INSTITUT FRANCOPHONE INTERNATIONAL

ĐẠI HỌC QUỐC GIA HÀ NỘI
ĐHQGHN VNU Since 1906

MASTER II - SYSTEMES INTELLIGENTS ET MULTIMEDIA (SIM)

TRAVAUX PRATIQUES MODULE APPRENTISSAGE

Human Activities Recognition (HAR)

Etudiants :
- KANA NGUIMFACK Kévin
- MBIAYA KWUITE Franck Anael
- OUEDRAOGO Wend-Panga Jérémie
- NGUYEN Truong Thinh

Encadreur :
Lê Hồng Phương

26th January 2021

Table of Contents

List of Tables

List of Figures

1 Introduction

Dans le cadre de notre module d'apprentiissage, il est question pour nous de transposer nos connaissances théoriques et pratiques acquises, dans la mise en oeuvre de notre mini-projet. Nous traitons dans ce contexte **la reconnaissance des activités humaines**.

La base de données que nous utilisons dans notre travail, est une base de données de reconnaissance d'activités humaines construite à partir des enregistrements de **30 sujets** effectuant des activités de la vie quotidienne tout en portant un **smartphone (Samsung Galaxy S II)** monté sur la taille avec des capteurs inertiels intégrés.

Les signaux du capteur (**accéléromètre** et **gyroscope**) ont été prétraités en appliquant des filtres de bruit, puis échantillonnés dans des fenêtres coulissantes de largeur fixe de 2,56 s et chevauchement de 50% (128 lectures / fenêtre). Le signal d'accélération du capteur, qui a des composants gravitationnels et de mouvement corporel, a été séparé à l'aide d'un filtre passe-bas Butterworth en accélération corporelle et gravité. A partir de chaque fenêtre, un vecteur de caractéristiques a été obtenu en calculant des variables du domaine temporel et fréquentiel.

Concernant notre dataset, nous rappelons que les expériences ont été menées par des volontaires dans une tranche d'âge de 19 à 48 ans. Chaque personne a réalisé **six activités** (marche, debout, pose, assise, marche montante et descente des escaliers). L'ensemble de données obtenu a été partitionné au hasard en deux ensembles, où 70% des volontaires ont été sélectionnés pour générer les données d'entraînement et 30% les données de test.

2 Objectifs du travail

Dans le cadre de ce travail, il est question pour de mettre en oeuvre un ou plusieurs modèle(s) ou hypothèse(s) pour apprendre à partir des données à fin de prédire une activité pour une observation.

Pour cela notre démarche se décline comme suite:

- La description et préparation des données
- La mise en oeuvre des hypothèes (modèles)
- L'analyse et interprétations des résultats

3 Description et préparation des données

Deux composants essentiels intégrés dans un smartphone, ont été utilisés pour l'acquisition des données du dataset, **l'accéléromètre** qui est un capteur capable de mesurer, en trois dimensions, les accélérations linéaires d'un sujet, permettant de détecter l'orientation du smartphone, mais aussi sa mise en mouvement. **Le gyroscope** à 3 axes permet de même déterminer l'orientation du smartphone. Tandis que l'accéléromètre mesure les mouvements de translation, le gyroscope mesure les mouvements de rotation.

3.1 Description des données

Les données collectées sont comprises dans les domaines temporels et fréquentiels.
Ils concernent 6 activités (labels, sauvegardés dans le fichier **activity_labels.txt**):

— **WALKING**, le sujet effectue une marche.
— **WALKING_UPSTAIRS**, le sujet monte les marches d'escaliers.
— **WALKING_DOWNSTAIRS**, le sujet descent les marches d'escaliers.
— **SITTING**, le sujet assis sur une chaise.
— **STANDING**, le sujet se tient debout.
— **LAYING**, le sujet est couhé.

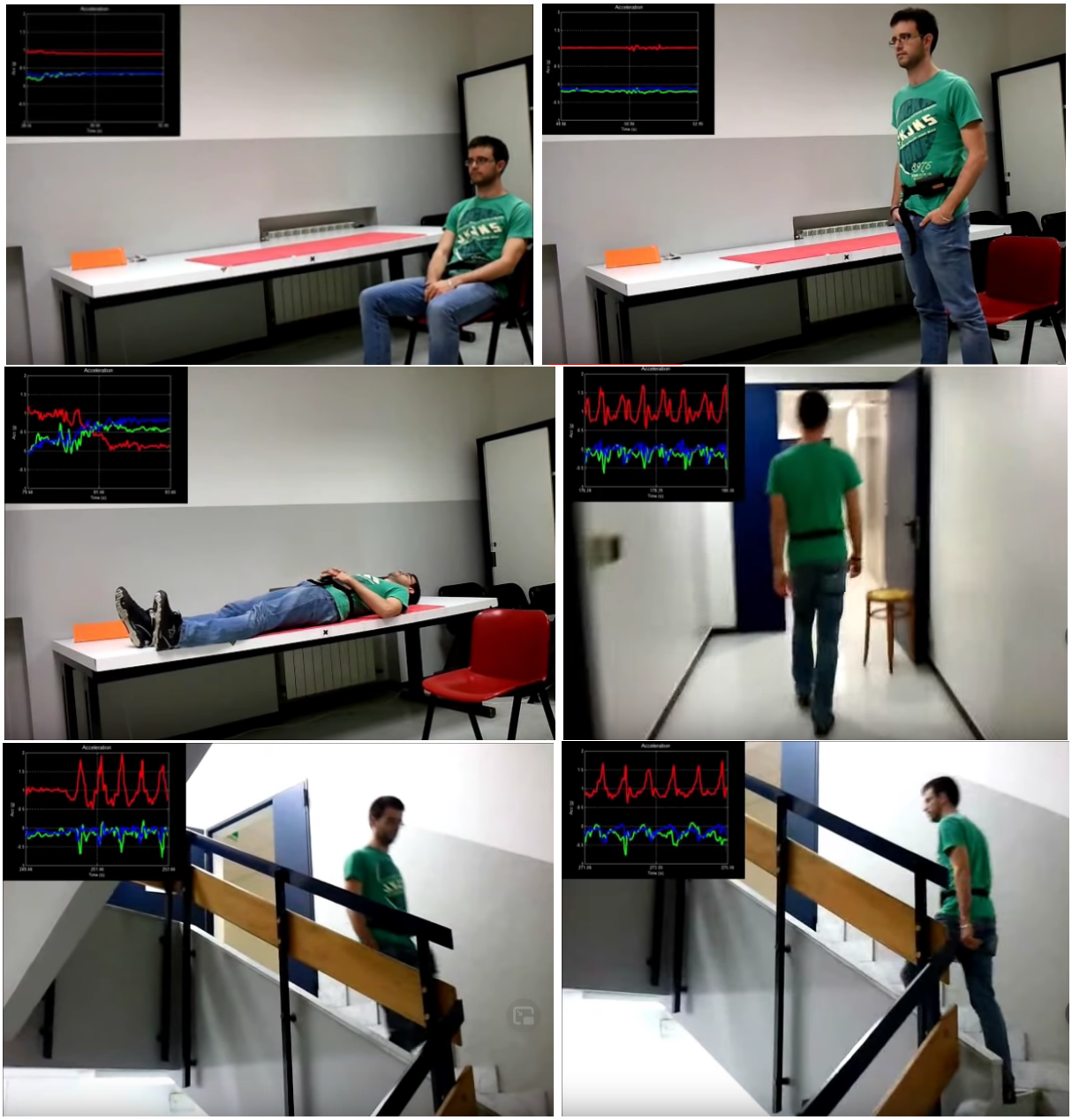

Figure 1 – Processus de construction du dataset à partir des sujets

Les signaux du domaine temporel ont été préfixe «**t**» pour désigner le temps, **tAcc-XYZ et tGyro-XYZ** representé sur trois axes (X, Y et Z). le signal d'accélération a ensuite été séparé en signaux d'accélération corporelle et gravitationnelle **tBodyAcc-XYZ et tGravityAcc-XYZ**.

L'accélération linéaire du corps et la vitesse angulaire ont été dérivées dans le temps pour obtenir des signaux Jerk, **tBodyAccJerk-XYZ et tBodyGyroJerk-XYZ**. L'amplitude de ces signaux tridimensionnels a également été calculée en utilisant la norme euclidienne, **tBodyAccMag, tGravityAccMag, tBodyAccJerkMag, tBodyGyroMag, tBodyGyroJerkMag**.

Les signaux du domaine fréquentiel ont été préfixe **«f»** pour désigne la fréquence. Une transformation de Fourier rapide (FFT) a été appliquée à certains de ces signaux produisant **fBodyAcc-XYZ, fBodyAccJerk-XYZ, fBodyGyro-XYZ, fBodyAccJerkMag, fBodyGyroMag, fBodyGyroJerkMag**.

Nous notons que ces signaux ont été utilisés pour estimer les variables du vecteur de caractéristiques pour chaque motif: **«-XYZ»** est utilisé pour désigner des signaux à 3 axes dans les directions X, Y et Z.

L' ensemble des variables, soit **561 variables**,ont été estimées à partir de ces signaux de base soit **17 variables**, par application d'un ensemble d'opérations sont, comme obeservé dans le fichier **features_info.txt**:

- mean (): valeur moyenne
- std (): écart type
- mad (): écart absolu médian
- max (): plus grande valeur du tableau
- min (): plus petite valeur du tableau
- sma (): zone de magnitude du signal
- energy (): mesure d'énergie. Somme des carrés divisée par le nombre de valeurs.
- iqr (): Intervalle interquartile
- entropie (): entropie du signal
- arCoeff (): coefficients d'autorégression avec ordre de Burg égal à 4
- corrélation (): coefficient de corrélation entre deux signaux
- maxInds (): indice de la composante fréquentielle de plus grande amplitude
- meanFreq (): moyenne pondérée des composantes de fréquence pour obtenir une fréquence moyenne
- skewness (): asymétrie du signal du domaine fréquentiel
- kurtosis (): kurtosis du signal du domaine fréquentiel
- bandesEnergy (): énergie d'un intervalle de fréquence dans les 64 bins de la FFT de chaque fenêtre.
- angle (): Angle entre les vecteurs.

L'ensemble de variables synthétisées ont été sauvegardé dans le fichier **'features.txt'**.

Alors, nous travaillerons sur un ensemble de données normalisées (compris entre -1 et 1), **Multivarié, série chronologique** avec **10299 instances** et **561 attributs**, organisé dans de fichiers textes.

Figure 2 – Repertoire dataset

3.2 Préparation des données

Pour la mise en oeuvre de nos hypothèses, il est important de préparer nos données pour l'entrainement et le test.

Toutefois notre dataset déjà fourni avec un ensemble d'entrainement (70% de sujets ayant contribués à la mise en place du train) et un ensemble de test (30% de sujets ayant contribués à la mise en place du test), organisés comme suite:

- **train/X_train.txt**: ensemble d'entraînement.
- **train/y_train.txt**: étiquettes de formation.
- **test/X_test.txt**: ensemble de tests.
- **test/y_test.txt**: étiquettes de test.
- **train/subject_train.txt** et **test/subject_test.txt**: chaque ligne identifie le sujet (**compris entre 1 et 30**) qui a réalisé l'activité pour chaque échantillon de fenêtre.
- **train/Inertial Signals/total_acc_x_train.txt** et **test/Inertial Signals/total_acc_x_test.txt**: Le signal d'accélération de l'axe X de l'accéléromètre du smartphone en unités de gravité standard **'g'**. Chaque ligne montre un vecteur de 128 éléments. La même description s'applique aux fichiers **«total_acc_x_train.txt»**, **«total_acc_x_test.txt»** **«total_acc_z_train.txt»** et **«total_acc_z_test.txt»** pour les axes Y et Z.
- **train/Signaux inertiels/body_acc_x_train.txt** et **test/Signaux inertiels/body_acc_x_test.txt**: Le signal d'accélération corporelle obtenu en soustrayant la gravité de l'accélération totale.
- **train/Inertial Signals/body_gyro_x_train.txt** et **test/Inertial Signals/body_gyro_x_test.txt**: Le vecteur de vitesse angulaire mesuré par le gyroscope pour chaque échantillon de fenêtre. Les unités sont en **radians / seconde**.

Afin de faciliter, notre travail par la suite, nous avaons fusionné les données des **sujets**, les **labels**, et les **données de features** respectivement pour le train et le test, sauvegardées dans les fichiers **X_train_new.txt** de taille **7352 X 563** et **X_test_new.txt** de taille **2947 X 563**, pour nos travaux.
Le fichier notebook **Comprehension_des_donnees.ipynb** implémente ce processus.

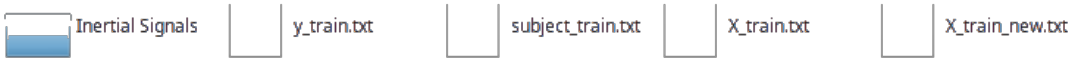

Figure 3 – Repertoire ensemble entrainement

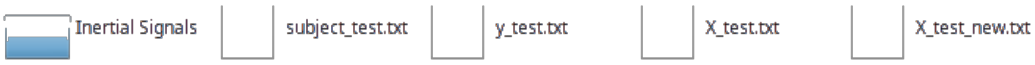

Figure 4 – Repertoire ensemble test

4 Mise en oeuvre des hypothèses (modèles)

Pour la mise en oeuve, nous utilisons deux languages de programmation **Python** et **Julia**.

Compte tenu, du nombre de variables extrêmement élevé, soit 561 variables, notre solution se décline comme suite:

- **Analyse des composantes principales**, dans un premier temps à fin de réduire le nombre de variables, à **155 variables**.
- **Reconnaissance des activités**, un ensemble d'hypothèses a été utilisé, notamment:
  - les machines à vecteurs supports (SVM)
  - les k voisins les plus proches (KNN)
  - les arbres de décision (DT)
  - la regression logistique (LOG)
  - Les réseaux de neurones artificielles (ANN)

4.1 Analyse des composantes principales (PCA)

l'analyse des composantes principales, ou principal component analysis (PCA) en anglais, est une méthode d'apprentissage non supervisée, permettant d'analyser et de visualiser un jeu de données contenant des individus décrits par plusieurs variables quantitatives.

La méthode consiste à trouver une autre représentation du jeu de données, tout en concervant au maximun les informations utiles, en éliminant les bruits (redondances, dependances entre les features).

Il s'agit donc, de truver une transformation Θ qui représente un jeu de données X défini dans un espace supérieur vers une nouvelle représentation Z dans un espace plus réduit, tel que: $X\Theta = Z$.
Où les vecteurs $\theta_1, \theta_2, ..., \theta_D$ sont les composantes principales de X.

$$\begin{pmatrix} - & \mathbf{x}_1 & - \\ - & \mathbf{x}_2 & - \\ \dots & \dots & \dots \\ - & \mathbf{x}_N & - \end{pmatrix} \begin{pmatrix} | & | & \vdots & | \\ \theta_1 & \theta_2 & \vdots & \theta_D \\ | & | & \vdots & | \end{pmatrix} = \mathbf{Z}$$

Nous appliquons cette methode, tant sur le train que sur le test.
Par application sur le train , nous avons:

- Notre jeu de données X, de taille **7352 X 561**.
- Notre matrice de transformation Θ, de taille **561 X 155**.
- Notre nouvelle représentation du jeu de données Z, de taille **7352 X 155**

Par application sur le test , nous avons:

- Notre jeu de données X, de taille **2947 X 561**.
- Notre matrice de transformation Θ, de taille **561 X 155**.
- Notre nouvelle représentation du jeu de données Z, de taille **2947 X 155**

La détermination préalable, de la transformation Θ suit les étapes suivantes:

- **Centrage des données**:
  - Pour chaque colonne, on calcule la moyenne.
  - Puis on soustraire à chaque valeur, la moyenne, et on divise par le nombre de valeurs.
- **Construire la matrice de covariance** $\sum = cov(\overline{X})$, où $\overline{X}$ représente le centrage des données.
- **Décomposer de la matrice de covariance** en vecteurs propres,valeurs propres v_i, λ_i
- **Ordonner les valeurs propres par ordre décroissant**
- **Recherche du nombre de composantes** en conservant au moins 99% d'informations.

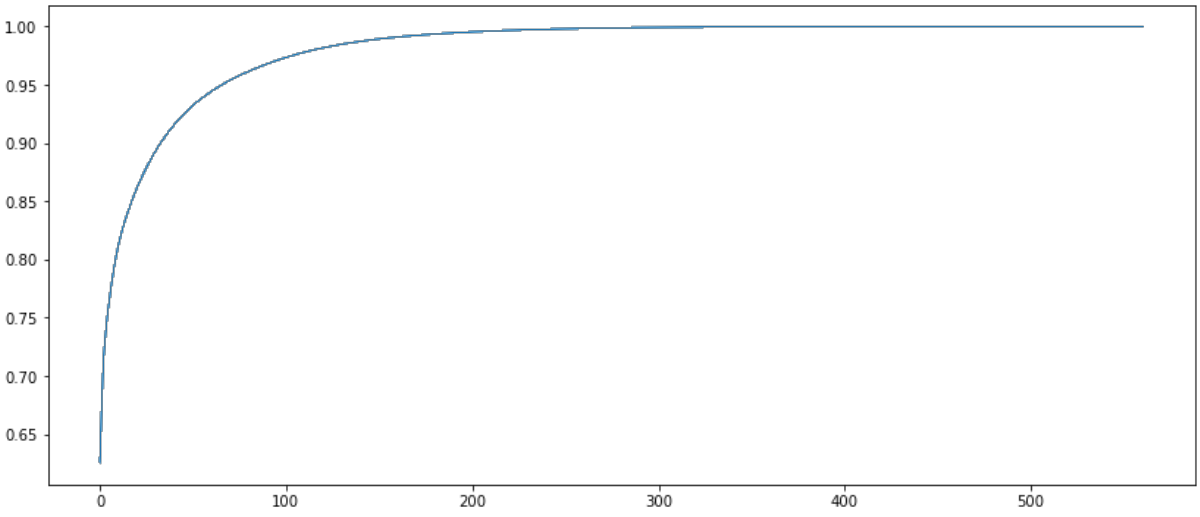

Figure 5 – Courbe de recherche du nombre de composantes, en conservant 99% d'informations

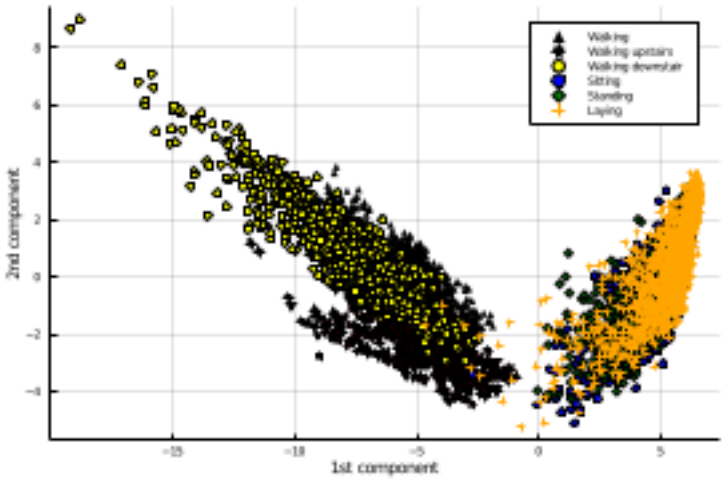

Figure 6 – Visualisation 1er et 2nde composantes

Nous notons que, pour chacune de nos hypothèses, nous procédons au préable par application de l'analyse de composents principales, avant de faire la classification.

4.2 Reconnaissance des activités humaines

Pour notre travail, nous utiliserons un ensemble de bibliothèques:

- En Julia:
  - **DelimitedFiles** : pour lire une matrice à partir de la source où chaque ligne (séparée par eol) donne une ligne, avec des éléments séparés par le délimiteur donné.
  - **Plots**: pour la visualisation.
  - **Statistics**: pour les calculs statistiques.
  - **LinearAlgebra**: pour les opérations algébriques.
  - **Flux: onehotbatch**: pour l'encodage des labels.
  - **Flux: @epochs**: pour définir le nombre d'entrainement.
  - **PlutoUI**: pour l'affichage de la progression de l'entrainement.
- En Python:
  - **pandas, numpy**: pour la manipulation des données.
  - **seaborn et matplotlib**: pour la visualisation.
  - **sklearn**: pour le machine learning

NB: Le repertoire PDF, comporte l'ensemble de code de nos modèles (5 hypothèses) au format pdf.

4.2.1 Réséaux de neurones artificielles (ANN)

Les réseaux de neurones ont été utilisés pour la reconnaissance des activités, implementés en language julia.
Plusieurs paramètres, ont été utilisés:

$$J(\mathbf{x}, y; \theta, b) = -\left[y \log(h_{\theta,b}(\mathbf{x})) + (1-y) \log(1 - h_{\theta,b}(\mathbf{x}))\right]$$

— **la fonction de perte**: cross entropy.

— **l'optimiseur**: Adam (learning rate = 0.001, La décroissance exponentielle de l'estimation du premier moment $\beta_1 = 0.9$ et du second moment $\beta_2 = 0.999$).

Nous avons expérimenté plusieurs modèles à couches variables, la fonction de train utilisée est la suivante:

```
# loss function
loss(x, y) = Flux.crossentropy(model(x), y)

# train fuction
function train(numEpochs=20)
    with_terminal() do
        @epochs numEpochs Flux.train!(loss, _rnn, [(X_train_acp', Y_train)],
        optimizer; cb = () -> println(loss(X_train_acp'[:,1:100],
        Y_train[:,1:100])))
    end
end

# accuracy function
accuracy(x, y) = mean(Flux.onecold(model(x)) .== y)
```

Nous présentons ci-dessous, les performances obtenues:

Modèles ANN			
Nombre Couches	**Nombre epochs**	**train accuracy**	**Test accuracy**
2	500	98.82%	94.2%
3	500	99.17%	95.15%
5	300	99.10%	95.52%

Table 1 – Performance modèle ANN

Les fichiers soucres sont respectivement **ACP_RNN_2_couches_500_epochs.jl**, **ACP_RNN_3_couches_500_epochs.jl** et **ACP_RNN_5_couches_300_epochs.jl**.

Nous constatons que le modèle généralise mieux, la performance du modèle, c'est améliorée avec l'augmentation des couches.
Le meilleur modèle est **ANN 5 couche**, avec **99.10%** sur l'ensemble d'entrainement et **95.52%** sur l'ensemble de test.

4.2.2 la regression logistique (LOG)

Pour la classification des activités, nous utilisons également le modèle de regression logistique.

Afin de mieux garantir la performance du modèle, nous utilisons une grille de recherche permettant d'entrainer, notre modèle avec les meilleurs paramères

($logisticregression\_C = 0.5$).
Nous rappelons ici, que le modèle a été mis en oeuvre en python.

Après évalution, nous obtenons une performance de **99.28%** sur l'ensemble d'entrainement et de **95.08%** sur l'ensemble de test.

La matrice de confusion, après évaluation est la suivante:

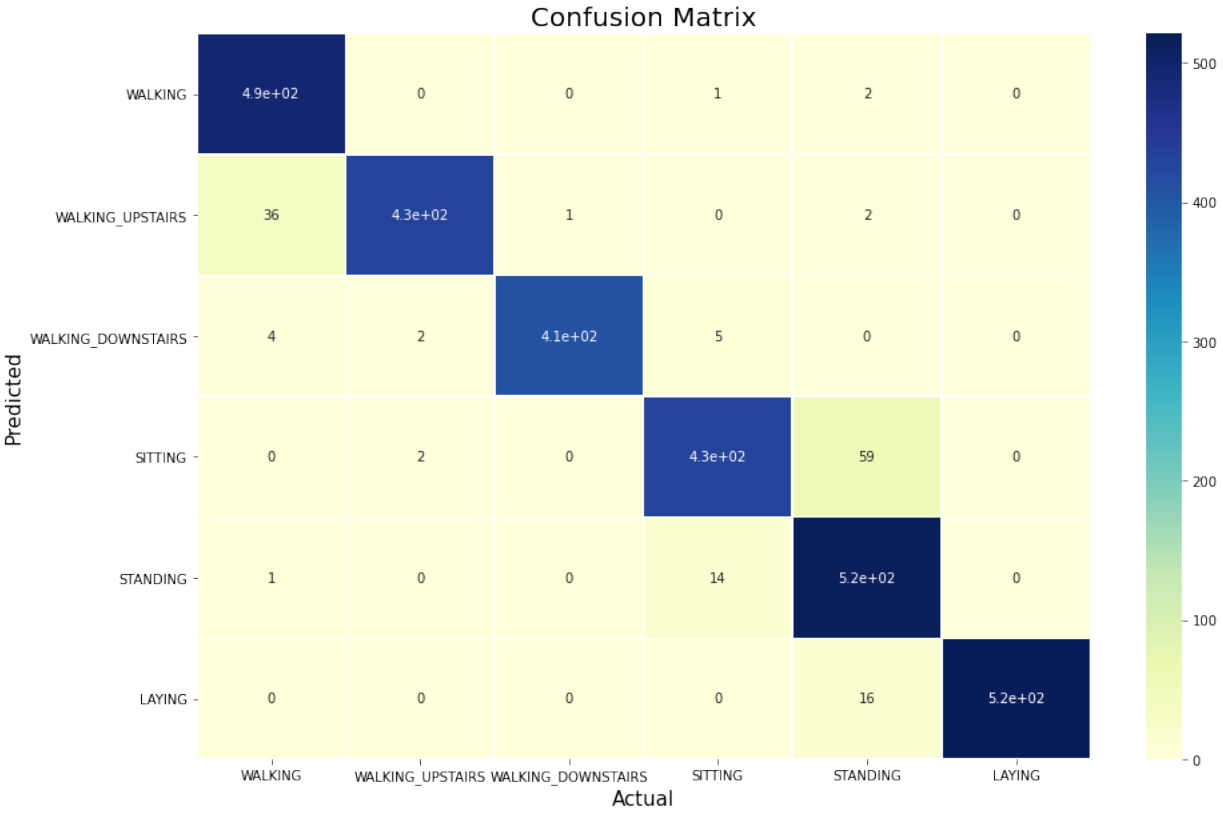

Figure 7 – Matrice de confusion du modèle de regression logistique

Le modèle obtenu, généralise mieux, avec une très bonne performance.
NB: fichier source, **PCA_RelLog_python.ipynb**.

4.2.3 les arbres de décision (DT)

Pour la classification des activités, nous utilisons également le modèle des arbres de decisions.

Afin de mieux garantir la performance du modèle, nous utilisons une grille de recherche permettant d'entrainer, notre modèle avec les meilleurs paramères ($decisiontreeclassifier\_\_criterion' : entropy$).
Nous rappelons ici, que le modèle a été mis en oeuvre en python.

```
# model pipeline
model_dt= make_pipeline(StandardScaler(), DecisionTreeClassifier())
# parameters
params = {'decisiontreeclassifier__criterion': ['gini', 'entropy']}
# gridsearch model
```

```
grid = GridSearchCV(model_dt, params, cv=5)
# training
grid.fit(X_train_pca, y_train)
```

Après évalution, nous obtenons une performance de **100%** sur l'ensemble d'entrainement et de **81.1%** sur l'ensemble de test.

La matrice de confusion, après évaluation est la suivante:

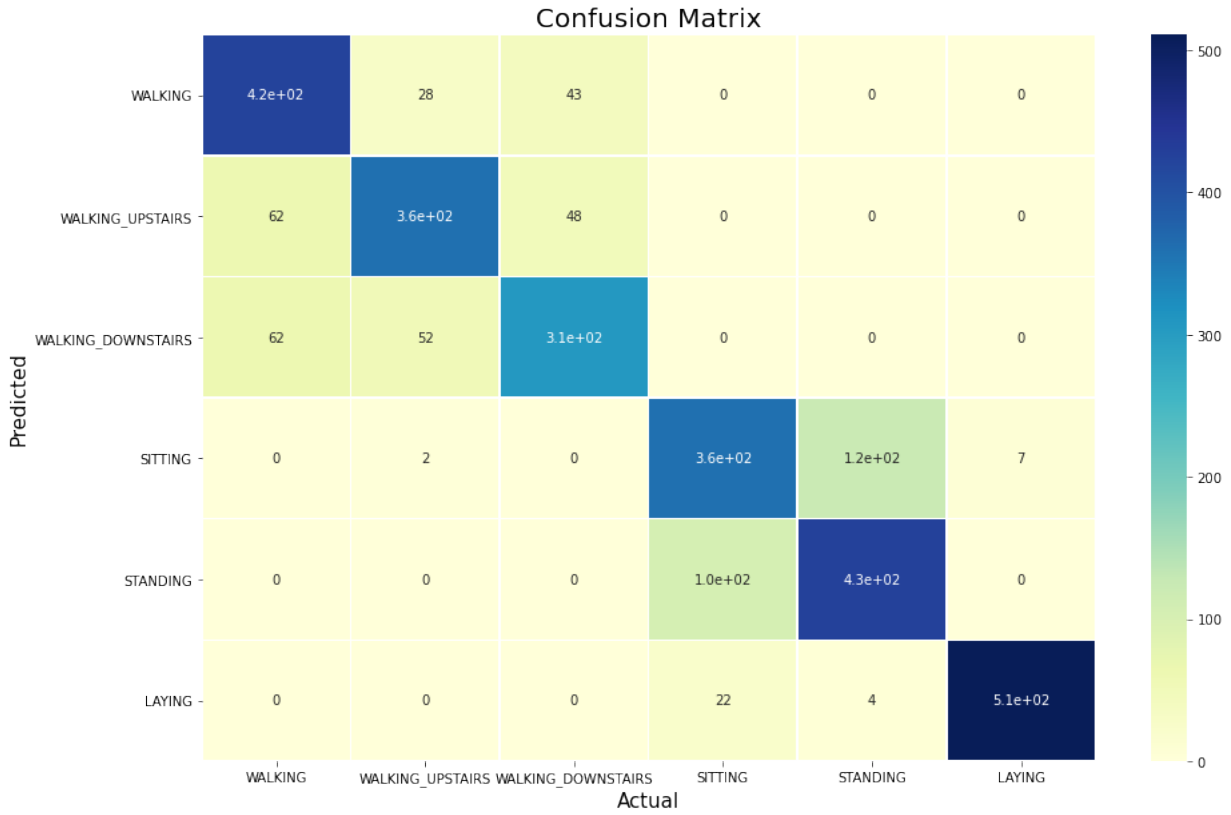

Figure 8 – Matrice de confusion du modèle d'arbres de décisions

Le modèle obtenu, généralise également mieux, avec une bonne performance.
NB: fichier source, **PCA_Decision_tree_python.ipynb**.

4.2.4 les k plus proches voisins (KNN)

Pour la classification des activités, nous utilisons également le modèle KNN.

Afin de mieux garantir la performance du modèle, nous utilisons également une grille de recherche permettant d'entrainer, notre modèle avec les meilleurs paramères ($kneighborsclassifier\_\_n\_neighbors : 3$).
Nous rappelons ici, que le modèle a été mis en oeuvre en python.

```
# model pipeline
model_knn= make_pipeline(StandardScaler(),KNeighborsClassifier(n_neighbors=3))
# parameters
params = {'kneighborsclassifier__n_neighbors': range(2,10)}
```

```
# gridsearch model
grid = GridSearchCV(model_knn, params, cv=5)
# training
grid.fit(X_train_pca, y_train)
```

Après évalution, nous obtenons une performance de **98.19%** sur l'ensemble d'entrainement et de **65.22%** sur l'ensemble de test.

La matrice de confusion, après évaluation est la suivante: Nous remarquons, que

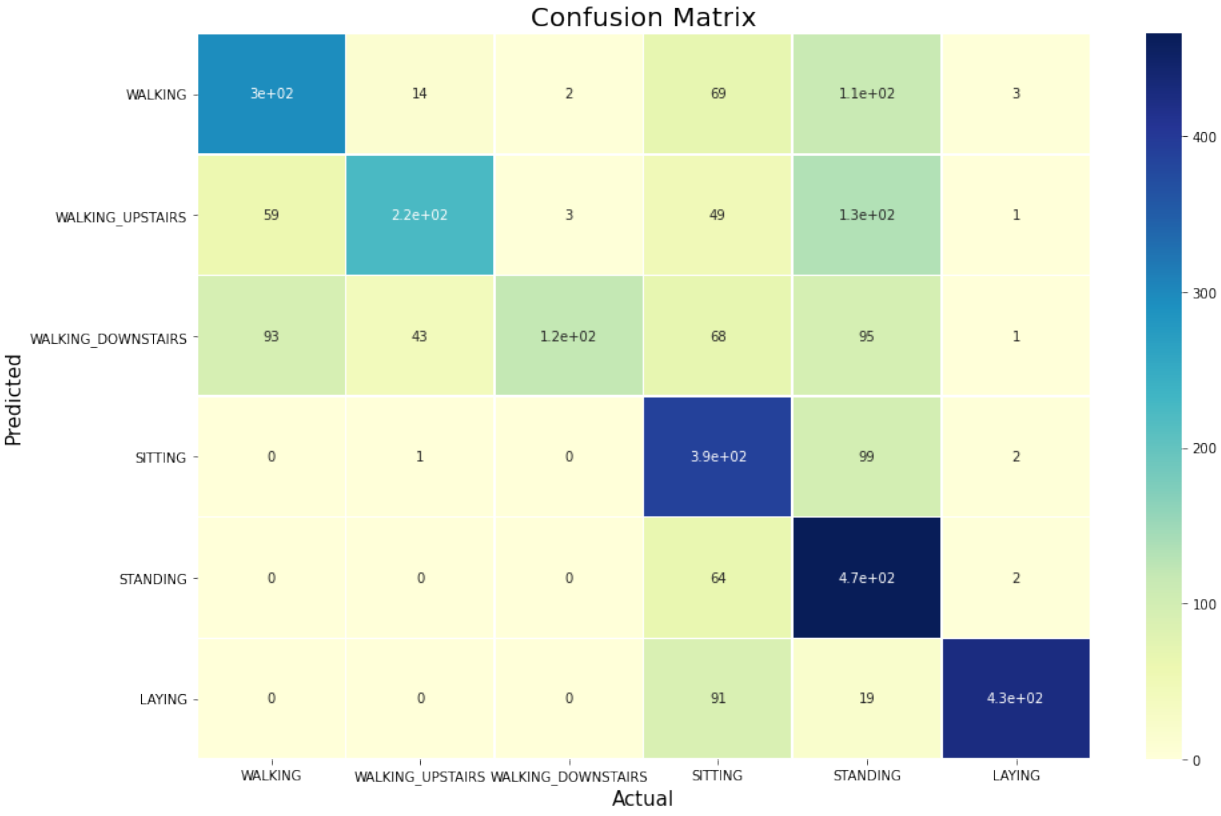

Figure 9 – Matrice de confusion du modèle KNN

le modèle se trompe beaucoup, d'ou une performance légèrement faible.
NB: fichier source, **PCA_KNN_python.ipynb**.

4.2.5 les machines à vecteurs supports (SVM)

Pour la classification des activités, nous utilisons également le modèle SVM.

Afin de mieux garantir la performance du modèle, nous utilisons une grille de recherche permettant d'entrainer, notre modèle avec les meilleurs paramères ($svc\_\_decision\_function\_shape : ovo, svc\_\_gamma' : auto, svc\_\_kernel : linear$).
Nous rappelons ici, que le modèle a été mis en oeuvre en python.

```
# model pipeline
model_svm= make_pipeline(StandardScaler(), SVC())
# parameters
```

```
params = {'svc__decision_function_shape': ['ovo', 'ovr'],
      'svc__gamma': ['auto', 'scale'],
      'svc__kernel': ['rbf', 'poly', 'linear']}
# gridsearch model
grid = GridSearchCV(model_svm, params, cv=5)
# training
grid.fit(X_train_pca, y_train)
```

Après évalution, nous obtenons une performance de **99.47%** sur l'ensemble d'entrainement et de **95.83%** sur l'ensemble de test.

La matrice de confusion, après évaluation est la suivante: Le modèle obtenu, généralise

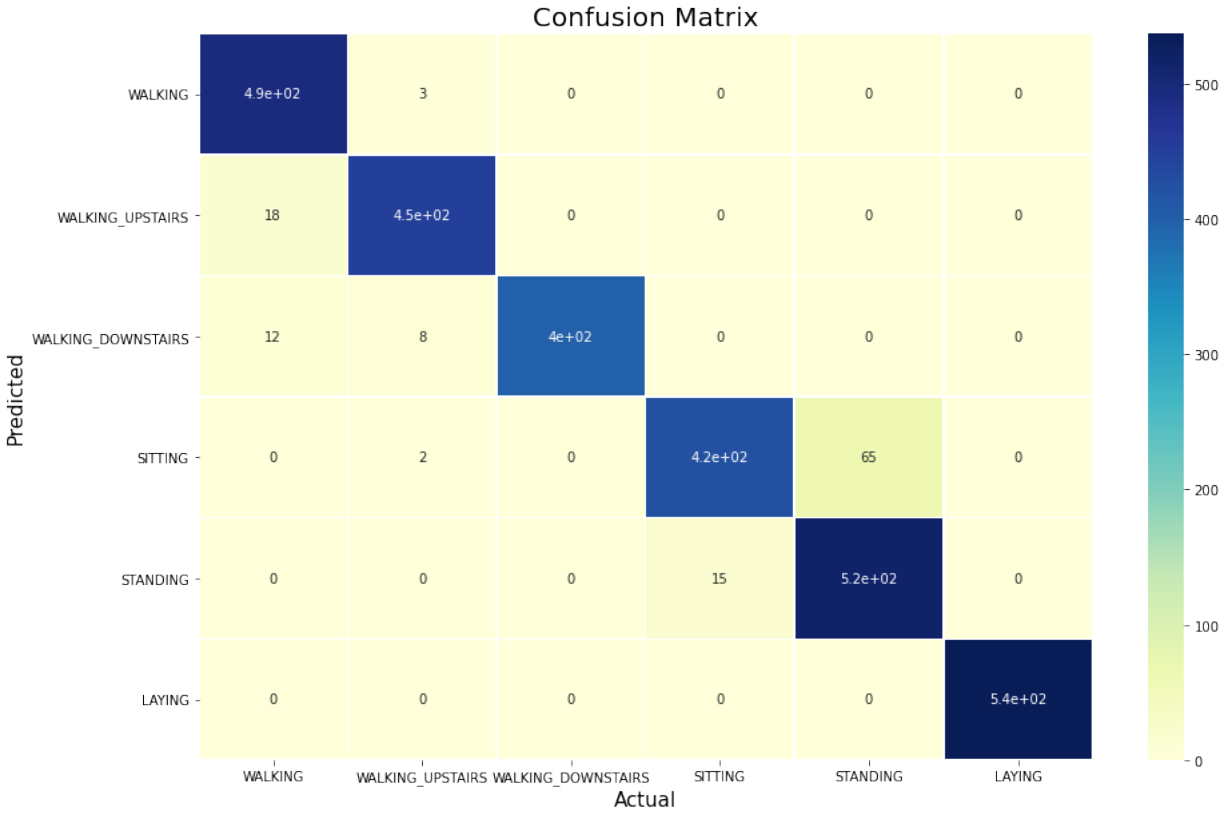

Figure 10 – Matrice de confusion du modèle SVM

mieux, avec une très bonne performance.
NB: fichier source, **PCA_SVM_python.ipynb**.

5 Analyse des résultats

Nous présonts ci-dessous, le bilan des hypothèes étudiées, pour notre tâche de reconnaissance des activités humaines.

Nous concluons que les modèles, **ANN**, **LOG**, **DT** et **SVM** généralisent les mieux sur l'ensemble de test.

Les meilleurs modèles, sont: **SVM** 95.83%, **ANN** 95.52%, et **LOG** 95.08%.

Résultats obtenus		
Modèles	**Train accuracy**	**Test accuracy**
ANN	99.10%	95.52%
LOG	99.28%	95.08%
DT	100%	81.1%
KNN	98.19%	65.22%
SVM	99.47%	95.83%

Table 2 – Performance modèle ANN

6 Conclusion

Parvenu au terme, de notre travail, concernant la reconnaissance d'activités (debout, assis, marche, pose, montée et descente d'escaliers), nous retonons 5 modèles, dont 4 parmi à savoir, les modèles **SVM**, **ANN**, **LOG** et **DT** ont donné des peformance meilleures en moyenne **95%**, mais sauf le modèle KNN avec une performance moyenne, soit **65.22%**.