en unités de degrés par seconde (°/s). Chaque valeur enregistrée dans les données du gyroscope représente la vitesse angulaire autour d'un axe spécifique, exprimée en degrés parcourus par seconde.

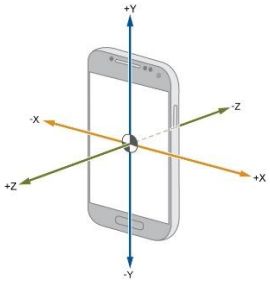


Figure 1

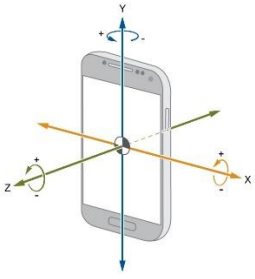


Figure 2

- **Extraction des caractéristiques à partir des noms des fichiers**

Exemple d'un nom de fichier csv:

A-bench-heavy_MetaWear_2019-01-14T14.22.49.165_C42732BE255C_Accelerometer_12.500Hz_1.4.4.csv

- ✓ Participant: A
- ✓ Label: bench
- ✓ Category: heavy

- **Ajout des colonnes utiles**

L'ajout des colonnes importantes à partir de l'extraction des caractéristiques des noms des fichiers: participant, category (medium ou heavy), label (squat, row, dead, ohp, bench) et set (numéro de série).

```
df["participant"] = participant
df["category"] = category
df["label"] = label
df["set"] = acc_set
```

- **Ré échantillonnage des données**

Le rééchantillonnage des données, crucial pour la manipulation de séries temporelles, est appliqué au DataFrame **data_merged** en définissant des règles spécifiques dans un dictionnaire appelé **sampling**. Les 1000 premières lignes sont rééchantillonnées pour donner un aperçu initial, puis l'ensemble des données est rééchantillonné par **jour**, résultant en un nouveau DataFrame (data_resampled)

- **Exportation des données**

Le DataFrame est sauvegardé au format binaire pickle, facilitant le stockage temporaire des données transformées pour une reprise rapide sans réexécution complète du traitement. La restauration ultérieure des données se fait simplement avec pd.read_pickle().

3. Visualisation des données

Chemin de script

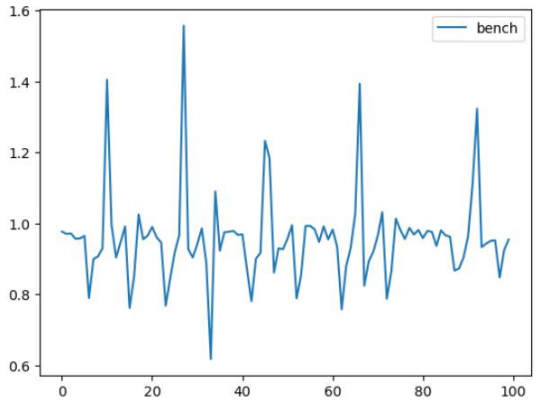
\Gym AI Tracker\src\visualization\visualize.ipynb

- **Traçage des colonnes individuelles par exercice**

Exemple:

Label: Bench

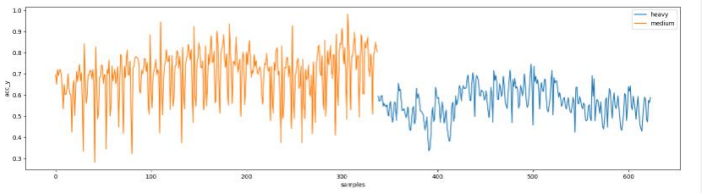
Colonne: acc_y



- **Comparaison des ensembles de catégories**

medium vs heavy

Exercice: squat et participant : A



- **Analyse**

On peut constater que nos données sont de forme sinusoïdale de valeurs positives.

4. Détection des aberrants

Chemin de script

\Gym AI Tracker\src\features\remove_outliers.ipynb

Dans cette partie on va tester Trois méthodes pour détecter les aberrants, et choisir celle qui détecte mieux mais avec tolérance (méthode de Chauvenets). C'est-à-dire sans éliminer beaucoup de points qui peuvent être importants.

NB: les méthodes sont des fonctions prises à partir du Github du livre mentionné au-dessus.

Le référence est au section des références.

- **Méthode 1: Interquartile Range(IQR)**

-Calcul des Quartiles:

Q1 (premier quartile, 25%) : C'est la valeur en dessous de laquelle réside le 25% des données

Q3 (troisième quartile, 75%) : C'est la valeur en dessous de laquelle

-Calcul de l'Interquartile:

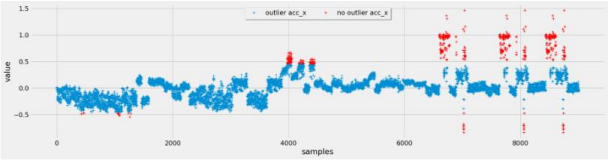
IQR=Q3−Q1 : C'est la plage interquartile qui représente la dispersion centrale des données.

-Calcul des Bornes inf et sup pour la Détection des outliers:

Borne Inférieure (lower_bound) : Q1−1.5×IQR

Borne Supérieure (upper_bound) : Q3+1.5×IQR

Les valeurs situées en dehors de ces bornes sont considérées comme des outliers.



- **Méthode 2: Chauvenets**

- Calcul du Critère Chauvenet :

- N : Le nombre total d'observations dans l'enjeu de données.
- criterion= 1.0 / (N*C) : Le critère de Chauvenet est calculé.

- Calcul de la déviation:

La déviation pour chaque point de données est calculée en utilisant la formule : ***|Xi – mean|/std***

- Calcul des bornes sup et inf :

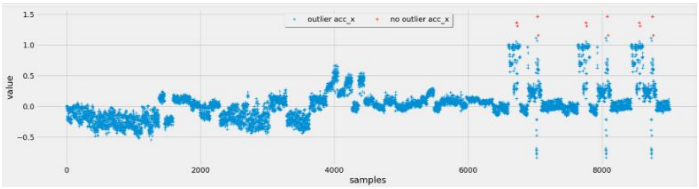
Low = -deviation/sqrt(C)

High = deviation/sqrt(C)

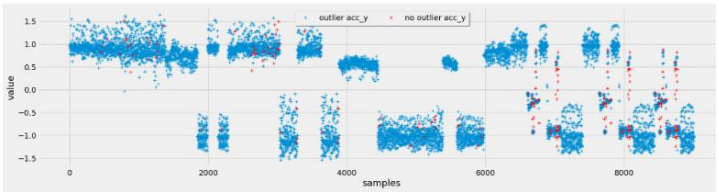
- Calcul de la Probabilité et Marquage des Outliers :

La probabilité d'observer chaque point de données est calculée en utilisant la fonction d'erreur (erf).

Les points sont marqués comme outliers si la probabilité est inférieure au critère de Chauvenet.



- **Méthode 3: Local Outlier Factor (LOF)**



Choix !! Chauvenets était la meilleur méthode pour détecter les aberrants sans éliminer des points importants (d'après les observations de toutes les visualisations). Donc cette fonction sera appliquée sur notre enjeu de données.

5. Extraction des caractéristiques (Feature Enginering) :

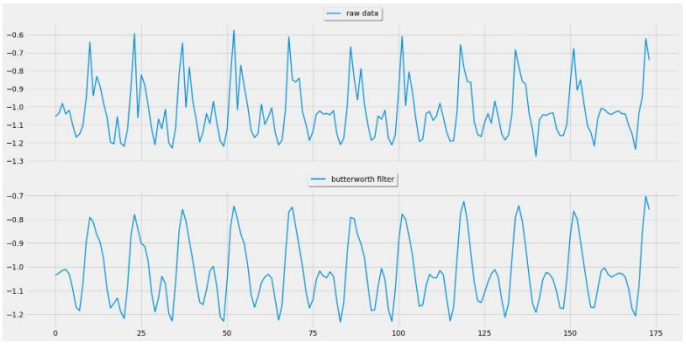
Chemin de script

\Gym AI Tracker\src\features\build_features.ipynb

- **Un filtre passe-bas Butterworth :**

On a appliqué un filtre passe-bas Butterworth pour supprimer le bruit haute fréquence des données brutes. Ce qui peut conduire à un modèle de meilleurs résultats.

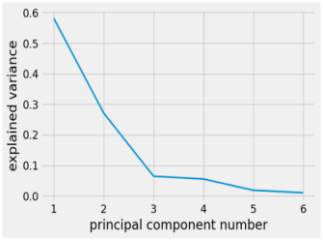
PS : *vous pouvez consulter le module DataTransformation.py pour savoir plus sur l'implémentation de LowPassFilter().*



- **Analyse des composantes principales(ACP) :**

Appliquant l 'ACP sur l'ensemble des données avec un k= 3 (le nombre des composantes), au-dessous illustre la manière avec

laquelle le meilleur “k” a été trouvé.

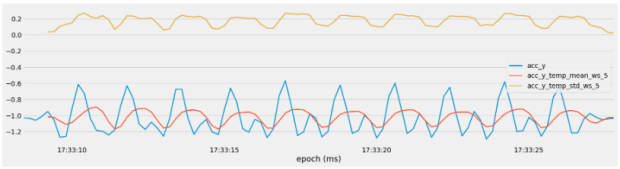


Dans le module DataTransformation.py, la classe personnalisée de l’ACP contient une méthode qui fait la normalisation des données, et une autre et une autre qui applique l’ACP sur les données normalisées et retourne la variance de chaqu’une des colonnes (dont On a servi avec la méthode Elbow pour visualiser le nombre des composantes optimal. Et puis l’ajout des Trois colonnes au dataset pour les utiliser plus tard…

- **Abstraction Temporelle:**

Cette technique consiste à agréger des données sur des intervalles de temps spécifiques (Les fonction d’agrégation ici sont : **Rolling mean** et **std**), facilitant ainsi l'analyse des tendances et des schémas de mouvements sur des périodes définies. La figure au-dessus illustre l’application sur une colonne (acc_y).

PS: Pour plus de détails vous pouvez consulter le module TemporalAbstraction.py dans le répertoire du projet.



- **Transformation de Fourier Discrete:**

“L’idée d’une transformation de Fourier est que toute séquence de mesures que nous effectuons peut être représentée par une combinaison de fonctions sinusoïdes avec des fréquences différentes “ ***Hoogendoorn, M., & Funk, B. (2018). Machine learning for the quantified self.***

Il est utilisé pour représenter les données en termes de composantes de fréquence, permettant une analyse plus efficace des données. Ainsi l’ajout des colonnes de différentes fonctionnalités :

- ***Amplitude (pour chacune des fréquences pertinentes faisant partie de la fenêtre temporelle)***
- ***Fréquence maximale***
- ***Fréquence pondérée (moyenne)***
- ***Entropie spectrale de puissance***

(Voire figure à coté)

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cdot e^{i2\pi kn/N}, \quad n \in \mathbb{Z},$$

Exemple :

Cet exemple montre comment appliquer la DFT à une séquence de longueur $N = 4$ et le vecteur d'entrée

$$\mathbf{x} = \begin{pmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2-i \\ -i \\ -1+2i \end{pmatrix}.$$

$$X_0 = e^{-i2\pi 0\cdot 0/4}\cdot 1 + e^{-i2\pi 0\cdot 1/4}\cdot(2-i) + e^{-i2\pi 0\cdot 2/4}\cdot(-i) + e^{-i2\pi 0\cdot 3/4}\cdot(-1+2i) = 2$$
$$X_1 = e^{-i2\pi 1\cdot 0/4}\cdot 1 + e^{-i2\pi 1\cdot 1/4}\cdot(2-i) + e^{-i2\pi 1\cdot 2/4}\cdot(-i) + e^{-i2\pi 1\cdot 3/4}\cdot(-1+2i) = -2-2i$$
$$X_2 = e^{-i2\pi 2\cdot 0/4}\cdot 1 + e^{-i2\pi 2\cdot 1/4}\cdot(2-i) + e^{-i2\pi 2\cdot 2/4}\cdot(-i) + e^{-i2\pi 2\cdot 3/4}\cdot(-1+2i) = -2i$$
$$X_3 = e^{-i2\pi 3\cdot 0/4}\cdot 1 + e^{-i2\pi 3\cdot 1/4}\cdot(2-i) + e^{-i2\pi 3\cdot 2/4}\cdot(-i) + e^{-i2\pi 3\cdot 3/4}\cdot(-1+2i) = 4+4i$$

$$\mathbf{x} = \begin{pmatrix} X_0 \\ X_1 \\ X_2 \\ X_3 \end{pmatrix} = \begin{pmatrix} 2 \\ -2-2i \\ -2i \\ 4+4i \end{pmatrix}.$$

1. Amplitude (pour chacune des fréquences pertinentes faisant partie de la fenêtre temporelle) :
Pour chaque fréquence f dans la fenêtre temporelle, on va calculer l'amplitude de la composante sinusoïdale correspondante. La formule générale de l'amplitude A pour une fréquence f est :

$$Af = \sqrt{(partie\ réelles)^2 + ((partie\ imagonaire)^2}$$

2. Fréquence Maximale :
On identifie la fréquence qui a l'amplitude maximale dans la fenêtre temporelle. Cela pourrait être la fréquence associée à la composante sinusoïdale ayant l'amplitude maximale.

3. Fréquence Pondérée (Moyenne) :
Calculez la moyenne pondérée des fréquences en utilisant les amplitudes comme poids. La formule générale est :

$$\text{Fréquence pondérée} = \frac{\sum_i A_i \cdot f_i}{\sum_i A_i}$$

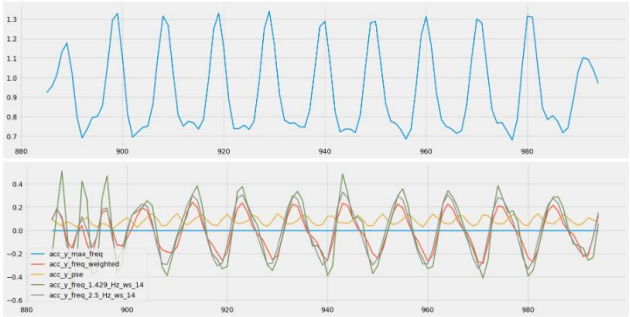
Où A_i est l’amplitude associée à la fréquence f_i

4. Entropie Spectrale de Puissance :
L'entropie spectrale de puissance mesure la répartition des puissances des différentes fréquences dans le signal. La formule générale de l'entropie spectrale de puissance H est :

$$H = -\sum_i \left(\frac{P_i}{P_{total}} \cdot \log_2 \frac{P_i}{P_{total}}\right)$$

Où P_i est la puissance associée à l fréquence f_i et P_{total} celle associée à la fréquence totale.

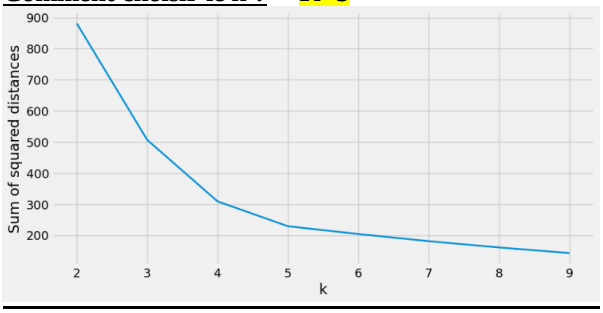
PS : La philosophie de cette transformation est expliquée en code dans le module FrequencyAbstraction.py (**\Gym AI Tracker\src\features\ FrequencyAbstraction.py**). Cette une classe qui effectue une transformation de Fourier sur les données pour trouver les fréquences qui se produisent souvent et filtrer le bruit.



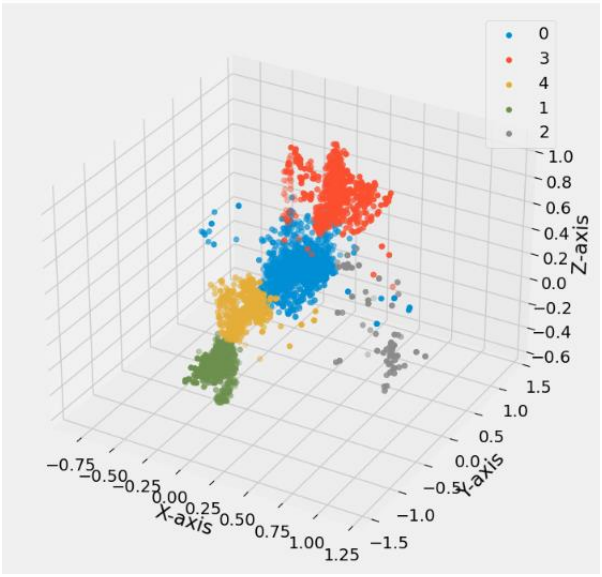
- **Clustering:**
  On cherche à construire des clusters pour notre enjeu de données.

```
df_cluster = df_freq.copy()
cluster_columns = ["acc_x", "acc_y", "acc_z"]
k_values = range(2, 10)
inertias = []
for k in k_values:
    subset = df_cluster[cluster_columns]
    kmeans = KMeans(n_clusters = k, n_init = 20, random_state =
    cluster_labels = kmeans.fit_predict(subset)
    inertias.append(kmeans.inertia_)
#inertias
```

Comment choisir le k ? **K=5**

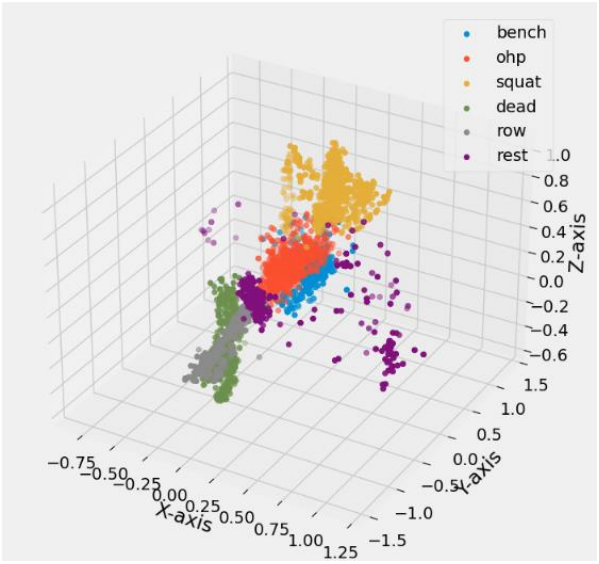


Application du clustering avec k=5



Comparaison:

On va visualiser les données groupées par label (type d'exercice



Remarque : le clustering était juste aussi que le k choisi, le fait de rassembler le label **ohp** avec **bench** en un cluster revient qu'ils se rassemblent selon les mouvements, de même pour **dead** avec **row.**

6. Modèles des prediction:

Chemin de script

\Gym AI Tracker\src\models\train_model.ipynb

On va entrainer 5 algorithmes; Random Forest, Neural Network, Decision Tree, KNN et Naive Bayesin.

Avant d'entamer les procédures pour chaque algorithme, On va sélectionner les meilleures colonnes à utiliser durant l'entrainement.
Ceci est fait en appliquant une Forward Selection, en utilisant un arbre de décision simple comme modèle de base. On ajoute séquentiellement des caractéristiques au modèle, sélectionnant celles qui contribuent le plus à améliorer la performance.
Voici comment la sélection est faite et la série des colonnes issue et leurs scores sur l'enjeu :

- Forward Selection reçoit comme paramètres :
  *X_train et y_train
  *max_features = 10 (nombre maximal des colonnes qu'on veut)
- selected_features=
  ['acc_z_freq_0.0_Hz_ws_14',
  'acc_x_freq_0.0_Hz_ws_14',
  'duration',
  'gyr_r_temp_mean_ws_5',
  'acc_y_temp_mean_ws_5',
  'pca_2',
  'gyr_x_temp_std_ws_5',
  'acc_z_freq_2.143_Hz_ws_14',
  'pca_1',
  'gyr_x_freq_2.5_Hz_ws_14']
- ordered_scores = [
  0.885556704584626,
  0.9903481558083419,
  0.9989658738366081,
  0.9993105825577387,
  0.9996552912788693,
  0.9996552912788693,
  0.9996552912788693,
  0.9996552912788693,
  0.9996552912788693,
  0.9996552912788693]

A. Neural Network

On va appliquer un réseau neuronal pour la classification sur les données d'entraînement (avec la composition spécifiée de couches cachées et le nombre d'itérations si une gridsearch n'est pas activée), et utiliser le réseau créé pour prédire les résultats à la fois pour l'ensemble de test et d'entraînement. Il renvoie les prédictions catégorielles pour l'ensemble d'entraînement et de test ainsi que les probabilités associées à chaque classe, chaque classe étant représentée comme une colonne dans le cadre de données.

Les meilleurs hyperparametres sont trouvés en utilisant une GridSearch sur des ensembles de valeurs :

- `Fonction d'activation:` 'logistic'
- `Alpha:` 0.0001
- `Hidden_layer_sizes:` (100,)
- `Learning_rate:` 'adaptive'
- `Max_iter:` 1000

Accuracy= 0.994829

B. Random Forest

On va utiliser Random Forest pour une classification sur les données d'entraînement (avec la valeur spécifiée pour le nombre minimum d'échantillons dans la feuille, le nombre d'arbres, et une Gridsaerch = True pour des meilleurs hyperparamètres). Il renvoie les prédictions catégorielles pour l'ensemble d'entraînement et de test ainsi que les probabilités associées à chaque classe.

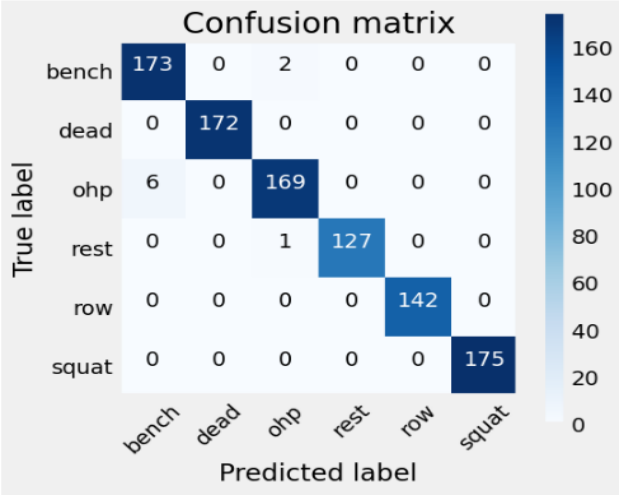
Voici les hyperparametres utilisés pour la construction du modèle, (avec l'utilisation d'une GridSearch)

- Criterion : Entropy
- min_samples_leaf : 2
- n_estimators : 50

Accuracy = 0.997932

PS:

```
if gridsearch:
    tuned_parameters = [
        {
            "min_samples_leaf": [2, 10, 50, 100, 200],
            "n_estimators": [10, 50, 100],
            "criterion": ["gini", "entropy"],
        }
    ]
    rf = GridSearchCV(
        RandomForestClassifier(), tuned_parameters, cv=5, scoring="accuracy"
    )
```



C. Decision Tree

La même procédure que Rondom Forest, seulement qu'on a ici un seul arbre, Voici les hyper paramètres trouvés par GridSearch :

- min_samples_leaf : 50
- Criterion : "gini"

Accuracy = 0.993795

D. Naïve Bayesien

Le modèle probabiliste Naive Bayesien est appliqué directement sur notre données d'entrainement, en utilisant la classe GausienneNB() et ses méthodes.

- Le modèle ajusté est utilisé pour effectuer des prédictions sur les ensembles d'entraînement (train_X) et de test (test_X).
- Les probabilités associées à chaque classe sont obtenues à l'aide de predict_proba().
- Les probabilités prédites pour les ensembles d'entraînement et de test sont stockées dans des DataFrames pandas (frame_prob_training_y et frame_prob_test_y). Chaque colonne de ces DataFrames correspond à une classe.

Accuracy = 0.959669

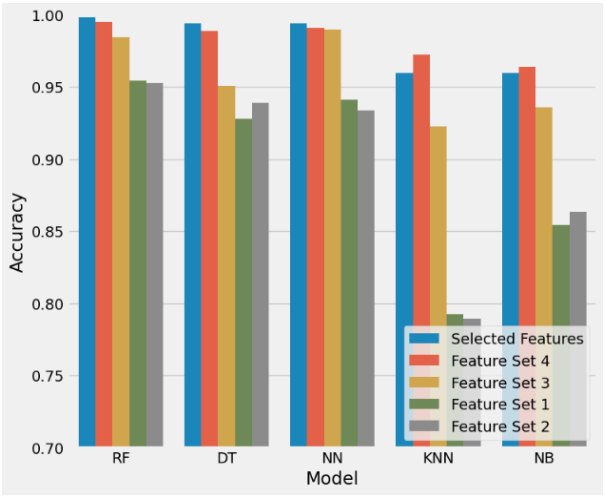
E. KNN

Le modèle est ajusté aux données d'entraînement (train_X et train_y) à l'aide de la méthode fit. La recherche des meilleurs hyperparamètres est faite en utilisant GridSearch :

- n_neighbors : 5

Accuracy = 0.959669

6.1. Comparaison des modèles:



Feature Set 1 , 2, 3, 4 sont des séries des colonnes de différents fonctionnalités que j'ai choisi depuis la Data Frame, puis entrainer les 5 modèles en se basant sur chaqu'une d'elles, pour s'assurer que Selected_Feature trouvée en utilisant Forward Selection est optimale, en effet les 5 modèles ayant une haute performance avec Selected_Features, un ensemble des colonnes qui portent des informations de fréquences sur les données accéléromètre et gyroscope , et temporelles caractéristiques telles que Rolling mean , et de même deux colonnes extraites par l'ACP (pca_1 et pca_2)

Conclusion :

On a RF et NN et DT sont un peu plus proches, Donc on a choisi Random Forest le modèle finale et efficace pour classifier nos exercices du sport.

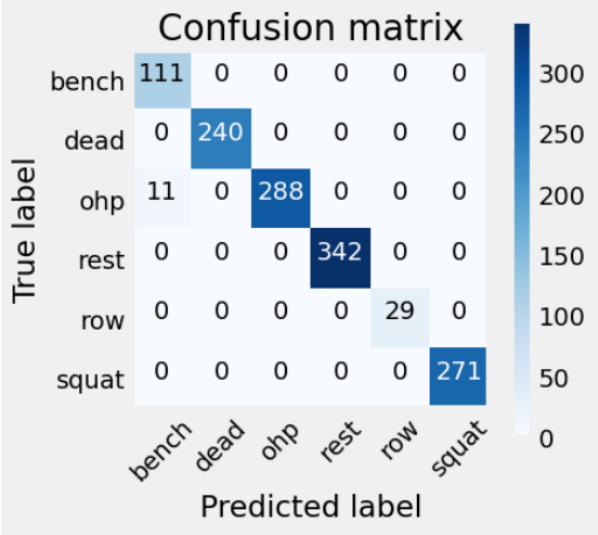
Test si le modèle apprend sans comprendre !!

On va diviser notre données en se basant sur les participants, pour tester le modèle s'il peut performer pour n'importe quel participant ou joueur dans la salle, pour ce faire on a fait : *(rappel : label est la variable cible)*

```
#training data without the participant A's infos and the label
participant_df = df.drop(["set", "category"], axis =1)
X_train = participant_df[participant_df["participant"] != "A"].drop(["label"], axis = 1)
y_train = participant_df[participant_df["participant"] != "A"]["label"]

#test data with the participant A's infos on the label
X_test = participant_df[participant_df["participant"] == "A"].drop(["label"], axis = 1)
y_test = participant_df[participant_df["participant"] == "A"]["label"]
```

Après entrainement du modèle RF sur le nouveau X_train et application sur les données de teste on avait la matrice de confusion suivant :



Cette Matrice de confusion associée à Random Forest Alogorithme entrainé sur les nouvelles données porte des informations précieuses par exemple pour l'exercice 'Bench ' telles que :

- **Précision positive** : la précision de la prédiction positive.

  TP/(TP+FN) =111/111+11 = 91%
- **Précision négative** : la précision de la prédiction négative.

  TN/(TN+FP)= 1170/1170 = 100%
- Accuracy

  (TP+TN)/(TP+TN+FP+FN)

  = 99.14%

Wow! C'est une approche passionnante, la précision reste élevée même si nous divisons les données en fonction des participants. Ainsi, cet algorithme RF conviendra à chaque participant, quel qu'il soit.

7. Construction d'un algorithme personnalisé:

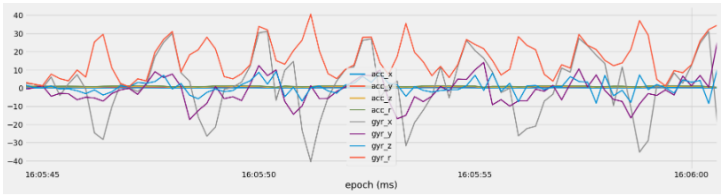
Chemin de script

\Gym AI Tracker\src\models\count_repetitions.ipynb

Dans cette section je vais nager un peu plus profond dans l'enjeu des données et construire un modèle qui compte les répétitions pour chaque série d'exercice (set) en le comparant à un simple regroupement des donnes selon label, catégorie et set. Pour savoir est ce que le nombre de répétitions prédit est le même que celui existe en données.

On va suivre quelques démarches pour arriver à la construction de notre algorithme :

Démarche 1 : Visualiser les données pour identifier des motifs.



Remarque : On remarque que les six colonnes principales (axes d'accéléromètre et gyroscopes) ont des variations différentes pendant une seconde à une autre, mais le point commun est qu'ils sont sinusoïdales (Analyse précédente) avec **des extremums** bien clairs qui signifient des flexions de se lever et de se baisser c'est-à-dire faire une première répétition d'un exercice quelconque.

Avant de passer, il est important d'éliminer le bruit dans ces formes sinusoïdales, et comme on a déjà fait on va procéder avec LowPasseFilter.

Démarche 2 : Elimination du bruit par un FiltrePasseBas

Même processus que celui appliquer dans la partie 5/ de l'Extraction des caractéristiques, et Voici une visualisation pour le cas actuelle d'une colonne de acc_r (R^2 des trois axes acc_x acc_y, acc_z).

Paramètre important à considérer pendant LowPassFilter est:

- **cutoff_frequency** (fréquence de coupure) :

  La fréquence à laquelle le filtre commence à atténuer les fréquences plus élevées.

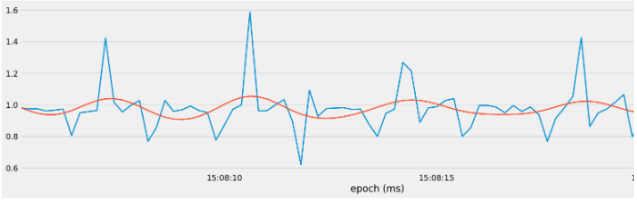
  Les valeurs de ce paramètre vont dépendre de type d'exercice (label) (il y a des exercices de grand potentiel donc un cuttoff_freq sera élevé) :

  •***Squat:*** cutoff_freq = 0.35

  •***Row:*** cutoff_freq = 0.65

  •***Ohp:*** cutoff_freq = 0.35

  •***Dead & bench:*** cutoff_freq = 0.4



Creation de la fonction pour compter les répetition !

Cette fonction **count_rep()** est conçue pour traiter un signal temporel (colonne)représentant des données d'exercice(label), notamment le mouvement d'un capteur. Elle applique un ***filtre passe-bas*** pour atténuer les hautes fréquences, puis identifie ***les pics*** dans le Signal filtré, supposant que ces pics représentent des ***répétitions de l'exercice***. Le nombre de pics détectés est ensuite ***renvoyé*** comme le nombre de répétitions.

- **Filre pass-bas :**

Comme expliqué avant l'utilisation de la fonction LowPass.low_pass_filter pour appliquer un filtre passe-bas aux colonnes.

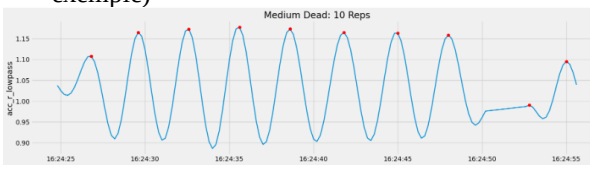
- **Détection des pics** :

A l'aide de la fonction *argrelextrema()* pour trouver les indices des pics dans le colonne filtré.

- **Tracé du signal filtré et des pics**
- **Nombre de répétitions en retourne :**

Le nombre de pics détectés est renvoyé par la fonction comme étant le nombre de répétitions de l'exercice.

- **Application sur les colonnes** ☺ (Voici un exemple)



Creation d'une Benchmark DataFrame !

Dans cette section on va générer un nouveau DataFrame (rep_df) qui contient le nombre maximal de répétitions prédites (reps_pred) pour chaque ensemble d'exercices spécifique (label), en utilisant la fonction **count_rep** construite au-dessus pour détecter les répétitions dans des sous-ensembles de données.

a) Groupement par label, category, et set :

- Utilisation de la méthode groupby pour regrouper le DataFrame initial (df) par les colonnes "label", "category", et "set".
- Appliquant la fonction max() sur la colonne "reps" pour obtenir le nombre maximal de répétitions par groupe.
- Ajout d'une colonne "reps_pred" au DataFrame rep_df et l'initialisation à zéro.

b) Application de l'algorithme:

- Création d'un sous-ensemble de données (subset) en fonction de l'ensemble d'exercices actuel dans l'itération.
- Application de la fonction count_rep pour compter le nombre de répétitions dans le sous-ensemble avec les paramètres déterminés précédemment. (differents cuttof_freq)

DataFrame Resultante !

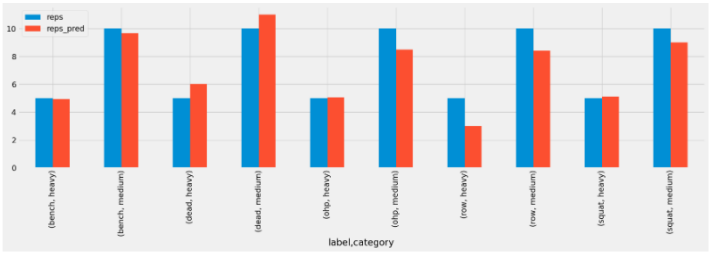
	label	category	set	reps	reps_pred
0	bench	heavy	1	5	5
1	bench	heavy	2	5	5
2	bench	heavy	3	5	5
3	bench	heavy	4	5	5
4	bench	heavy	30	5	4
...	...	...	...	...	...

Evaluation des résultats !

Erreur Moyenne Abs = 0.93

Comparaison !

Entre les répétitions prédites par notre algorithme ***count_rep*** *et celle calculées simplement par max().*



Conclusion !

Notre algorithme de comptage s'avère efficace pour compter les répétitions de chaque série d'exercice, avec une grande similarité à un simple regroupement par exercice et catégories sur les données

Discussion

Les modèles en top; Random Forest, Decision Tree et Neural Network avaient une très bonne performance, Le fait qu'on a seulement une précision positive de 91% revient à dire que notre algorithme a des limitations. Quelles sont alors ces limitations?

Comme vous savez notre projet est sur la suivie et classification des exercices d'entrainement dans une salle du sport ***(le Pourquoi ?vue que la majorité des suivies étaient sur L'activité sanguine, cardiaque ou respiratoire),*** et comme les exercices **ohp** et **bench** se ressemblent au niveau de la position des mains et de corps, aussi pour l'exercice **dead** et **row** qui se resemblent au niveau d'inclination du corps (C'est mieux de voire la figure des exercices dans l'introduction). Donc on peut clairement justifier la baisse de TPR= 91% causé par la classification des 11 exercices **ohp** comme exercice type **bench**.(Voire matrice de confusion)

GitHub code
https://github.com/ZAHIRA201/GYM_AI_Tracker.git
Lien Drive
https://drive.google.com/drive/folders/19F80kKt1KW41HO0qFDQqYwR9la-EL9rn?usp=sharing

References

[1] Le livre : ***"Machine Learning for the Quantified Self: On the art of learning from sensory data"*** (2018). https://link.springer.com/book/10.1007/978-3-319-66308-1
https://ml4qs.org/t

[2] Github page de meme livre :
https://github.com/mhoogen/ML4QS/
En effect ce Github comporte le code certains modules utilisés durant la construction du projet :
DataTansformation.py
(Pour LowPassFilter & ACP)
TemporalAbstraction.py (Rolling mean & std)
FrequancyAbstraction.py
(Pour transformation de Fourier)
LearningAlgorithms.py

[3] Pandas resampling rules:
https://pandas.pydata.org/pandas-docs/stable/user_guide/timeseries.html#offset-aliases

[4] Understanding Pickle files
https://www.geeksforgeeks.org/understanding-python-pickling-example/

[5] Customizing Matplotlib with style sheets and rcParams
https://matplotlib.org/stable/users/explain/customizing.html

[6] Using Pandas Method Chaining to improve code readability
https://towardsdatascience.com/using-pandas-method-chaining-to-improve-code-readability-d8517c5626ac

[7] Boxplot and interquartile range (IQR)
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.boxplot.html/

[8] Local Outlier Factor (LOF)
https://scikit-learn.org/stable/auto_examples/neighbors/plot_lof_outlier_detection.html

[9] Butterwoth LowPass Filter
https://ml4qs.org/t

[10] Temporal Abstraction (Rolling mean &std)
https://datagy.io/rolling-average-pandas/

[11] Transformation de Fourier
https://en.wikipedia.org/wiki/Discrete_Fourier_transform

[12] Elbow method
https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/

[13] GridSearch understanding
https://www.geeksforgeeks.org/grid-searching-from-scratch-using-python/