BLOC 6 certification
Mars 2024

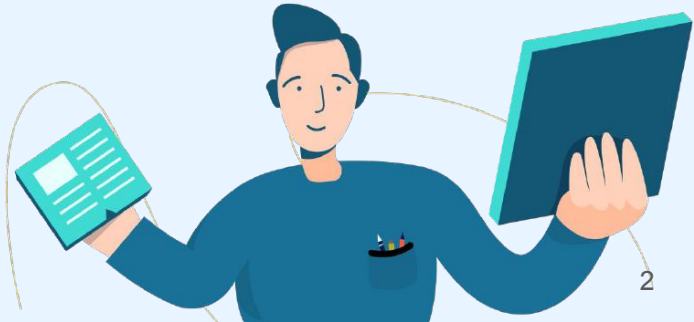
Optimizing Sports Performance:

Activity Monitoring focused on vacation period

Data Science Fullstack (ds-ft-od-03)

Eugenia Mellano & Doriane Doyon-Dousse

Sommaire :



Problématique & Objectifs

Problématique

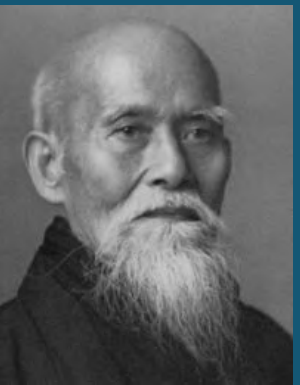
- Maintenir un niveau optimal de performance et condition physique pendant les périodes de repos
- Maximisation du temps de récupération.
- Les sportifs oublient ou évitent de partager toutes les informations.

Objectif

- Créer un modèle pour pouvoir identifier **passivement** le activités réalisées via le senseur (accéléromètre) intégré du smartphone ⭐

Source image : https://capsurvous.com/objectifs-pedagogiques-finaux-et-intermediaires/

“Huit forces soutiennent la Création : **Le mouvement et l'immobilité**, La solidification et la fluidité , L'extension et la contraction, L'unification et la division”



Morihei Ueshiba

Morihei Ueshiba est le fondateur de l’aïkido.

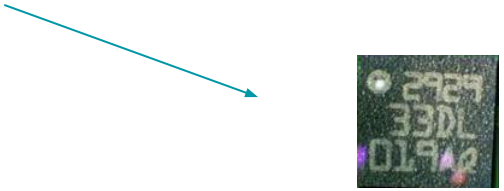
14 décembre 1883 - 26 avril 1969

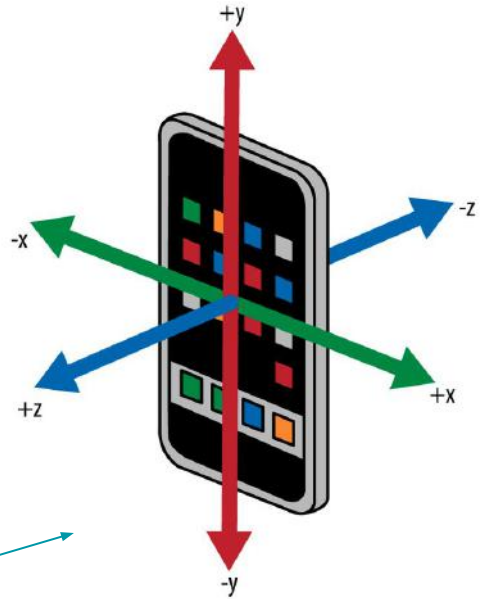
Contexte

Contexte 🔍

Sans y prêter attention, nos smartphones regorgent de capteurs / **sensors** :

- Capteurs GPS
- Capteurs de vision (cameras)
- Capteurs audio (microphones)
- Capteurs de lumière
- Capteurs de température
- Capteurs de direction (compas magnétiques)
- **Capteurs d'accélération (accéléromètres)**

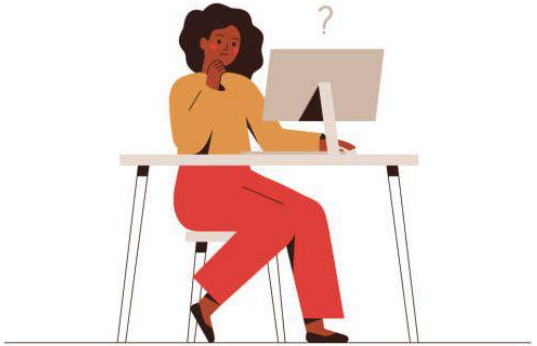




Source image : https://steemit.com/technology/@mike11/accelerometer-in-a-smartphone-how-it-works

Contexte 🔍

- Et si on utilisait ce capteur (**accéléromètres téléphoniques**) afin d'**exploration ses données** ?
- **Dans quel but ?**
  - Reconnaissance d'activité ✅
    - Identifier l'activité physique pratiquée par un utilisateur
      - Marche
      - Jogging
      - Descente escaliers
      - Monter escaliers
      - Assis
      - Debout



- **Time series data** : série chronologique agrégé
  - résument l'activité de l'utilisateur sur **10-secondes intervalles**.

Source image : https://www.reseauode.fr/ameliorer-mon-quotidien/pratiquer-une-activite-physique/

Eugenia **Mellano** & Doriane **Doyon-Dousse**

Contexte 🔍

- **Et si ces données résultantes nous permettait d'induire un modèle prédictif de reconnaissance d'activité?**
  - L'utilisateur *n'aurait rien à faire* 😎
  - Seulement porter son smartphone dans sa poche
    - Récolte passive de données
  - Reconnaissance d'activité physique ✅

- Dans un but d'**optimisation des performances sportives :**
  - Suivi d'activité du sportif centré sur les périodes off (vacances, week-end)
- Permettrait de :
  - Générer un **profil d'activité quotidien/hebdomadaire** 🏋
  - Déterminer si le sportif effectue une **quantité d'activité saine pré compétition**
    - Envoyer notification téléphonique 📲
      - Si trop d'activité (non respect temps de repos) ❌
      - Si pas assez d'activité (avant compétition importante par exemple)

Source image : https://meritis.fr/blog/intelligence-artificielle-ia-comment-ca-fonctionne/
https://fr.freepik.com/photos-premium/femme-met-telephone-portable-dans-poche-jeans-fond-clair-gris-isolation-espace-pour-texte_13338791.htm

Dataset

Dataset

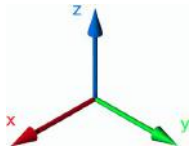
Laboratoire WISDM

(Wireless Sensor Data Mining)

- **Les variables**

- User : 36 utilisateurs différents
- Activity : 6 activités différentes
- Timestamp : nanosecondes
- X-axis
- Y-axis
- Z-axis

} 3D variables (accéléromètre)



Collecte

de la donnée

Collecte de la donnée 📂

- **Contrôlée par une application créée par le Laboratoire WISDM**
  - Application & Smartphone basé sur **Android** (car gratuit)
  -
- **L'application :**
  - Grâce à une **simple interface utilisateur graphique**, a permis d'enregistrer
    - Nom de l'utilisateur,
    - **Démarrer/Arrêter** la collecte de données
    - **Etiqueter** l'activité en cours.
  -
- **Les sujets :**
  - Port téléphone Android : **poche avant du pantalon**
  - Doivent **pratiquer les 6 activités** (périodes de temps spécifiques)
  -
- **Données collecté de l'accéléromètre**
  - Toutes les 50 ms = **20 échantillons par seconde.**

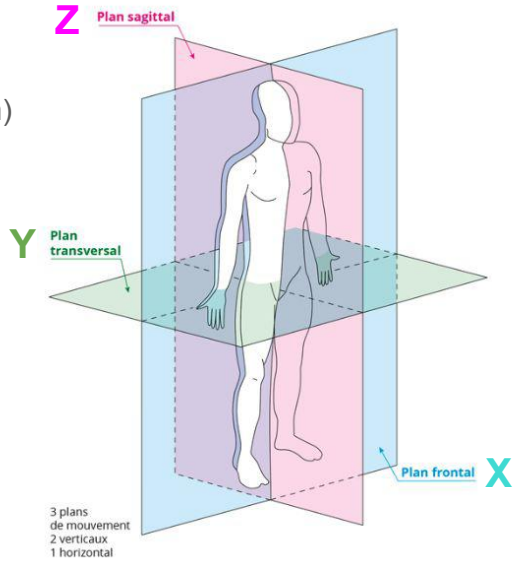
Source image : https://www.edps.europa.eu/data-protection/our-work/subjects/transparence_fr

Collecte de la donnée 📂

Comment l'accéléromètre téléphonique analyse les mouvements ?

- **Analyse linéaire tridimensionnelle** (3D)
  - Détecte et analyse les **changement de vitesse**
  - Mesure la force résultante exercée (composante sensible à l'accélération)

- **X - axis**
  - Représente accélération dans le **plan horizontal**
  - De **droite à gauche** et vice versa
- **Y - axis**
  - Représente accélération dans le **plan vertical**
  - De **haut en bas** or vice versa
- **Z - axis**
  - Représente accélération dans le **plan axial**
  - **Devant / derrière** par rapport au centre de l'appareil

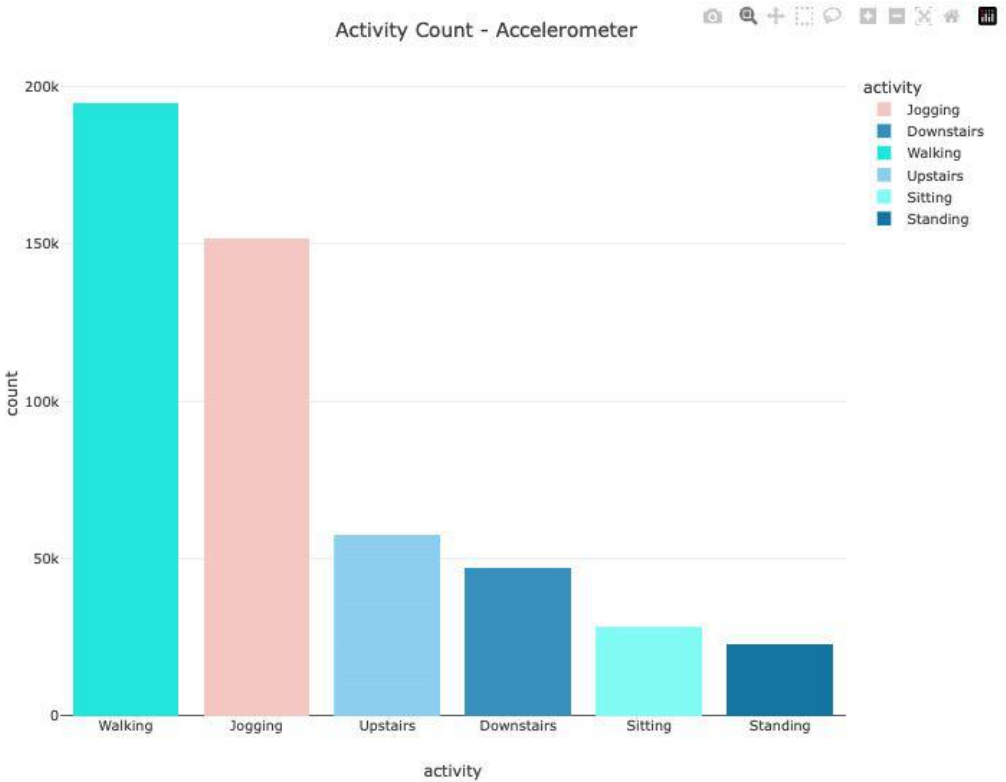


Source image https://lifeforcewithyou.com/plan-anatomique/

Exploratory Data Analysis

Analysons nos données

Exploratory Data Analysis

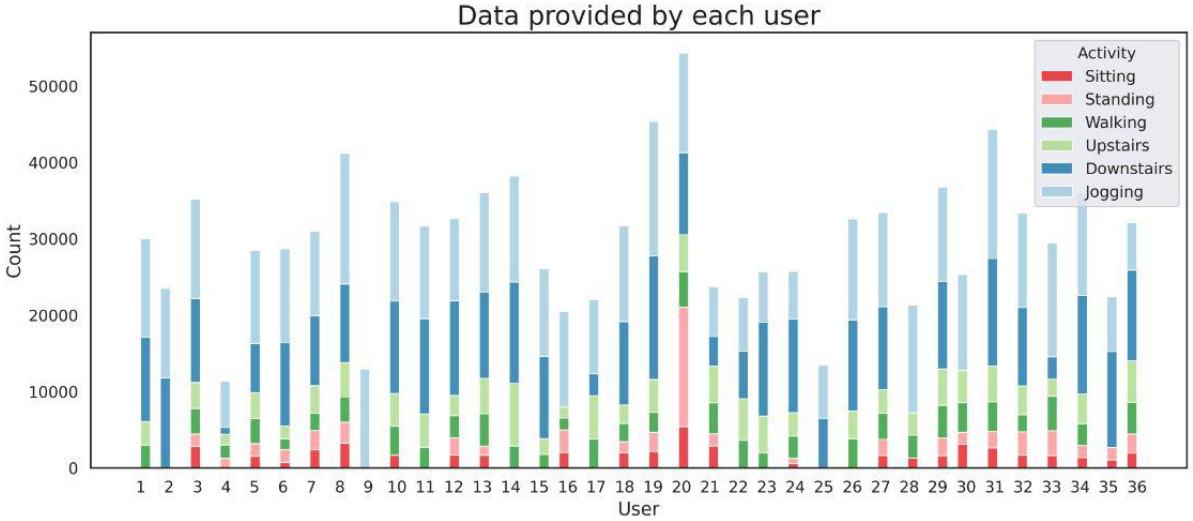



```python
# Graph of acitivity frequency with matplotlib
# ------------------------------------------------------------------------------
import plotly.express as px # importations of plotly librairies
import plotly.graph_objects as go
import plotly.io as pio

pio.templates["doriane"] = go.layout.Template( # building a color template)
    layout_colorway=[
        "#f3cec9",
        "#4B9AC7",
        "#4BE8E0",
        "#9DD4F3",
        "#97FBF6",
        "#2A7FAF",
        "#23B1AB",
        "#0E3449",
        "#015955",])

pio.templates.default = "doriane"

# Graph with diffrent colors for each activities
fig = px.bar(df_sample, x='activity', color='activity')
# Setting title and figsize
fig.update_layout(title="Activity Count - Accelerometer",
                  autosize=False, width=900, height=700)
fig.update_xaxes(categoryorder='array',
                 categoryarray= ['Walking', 'Jogging', 'Upstairs',
                                 'Downstairs', 'Sitting', 'Standing'])
fig.show('colab') # after ordering values, show the graph
```

What are the most frequent activities ?

- **Top 3 most frequent activities :**
  - **Walking**
    - Largely above all other subclasses
  - **Jogging**
    - In second position
  - **Upstairs**
    - In third position

Exploratory Data Analysis

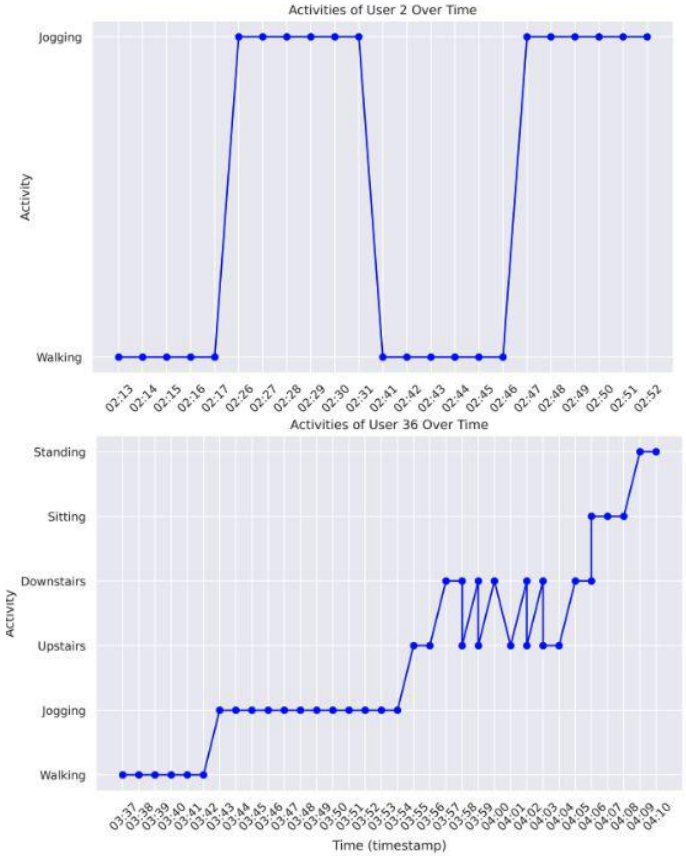


Are activities frequency different between users ?

- 👋 Almost all users performed 6 different activities
- Let's take a closer look at the **behavior over a timeline** of some user

```
# Graph activity frequency per users seaborn
# ------------------------------------------------------------------------------------------------------------
# Counting frequencies of each activity for each user
activity_counts = df_acc.groupby(['user', 'activity']).size().unstack(fill_value=0)
activity_counts = activity_counts[activity_order]

plt.figure(figsize=(15, 6))
plt.title('Data provided by each user', fontsize=20)
sns.set_style("white")
sns.set_theme(rc={'figure.dpi': 600})
sns.histplot(data=df_acc, x='user', hue='activity', multiple='stack', palette='Paired')

# Manually create a legend based on actity order
legend_labels = activity_order
plt.legend(legend_labels, title='Activity', fontsize=12)

# Setting titles and showing graph
plt.xlabel('User', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(ticks=df_acc['user'].unique(), fontsize=12)
plt.yticks(fontsize=12)

plt.show()
```

Exploratory Data Analysis

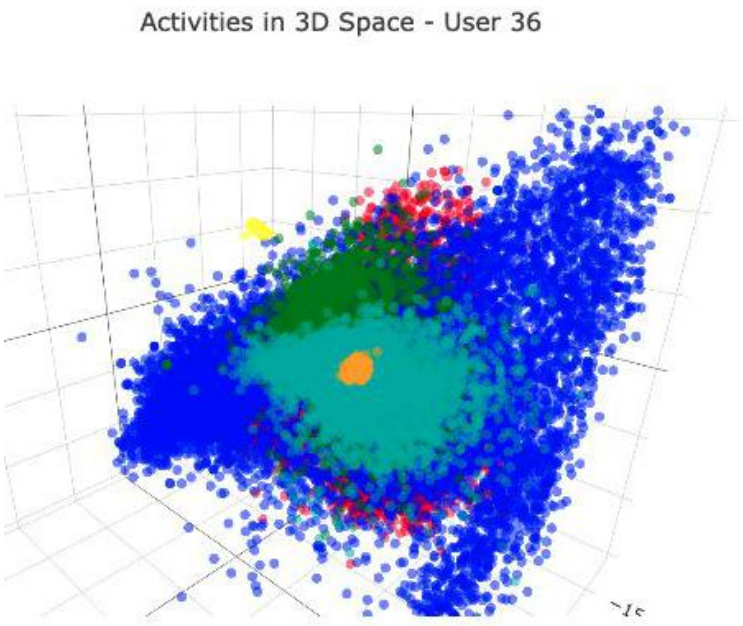



```python
# Timestamps per user : taking the example of user number 2 and user number 36
# ------------------------------------------------------------------------------------------------------------------------
df_acc['timestamp_str'] = df_acc['timestamp'].apply(lambda x: timedelta(microseconds=round(x, -3) // 1000))
# Filter data for user 2 and user 36
user_2_data = df_acc[df_acc['user'] == 2]
user_36_data = df_acc[df_acc['user'] == 36]
#user_2_data['timestamp_hour'] = user_2_data['timestamp_str'].apply(lambda x: str(x)[6:12])
#user_36_data['timestamp_hour'] = user_36_data['timestamp_str'].apply(lambda x: str(x)[6:12])
user_2_data.loc[:, 'timestamp_hour'] = user_2_data['timestamp_str'].apply(lambda x: str(x)[6:12])
user_36_data.loc[:, 'timestamp_hour'] = user_36_data['timestamp_str'].apply(lambda x: str(x)[6:12])

user_2_data_grouped = user_2_data.groupby(['timestamp_hour', 'activity']).size().reset_index(name='count')
user_36_data_grouped = user_36_data.groupby(['timestamp_hour', 'activity']).size().reset_index(name='count')


# Set up the figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(18, 6))

# Plot the activities of user 2
axs[0].plot(user_2_data_grouped['timestamp_hour'], user_2_data_grouped['activity'], marker='o', color='blue')
axs[0].set_xlabel('Time (timestamp)')
axs[0].set_ylabel('Activity')
axs[0].set_title('Activities of User 2 Over Time')
axs[0].tick_params(axis='x', labelrotation=45)
axs[0].grid(True)

# Plot the activities of user 36
axs[1].plot(user_36_data_grouped['timestamp_hour'], user_36_data_grouped['activity'], marker='o', color='blue')
axs[1].set_xlabel('Time (timestamp)')
axs[1].set_ylabel('Activity')
axs[1].set_title('Activities of User 36 Over Time')
axs[1].tick_params(axis='x', labelrotation=45)
axs[1].grid(True)

# Adjust layout
plt.tight_layout()

# Display the plot
plt.show()
```

Let's have a look of activities evolution over time (2 users example)

- **User 2 :** we can see this user alternatively jogging and walking
- **User 36 :** this user is going through all types of the 6 activities

Exploratory Data Analysis



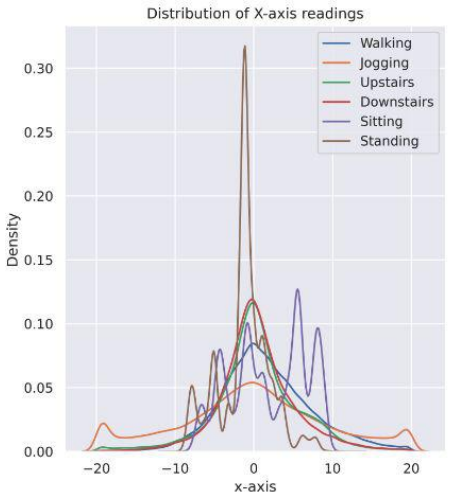

```python
# Let's represent in 3D all axis in relation with activities (example of user number 36)
# ------------------------------------------------------------------------------------------

df_user36 = df_acc[df_acc['user'] == 36] # we select on user going through all activities

activity_colors = {'Walking': 'red',   # Defining a unique color for each activities
                   'Jogging': 'blue',
                   'Upstairs': 'green',
                   'Standing': 'orange',
                   'Upstair': 'purple',
                   'Sitting': 'yellow'}

fig = px.scatter_3d(df_user36, x='x-axis', y='y-axis', z='z-axis', # 3D graph with plotly
                    color='activity', color_discrete_map=activity_colors,
                    title='Activities in 3D Space - User 36')

fig.update_traces(marker=dict(size=4)) # Reducing size of points for better visualization

fig.update_traces(opacity=0.5) # Ajust opacity of points
fig.update_layout(legend=dict(font=dict(size=16)))

fig.show() # Show the interactive graph
```

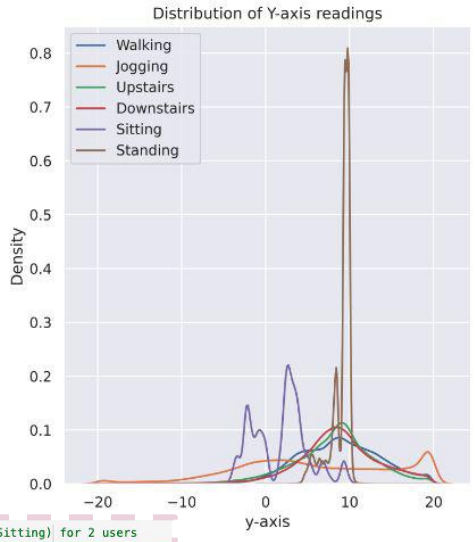
Dispersion of activities in a tri-axial plan (3D)

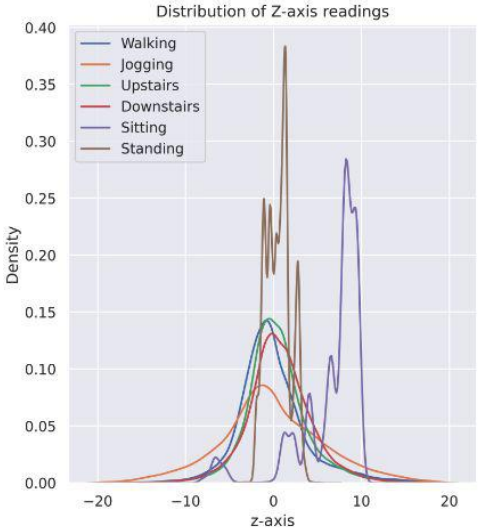
- We can observe that **Standing** and **Sitting** tend to concentrate in one point (no dispersion)
- While **Jogging** is the most spread variable (**Walking** also but less)
- **Upstairs** and **Downstairs** are spread in 3 plan but more concentrated

Exploratory Data Analysis

Eugenia Mellano & Doriane Doyon-Dousse








```python
# Let's represent in 3D all axis in relation with 2 activities (Jogging/Sitting) for 2 users
# ------------------------------------------------------------------------------------------
x_acc_cols = ['x-axis']
y_acc_cols = ['y-axis'] # List of columns representing X, Y and Z axis
z_acc_cols = ['z-axis']

axis_labels = ['X', 'Y', 'Z'] # Naming the axis

fig, axes = plt.subplots(1, 3, figsize=(16, 6)) # Creation of subgraphs
axes = axes.flatten()

# For each axis we generate a plot in relation with activities (for loop)
for label, cols, ax in zip(axis_labels, [x_acc_cols, y_acc_cols, z_acc_cols], axes):
    for activity in df_acc['activity'].unique(): # Kernel density for each activities (per axis)
        sns.kdeplot(data=df_acc[df_acc['activity'] == activity][cols[0]], ax=ax, label=activity)

    ax.set_title(f'Distribution of {label}-axis readings') # Setting the title
    ax.legend() # Aggregating legend

plt.tight_layout()
plt.show() # Display of graphs
```

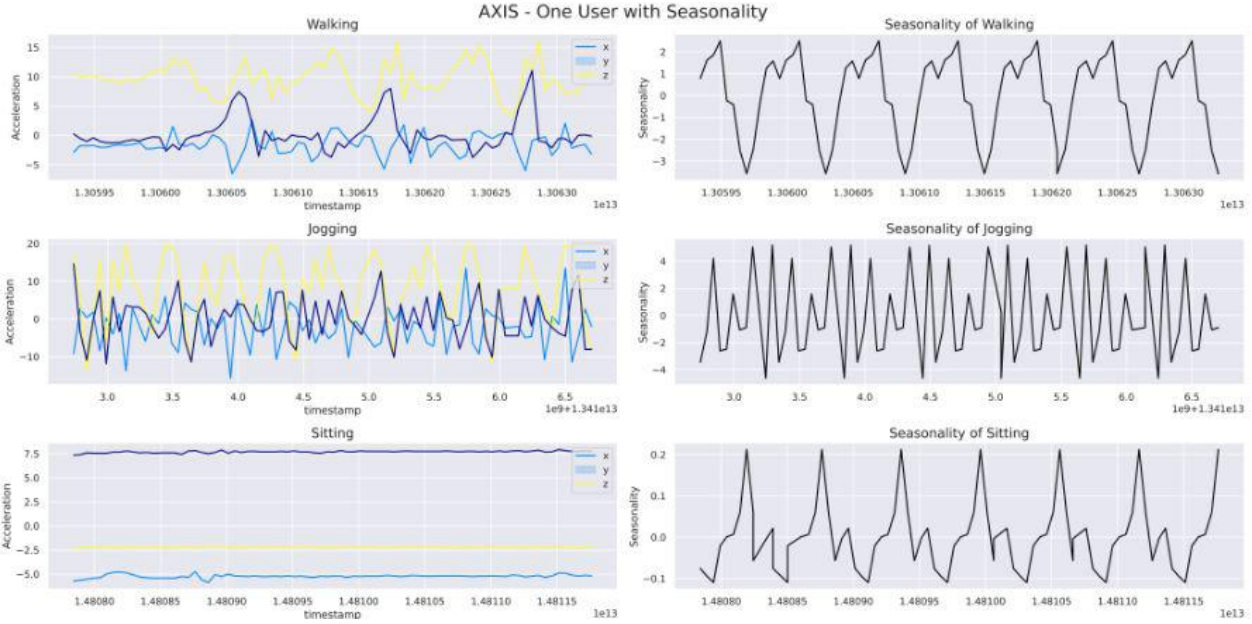
Representation of activities for each axis ?

- **Standing** has the highest kernel density in all 3 plans
- **Jogging** has the highest spread in all 3 plans
- **Activities for Y axis (transversal plan) are NOT centered :**
  - **It's normal because the phone is either on the right or left pocket of the pant : creating a deviation ✅**

Exploratory Data Analysis

Eugenia Mellano & Doriane Doyon-Dousse

Exemple 3 activités




```python
# Goal here is to detect some patterns of activities over time : seasonality (for 1 user)
# ------------------------------------------------------------------------------------

activities = ['Walking', 'Jogging', 'Sitting', # Defining list of activities
              'Standing', 'Upstairs', 'Downstairs']

plt.figure(figsize=(15, 10)) # Setting size of the figure
sns.set_theme(rc={'figure.dpi': 800}) # Setting theme

for idx, activity in enumerate(activities, start=1): # Graphs creation : loop over activities
    # Filtering data : focusing on user 36 and its associated activities
    data_user_36 = df_acc[(df_acc['user'] == 36) & (df_acc['activity'] == activity)][:80]

    # For each axis : calcul of seasonality
    seasonal_x = seasonal_decompose(data_user_36['x-axis'], model='additive', period=12).seasonal
    seasonal_y = seasonal_decompose(data_user_36['y-axis'], model='additive', period=12).seasonal
    seasonal_z = seasonal_decompose(data_user_36['z-axis'], model='additive', period=12).seasonal
    seasonal_combined = seasonal_x + seasonal_y + seasonal_z # Sum of seasonality for each axis

    plt.subplot(6, 2, idx * 2 - 1) # Creation of a subplot

    # Building lineplots for x, y and z axis  with associated color
    sns.lineplot(y='x-axis', x='timestamp', data=data_user_36, color='dodgerblue')
    sns.lineplot(y='y-axis', x='timestamp', data=data_user_36, color='yellow')
    sns.lineplot(y='z-axis', x='timestamp', data=data_user_36, color='darkblue')

    plt.legend(['x', 'y', 'z'], loc='upper right') # Setting legend and titles
    plt.ylabel('Acceleration')
    plt.title(activity, fontsize=15)

    plt.subplot(6, 2, idx * 2) # Creation of a subplot

    plt.plot(data_user_36['timestamp'], seasonal_combined, color='black') # combined seasonality

    plt.ylabel('Seasonality') # Setting titles and labels
    plt.title(f'Seasonality of {activity}', fontsize=15)

plt.suptitle('AXIS - One User with Seasonality', fontsize=20) #Setting general title

plt.tight_layout() # Global title set and display the graph
plt.show()
```

- Accélérations plus élevés dans les valeurs Y dans la plupart des activités
- la marche et la course peuvent être décrits en terme du temps entre les pics
- S'asseoir présent des différences dans les valeurs des axes en raison de l'orientation du dispositif par rapport à la Terre

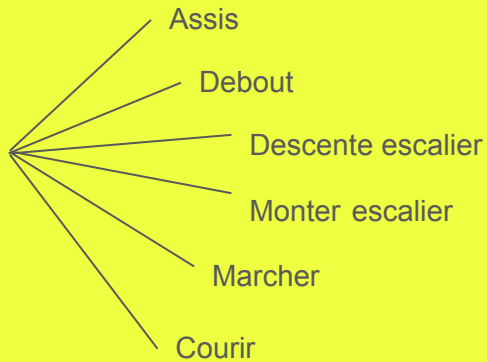
Les modèles

Analyse prédictive

Type de problématique
(intelligence artificielle)

- Problème de **classification multiclasse**

- **6 classes**
  'activity'



Variable cible
(target)

'activity'

'activity'

Eugenia **Mellano** & Doriane **Doyon-Dousse**

Recurrent Neural Network

Modèle de Deep Learning

LSTM Network

Recurrent Neural Network 🧠

LSTM Network

Pourquoi choisir un modèle LSTM?

- Capable de capturer et d'apprendre dans les données
- Capables de se souvenir des informations sur de longue durée
- Il n'est pas nécessaire de réaliser une extraction manuelle des caractéristiques des données des capteurs

Recurrent Neural Network 🧠

LSTM Network

Extraction des segments de données et leurs étiquettes 📎

- `segments` : Liste vide pour stocker les **segments** de données.
- `labels` : Liste vide pour stocker les **étiquettes** des activités.
- `df_acc` : DataFrame contenant des **données d'accélération**.
- `N_TIME_STEPS` : **Taille** de chaque **segment de données**.
- `step` : Pas entre les segments
  Avec un chevauchement de 80 % dans ce cas).
- `xs, ys, zs` : Valeurs d'accélération dans les segments.
  label. L'étiquette la plus commune dans chaque segment.

Recurrent Neural Network 🧠

LSTM Network

- Data preparation 🛠️

1.

```
# Let's set necessary for the future execution of the model
# ------------------------------------------------------------
RANDOM_SEED = 24
N_TIME_STEPS = 50   # 50 records in each sequence
N_FEATURES = 3      # x, y, z
step = 10           # window overlap = 50 -10 = 40  (80% overlap)
N_CLASSES = 6       # class labels
N_EPOCHS = 50
BATCH_SIZE = 1024
LEARNING_RATE = 0.0025
L2_LOSS = 0.0015
```

Méthode Windowing:

- Garantie que les informations pertinentes ne soient pas perdues lors de la segmentation

- Utile quand les segment adjacents sont importantes pour l'analyse et prédiction

2.

```
# Building the tri dimentionals inputs for the Long Short-Term Memory model
# ------------------------------------------------------------
segments = []
labels = []

for i in range(0, df_acc.shape[0] - N_TIME_STEPS, step):
    xs = df_acc['x-axis'].values[i: i + N_TIME_STEPS]
    ys = df_acc['y-axis'].values[i: i + N_TIME_STEPS]
    zs = df_acc['z-axis'].values[i: i + N_TIME_STEPS]

    # We calculate the mode of the 'activity' column
    label = df_acc['activity'][i: i + N_TIME_STEPS].mode()[0]
    segments.append([xs, ys, zs])
    labels.append(label)

# Reshape the segments to a single list
reshaped_segments = np.asarray(segments, dtype=np.float32).reshape(-1, N_TIME_STEPS, N_FEATURES)
labels = np.asarray(pd.get_dummies(labels), dtype=np.float32)
reshaped_segments.shape
```

Recurrent Neural Network 🧠

LSTM Network

- The model 🚀

3.

```
# THE   MODEL  (Long  Short-Term  Memory)
# ----------------------------------------------------------------------------------------------------
epochs = 20
batch_size =  1024

model = Sequential()
# RNN layer
model.add(LSTM(units = 128, input_shape= (X_train_m.shape[1], X_train_m.shape[2]))) # input_shape = 50, 3
# Dropout layer
model.add(Dropout(0.4))
# Dense layer with ReLu
model.add(Dense(units = 64, activation='relu'))
# Softmax layer
model.add(Dense(y_train_m.shape[1], activation = 'softmax'))
# Compile model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

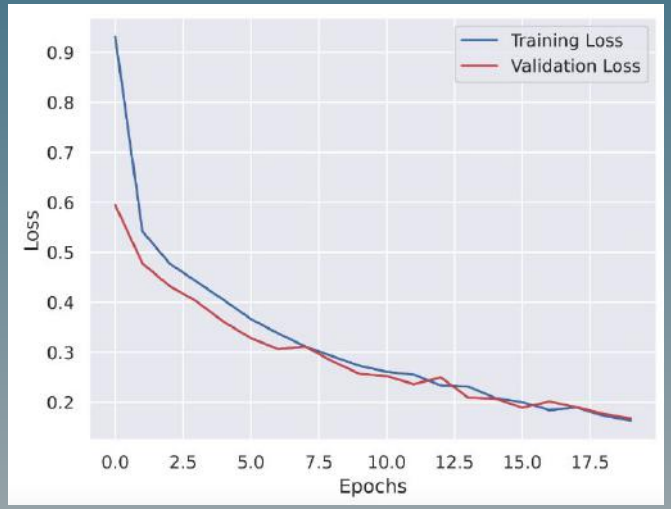
Recurrent Neural Network 🧠

LSTM Network

- Performances 🔍

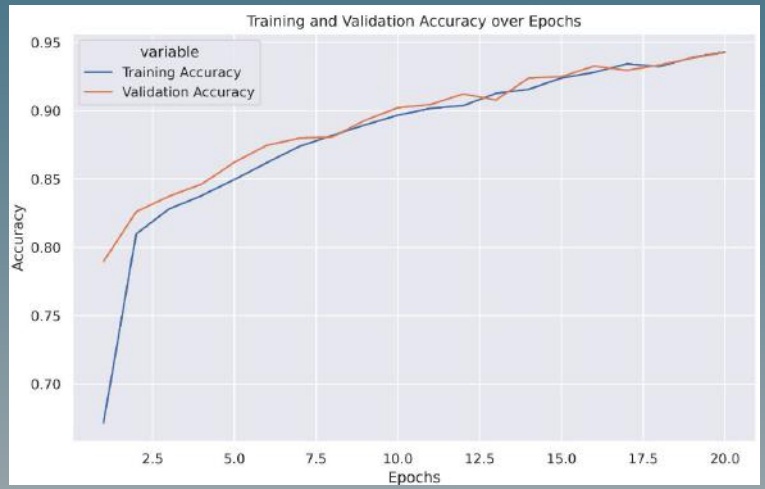
4.

➔ Training and Validation **loss** over epochs




```
# Visualize the loss during training
# ------------------------------------------------------------------------------
plt.plot(history.history["loss"], color="b", label="Training Loss")
plt.plot(history.history["val_loss"], color="r", label="Validation Loss")
plt.ylabel("Loss")
plt.xlabel("Epochs")
plt.legend()
plt.show()
```

➔ Training and Validation **accuracy** over epochs




```
# Use Seaborn to visualize traing and validation curves
# ------------------------------------------------------------------------------
plt.figure(figsize=(10, 6))
sns.lineplot(x='Epochs', y='value', hue='variable', data=pd.melt(accuracy_data, ['Epochs']))
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Training and Validation Accuracy over Epochs')
plt.show()
```

Recurrent Neural Network 🧠

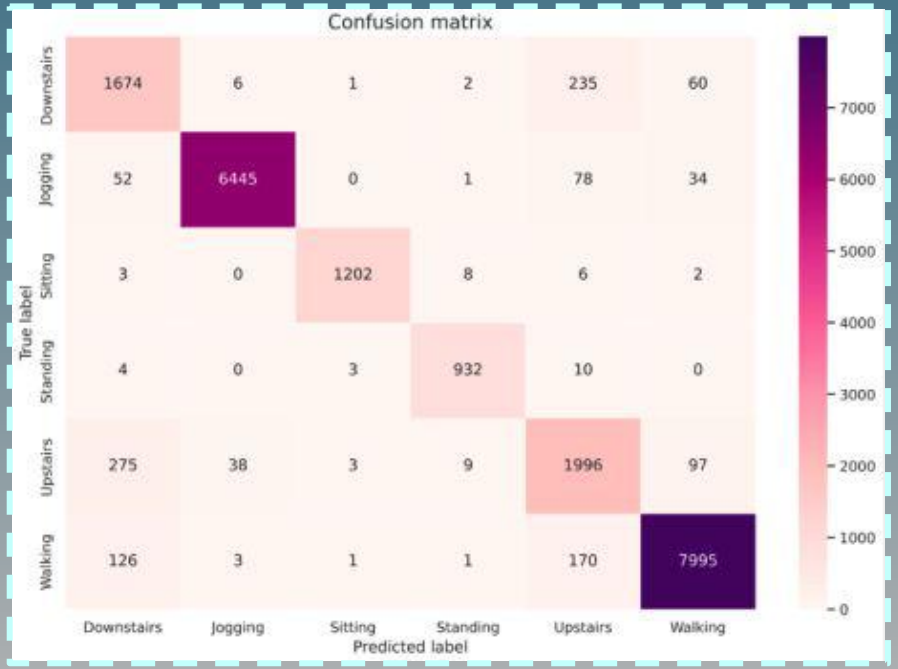
LSTM Network

- Performances 🔍

➔ Weighted F1 Score obtained

```
F1-score for Downstairs: 0.8042
F1-score for Jogging: 0.9826
F1-score for Sitting: 0.9905
F1-score for Standing: 0.9825
F1-score for Upstairs: 0.8092
F1-score for Walking: 0.9674

Weighted F1-score: 0.9412
```

➔ Confusion Matrix of the model



Logistic Regression

Modèle de Machine Learning

Supervisé

Logistic Regression 🤖

Supervised model

- Train and test sets 🔍

1.
```
# Creation of two dataframes : train and test (users dispastch)
# ------------------------------------------------------------------------------
# train data -> Users upto User ID = 27 (i.e. 27 users)
df_train = df_acc[df_acc['user'] <= 27]
# test data -> Users from User ID = 28 to 36 (i.e. 9 users)
df_test = df_acc[df_acc['user'] > 27]
```

   - ➔ We do not divide with Train Test Split
   - ➔ We do it manually

Eugenia **Mellano** & Doriane **Doyon-Dousse**

Logistic Regression 🤖

Supervised model

- **X_train** & **X_test** preprocessing 🔍

Statistical characteristics in the time domain for x, y and z

Averages (classic & absolute)

Standard deviation

Minimum

Maximum

2.

```python
# Function to detect peaks in a time series
def detect_peaks(data):
    peaks_indices = find_peaks(data)[0]
    if len(peaks_indices) < 2:
        return np.nan
    time_between_peaks = np.diff(peaks_indices) * 10  #Convert from indices to milliseconds (each index represents 10 ms)
    return np.mean(time_between_peaks)  # Returns average time between peaks

# Lists to store features and labels
x_list = []
y_list = []
z_list = []
train_labels = []
time_between_peaks_x = []
time_between_peaks_y = []
time_between_peaks_z = []

window_size = 100
step_size = 25

# Creation of window-size (200)
for i in range(0, df_train.shape[0] - window_size, step_size):
    xs = df_train['x-axis'].values[i: i + window_size]
    ys = df_train['y-axis'].values[i: i + window_size]
    zs = df_train['z-axis'].values[i: i + window_size]
    label = df_train['activity'][i: i + window_size].value_counts().idxmax()  # Obtener la actividad más común

    # Calculate time between peaks for each axis
    time_between_peaks_x.append(detect_peaks(xs))
    time_between_peaks_y.append(detect_peaks(ys))
    time_between_peaks_z.append(detect_peaks(zs))

    x_list.append(xs)
    y_list.append(ys)
    z_list.append(zs)
    train_labels.append(label)

# Statistical characteristics in the time domain for x, y and z
X_train = pd.DataFrame()

# Average calculation for each axis
X_train['x_mean'] = pd.Series(x_list).apply(lambda x: x.mean())
X_train['y_mean'] = pd.Series(y_list).apply(lambda x: x.mean())
X_train['z_mean'] = pd.Series(z_list).apply(lambda x: x.mean())
```

Logistic Regression 🤖

Supervised model

- Train & test sets répartition ✂️

3.

```
[56] # Y train and test definition
     # ----------------------------------------
     y_train = np.array(train_labels)
     y_test = np.array(test_labels)
```

```
# Remove 'most_common_activity' column from X_train and X_test
# ------------------------------------------------------------
X_train = X_train.drop(columns=['most_common_activity'])
X_test = X_test.drop(columns=['most_common_activity'])
```

➔ Drop of most common activity (created col) to not bias data

Logistic Regression 🤖

3.

Supervised model

- Preprocessing values 🛠️

4.

```
# Preprocessing of values before applying the model
# ----------------------------------------------------------------------------
encoder = OneHotEncoder(drop='first')
scaler = StandardScaler()

y_train_encoded = encoder.fit_transform(y_train.reshape(-1, 1))
y_test_encoded = encoder.transform(y_test.reshape(-1, 1))


scaler.fit(X_train)
X_train_data_lr = scaler.transform(X_train)
X_test_data_lr = scaler.transform(X_test)
```

➔ No missing values so no need to do imputations

➔ We just instantiate OneHotEncoder() & StandardScaler()

Logistic Regression 🤖

3.

Supervised model

- The model 🚀

➔ Classical model of Logistic Regression

5.

```
# Logistic regression with scores and classification reports
# ------------------------------------------------------------------------------
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.model_selection import GridSearchCV

lr = LogisticRegression()

lr = LogisticRegression(random_state = 21,max_iter=2000)

lr.fit(X_train_data_lr, y_train)

y_pred = lr.predict(X_test_data_lr)

print("Accuracy:", accuracy_score(y_test, y_pred))

print("\n ------------- Classification Report -------------\n")
print(classification_report(y_test, y_pred))
print("\n ------------------------------------------------\n")
```

Logistic Regression 🤖

Supervised model

- Performances 🔍

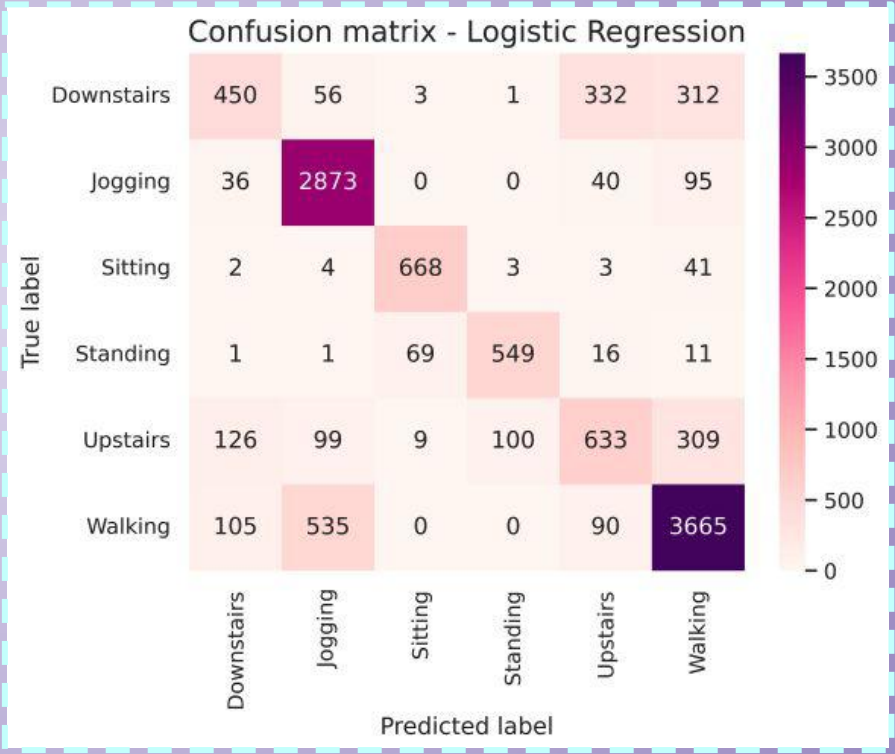
➔ Classification report

7 .

```
Accuracy: 0.7865088546765151

------------- Classification Report -------------

              precision    recall  f1-score   support

  Downstairs       0.62      0.39      0.48      1154
     Jogging       0.81      0.94      0.87      3044
     Sitting       0.89      0.93      0.91       721
    Standing       0.84      0.85      0.84       647
    Upstairs       0.57      0.50      0.53      1276
     Walking       0.83      0.83      0.83      4395

    accuracy                           0.79     11237
   macro avg       0.76      0.74      0.74     11237
weighted avg       0.78      0.79      0.78     11237


-------------------------------------------------
```

➔ Confusion Matrix of the model

6 .



Adaboost Classifier

Combined with Logistic Regression

Ensemble

Eugenia **Mellano** & Doriane **Doyon-Dousse**

Adaboost classifier w/ Logistic Regression ⚙️

Ensemble technic

- The model 🚀 & Performances 🔍

➔ Code for launching the algorithm

```
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix
from sklearn.ensemble import AdaBoostClassifier
from sklearn.model_selection import GridSearchCV

print("Grid search...")
print()
logistic_regression = LogisticRegression(max_iter = 1000)
adaboost_logreg = AdaBoostClassifier(logistic_regression)

# Grid of values to be tested
adaboost_logreg_params = {
    'base_estimator__C': [0.01, 0.05, 0.1, 0.5, 1.0, 5.0, 10.0, 50.0],
    'n_estimators': [5, 10, 20, 40, 60, 80, 100]
}

gridsearch = GridSearchCV(adaboost_logreg, param_grid = adaboost_logreg_params, cv = 3,
gridsearch.fit(X_train_data_lr, y_train)
print("...Done.")

print()
print("Best hyperparameters : ", gridsearch.best_params_)
print("Best validation accuracy : ", gridsearch.best_score_)
print()

GS_Y_test_pred = gridsearch.predict(X_test_data_lr)
print('Predictions on the test set : ', GS_Y_test_pred)
```

Raison décroissance des performances d'AdaBoost ? 📉

- Inadéquation du modèle donné à ingéré
- Plutôt qu'une Logistic Regression : un RNN / CNN aurait été plus adapté
- Problématique : **RNN combiné avec Adaboost** nécessite beaucoup de capacité computationnelle

➔ Classification report of the model

---------- Classification Report ----------

	precision	recall	f1-score	support
Downstairs	0.73	0.22	0.34	1154
Jogging	0.82	0.92	0.87	3044
Sitting	0.99	0.90	0.94	721
Standing	0.92	0.93	0.92	647
Upstairs	0.53	0.32	0.40	1276
Walking	0.71	0.87	0.78	4395
accuracy			0.76	11237
macro avg	0.78	0.69	0.71	11237
weighted avg	0.75	0.76	0.73	11237

Axes d'amélioration potentiels

Axes d'améliorations 🚀

- **Le positionnement de l'accéléromètre**
  - Positionner l'accéléromètre sur la hanche (téléphone dans la poche) est ce suffisant ?
    - Mettre en relation une **Montre connectée** pourrait améliorer les prédictions
  -
- **Montre connectée :**
  - Grâce à la **mesure de la fréquence cardiaque**
    - Mise en relation entre application téléphonique + la montre connectée
    - **Communication entre les deux outils** :
      - Ajouterait une **variable**
      - Augmenterait les **performances algorithmique**
      - **Meilleure reconnaissance d'activité** physique
  -
- **Reconnaître plus d'activités que les 6 mentionnées :**
  - On pourrait également apprendre à l'algorithme à :
    - **Détecter d'autres activités physique**
      - **Velo / Bicyclette**
      - **Jardinage**
      - **Rameur / Aviron**

Source image : https://www.apple.com/fr/shop/buy-watch?afid=p238%7Cs5Fq4AmPl-dc_mtid_187079nc38483_pcrid_616231905301_pgrid_24143240124_pntwk_g_pchan__pexid__&cid=aos-fr-kwgo-watch--slid--bran-product-

Axes d'améliorations 🚀

- **Obtenir un échantillon plus grand**
  - En statistique on sait bien que :
    - Plus l'échantillon est grand meilleur est la qualité des données exploitées
    - Avoir plus que 36 'users' = **meilleurs performances de l'algorithme**

- **Tester different positions du téléphones lors de la récolte de données**
  - Avoir le téléphone dans la poche avant apporte un **biais de mesure** :
    - L'utilisateur le met du **côté droit ou gauche** :
      - **Si une asymétrie corporelle est présente alors les données varient**
  - Tester un positionnement au niveau du **centre de gravité**
    - Pourrait potentiellement **augmenter la qualité des mesures** prises

Source image : https://www.google.com/search?sca_esv=61f1a3f0cdc4b4d1&rlz=1C5CHFA_enFR1078FR1078&q=personnes+images&tbm=isch&source=lnms&sa=X&ved=2ahUKEwiMlZeR-JuFAxU8QaQEHZY0Dq0Q0pQJegQIDBAB&biw=1920&bih=959

Conclusion ✅

LSTM 🧠

- BEST MODEL ➡️ **F1 score = 0.94**
- From the confusion matrix :
  - Model hardly detect ascending or descending stairs
  - For the rest of activities : excellent performances

Logistic Regression 🤖

- Requires much more significant feature engineering
- Compared to the LSTM model : do not achieve as good results. **F1 score 0.78**

AdaBoost Classifier with LogReg (ensemble) ⚙️

- Takes a lot of computation time
- Do not achieve significant results. **F1 Score = 0.73**
- Less interesting performances from all the above models.

Merci beaucoup! ✨

Nous vous remercions pour votre attention.