Multimodal Processing, Recognition, and Interaction (MPRI)

Challenge

Auteur : Spinelli Isaia

Prof : Stefan Carrino

Simon Ruffieux et

Elena Mugellini

Carrino Francesco

Date : 22.12.2020

Salle : A2 – Lausanne

Classe : MPRI

Table des matières

[Introduction - 2 -](#_Toc58947439)

[Système général - 3 -](#_Toc58947440)

[Sauvegarde et chargement des données prétraitées - 4 -](#_Toc58947441)

[Affichage des informations concernant notre modèle et ses performances - 4 -](#_Toc58947442)

[Sauvegarde et chargement du meilleur modèle - 6 -](#_Toc58947443)

[Comparaison des améliorations - 7 -](#_Toc58947444)

[Amélioration des caractéristiques - 7 -](#_Toc58947445)

[Recherche des meilleurs paramètres - 8 -](#_Toc58947446)

[Comparaison des différents levels de segmentation - 9 -](#_Toc58947447)

[Évolution des performances - 10 -](#_Toc58947448)

[Conclusion - 11 -](#_Toc58947449)

[Difficultés rencontrées - 11 -](#_Toc58947450)

[Compétences acquises - 11 -](#_Toc58947451)

[Résultats obtenus - 11 -](#_Toc58947452)

[Annexe - 11 -](#_Toc58947453)

# Introduction

Ce challenge à pour objectif d’appliquer les concepts théoriques vus en classe. Il permet de travailler sur des données réelles et d’essayer d’améliorer au maximum la précision d’un modèle.

L'ensemble de données contient des données enregistrées dans le cadre de la thèse de doctorat de Quentin Meteier. Différents capteurs ont été utilisés pour enregistrer les données physiologiques lors des séances de conduite réalisé avec un simulateur de voiture. Les sujets expérimentaient une conduite autonome de niveau 3. Cela signifie qu'ils pouvaient effectuer une tâches secondaires (au lieu de conduire) mais devaient reprendre le contrôle du système lorsque des événements dangereux se produisaient (passage d'animaux, objets sur la route, etc.).

Il y a donc deux classes distinctes :

1. Conduire normalement sans tâche secondaire
2. Conduire en faisant une tâche secondaire

Pour plus d’informations concernant cet expérience, vous pouvez consulter l’énoncé du challenge en annexe [[1].](#_Annexe)

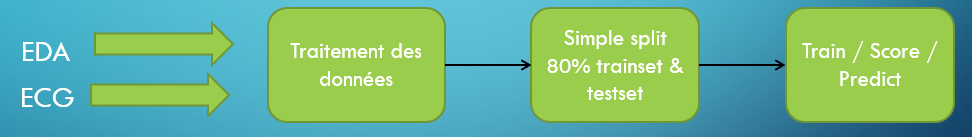
Afin de mener à bien ce challenge, voici dans l’ordre toutes les étapes qu’il a fallu effectuer :

1. Prise en main du code fourni
2. Survol des données fournies
3. Mise en place d’un système afin de choisir si on souhaite retraiter les données EDA et/ou ECG.
4. Mise en place d’un système afin d’enregistrer et charger l’ensemble des données traitées.
5. Divisé les données entre entrainement et test.
6. Mise en place de notre algorithme.
7. Afficher des informations concernant notre modèle et ses performances (Gini importance, learning curve, la matrice de confusion, les performances générales).
8. Mise en place d’un système pour enregistrer et charger notre meilleur modèle.
9. Comparaison des améliorations avec le seed le plus « compliqué » (limité, pas très représentatif des réelle nouvelle données)
10. Comparaison des améliorations avec Kfold.
11. Amélioration des caractéristiques (features).
12. Recherche des meilleurs paramètres.
13. Comparaison des différents levels de segmentation
14. Fusion des résultats des modèles

En vue du nombre d’étape, uniquement les plus intéressantes seront présentées. Dans tous les cas, l’entièreté de code python est en annexe [[2]](#_Annexe).

# Système général

On peut voir ci-dessous, le système complet dans le squelette du code fourni :



Premièrement, on récupère et traite les données en entrée (EDA, réponse galvanique de la peauet ECG, Electrocardiogramme). Ensuite, une séparation simple de 80% - 20 % est effectuée. Finalement, un algorithme avec un comportement aléatoire est utilisé pour l’entrainement, le score et la prédication. Les résultats (0 = NST – 1 = ST) sont stockés dans un fichier excel (.csv).

Ce système de base complet a été très utile afin de commencer plus facilement ce projet. Par la suite, cette structure a été modifié étape par étape afin de développer un système plus complexe et fonctionnel. Ci-dessous, un schéma représentant le système final pour l’évaluation des performances:

Traitement des données + sauvegarde

Normalisation robuste

Chargement des données prétraitées

K-fold (5) Split trainset & testset

Train / Score / Predict

10x

Affichage performances moyennes

Recherche/sauvegarde des meilleurs paramètres

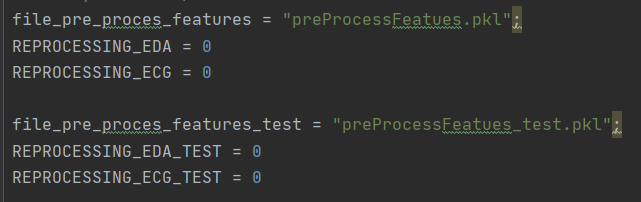
EDA

ECG

Les parties en rouge sont les parties ajoutées au système de base.

# Sauvegarde et chargement des données prétraitées

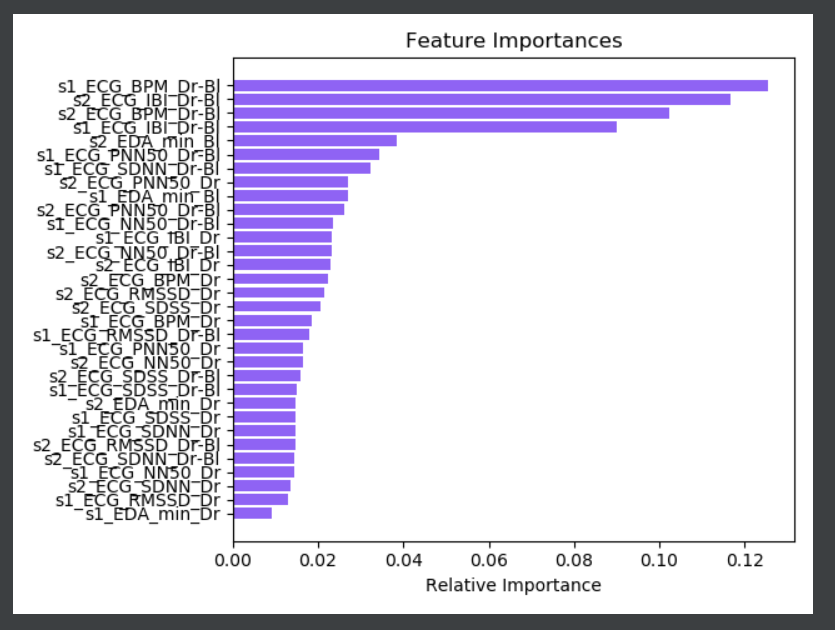
Au début du fichier python, il y a plusieurs paramètres permettant d’indiquer si on souhaite traiter les données d’entrée ou si on souhaite simplement les charger depuis un fichier. Le nom du fichier est aussi un paramètre. Ce fichier sera donc utilisé pour charger les données dans le cas ou on ne souhaite pas retraiter les données et sera utilisé dans tous les cas pour sauvegarder les données traitées.



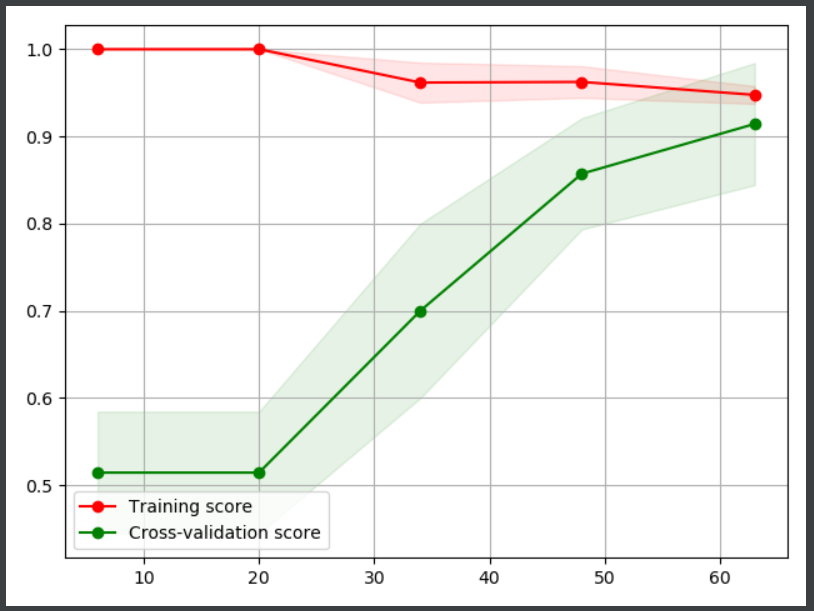
# Affichage des informations concernant notre modèle et ses performances

Assez rapidement les informations concernant notre modèle et ses performances a été mis en place afin d’avoir un meilleur aperçu de l’évolution de notre système.

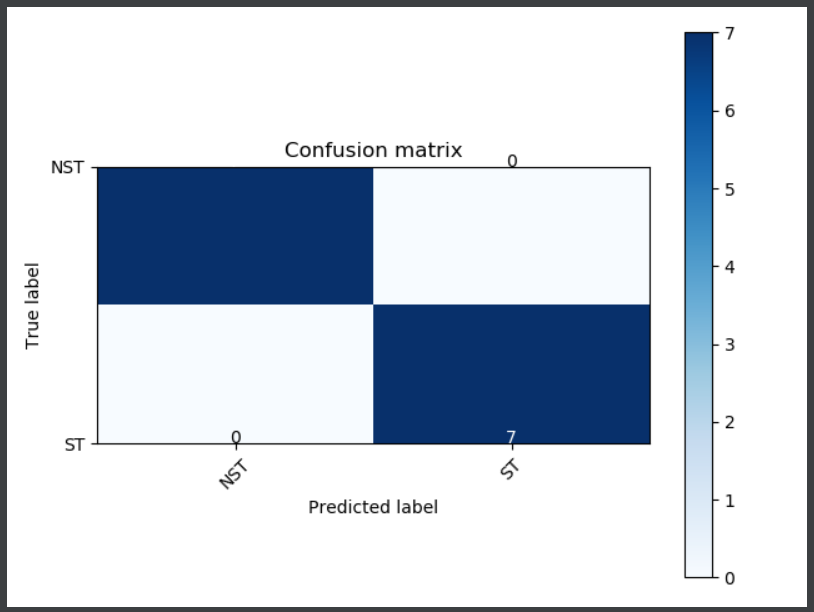
Pour commencer, l’affichage de l’importances des caractéristiques (Gini importance) a été affichées sous forme graphique :

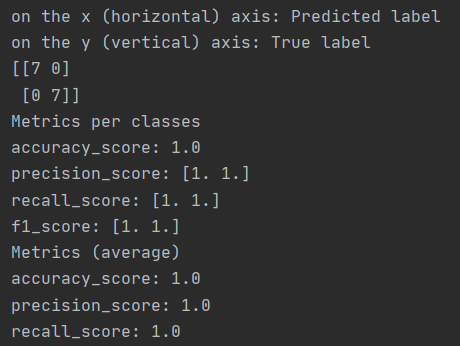


Ensuite, la courbe d’apprentissage a été affichées sous forme graphique :



Ainsi que la matrix de confusion et différents paramètres d’évaluations :

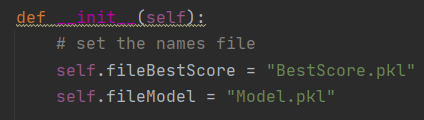




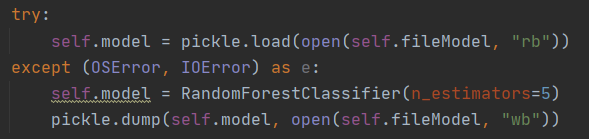
# Sauvegarde et chargement du meilleur modèle

Dans la fonction d’initialisation de la classe de notre algorithme, deux noms de fichiers sont initialisés afin d’indiquer les fichiers contenant :

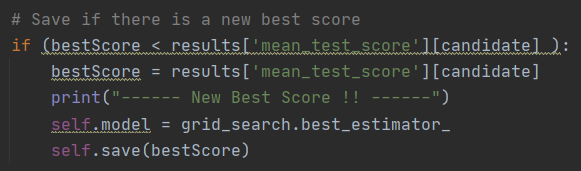
1. Le meilleur score du meilleur modèle en cours.
2. Le meilleur modèle en cours.



Dans cette même fonction d’initialisation, nous allons charger le meilleur modèle en cours en appelant la fonction « load\_model ». Cette fonction permet de charger le modèle sauvegardé dans le fichier indiqué précédemment. Dans le cas où ceci ne fonctionne pas, un algorithme de base est créé et sauvegardé dans ce fichier. Voici une partie du code de la fonction « load\_model » :



Plus tard, nous allons voir que la recherche des meilleurs paramètres est une étape qui est implémenté dans notre système. Il est important que lorsqu’un modèle avec des paramètres définies fourni un meilleur score que notre modèle en cours celui-ce soit sauvegardé dans le fichier présenté precedemment. Voici un bout de code dans la fonction de recherche des meilleurs paramètres qui permet cette sauvegarde :



# Comparaison des améliorations

Pour commencer, j’ai pensé à chercher et observer le pire cas de séparation des données afin d’améliorer mon système en me basant sur le pire cas. Pour ce faire, il a fallu tester plusieurs seed lors de la séparation des données afin d’utiliser un set de test compliqué. Voici la commande permettant de séparer les données en fonction du seed choisis :



Ensuite, on peut voir un tableau résumant différents seed testés et la valeur F1 moyenne observée :



Dans notre cas, on peut voir que le seed N°21 sépare les données de la manière le plus compliquée.

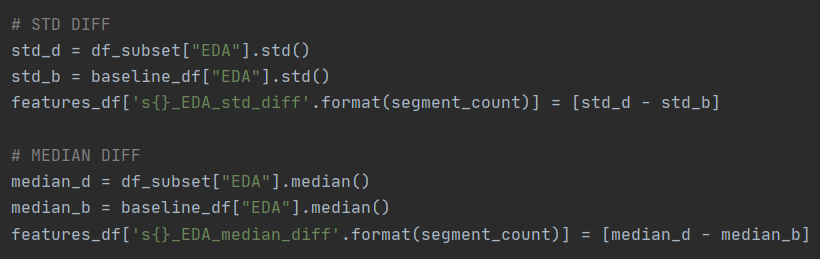
Après plusieurs comparaisons, cette méthode a été abonnée car elle est limitée et représente mal des nouvelles réelles données.

La nouvelle méthode permettant de comparer convenablement les performances du système utilise la validation croisée à k blocs (k-fold). De plus, étant donné que l’algorithme « Random Forest » est non déterministe, une boucle répétant plusieurs fois cette méthode a été mise en place afin d’avoir une vue sur la moyenne de la précision de notre modèle ainsi que le F1-score. Cette partie du code est assez conséquente, donc je vous laisse observer par vous-même dans le code en annexe (challenge\_skeleton.py) entre la ligne 240 et 275.

# Amélioration des caractéristiques

La caractéristique ECG a déjà bien été traité dans la fonction fournie « compute\_ECG\_features » afin d’en sortir des caractéristiques hauts niveaux. Pour ce qui est des données EDA, les minimums de la baseline et du mode de conduite ont été fournis.

Il est connu que des personnes transpires plus facilement et/ou fortement que d’autres. De ce fait, je pense qu’il est plus judicieux de prendre en compte des caractéristiques pas trop influencées par ce fait. J’ai donc décidé d’en sortir l’écart-type ainsi que le médiane des données EDA. Ensuite, j’ai ajouté la différence entre la mode conduite et la baseline qui est je pense très important pour ne pas être influencé par les personnes transpirants plus ou moins facilement que la moyenne. Voici le code permettant de récupérer ces caractéristiques :

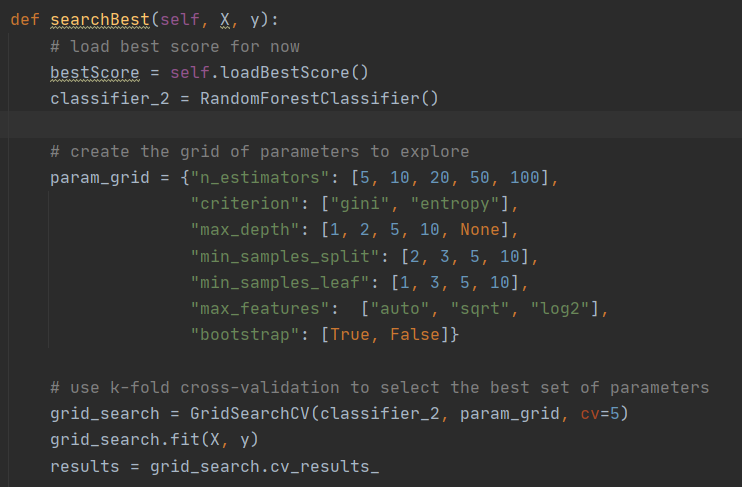


# Recherche des meilleurs paramètres

La fonction « searchBest » permettant de rechercher les meilleurs hyperparamètres a été ajoutée à la classe de notre algorithme. Celle-ci commence par récupérer le meilleur score observé jusqu’à maintenant qui est stocké dans le fichier indiqué dans la fonction d’initialisation.

Ensuite, une grille de tous les paramètres a testé est initialisée. On peut voir qu’un grande partie des paramètres et qu’une large échelle a été entrée. Cette grande grille fait que le temps d’exécution est long mais couvre un maximum de possibilités.

Le principe de cross-validation a aussi été ajouté dans la recherche des hyperparamètres.



Finalement, on récupère les résultats et test si un cas a dépassé notre meilleur score actuel. Si oui, on enregistre le nouveau meilleur modèle dans le fichier indiqué dans la fonction d’initialisation.

# Comparaison des différents levels de segmentation

Après avoir mis un système relativement complexe en place, différents niveaux de segmentation ont été testés afin d’observer l’importance de ce paramètre. En laissant mon système actuel mais en réactivant les traitements des données et en testant le niveau de segmentation à 1, 5 et 10. Voici, ci-dessous, les différents résultat obtenus :

On peut constater que plus le niveau de segmentation est élevé, meilleures sont les performances. Cet effet peut sûrement être expliqué du fait que la segmentation permet de différencier plus facilement les différents évènements survenus lors de l’expérience. Avec cette conclusion, le niveau de segmentation de 10 a été maintenu et une nouvelle recherche des hyperparamètres a été effectuée.

# Évolution des performances

Ci-dessous, l’évolution complète des performances par étape :

*Remarque : par défait, les premières étapes utilisent une segmentation de 2.*

Voici la liste des descriptions des étapes :

1. Nothing: Simple Random Forest sans modification
2. Features : Ajout des caractéristiques écart-type et médiane.
3. Scale : Ajout d’une normalisation robuste.
4. Best : Recherche des meilleures paramètres.
5. 1 : Niveau de segmentation à 1.
6. 5 : Niveau de segmentation à 5.
7. 10 Niveau de segmentation à 10.
8. 10 + best : Niveau de segmentation à 10 avec une recherche des meilleures paramètres.

On peut constater que l’ajout des caractérisés ne change pas grandement les performances. Cependant, on peut penser qu’en augmentant la segmentation, la différence aurait pu être remarquée. La normalisation robuste des datas ne modifie pas non plus beaucoup les performances. Comme nous l’avons vu, cela s’explique car l’algorithme RF ne nécessite pas de normalisation. En revanche, la recherche des meilleures paramètres augmente bien les performances. Comme constaté dans le chapitre précédent, l’augmentation du niveau de segmentation améliore les performances. Finalement, avec une segmentation de 10 et en recherchant les meilleurs paramètres, les performances les plus élevées ont pu être remarquées et appliquées.

Remarque : En plus d’observer la précision moyenne et le f1-score moyen, il aurait été intéressant d’analyser la variation des différents systèmes.

# Conclusion

## Difficultés rencontrées

* La quantité de données fournie était très grande et pas facile à saisir.
* L’algorithme RF étant non déterministe, l’évaluation des performances n’était pas simple.
* Malheureusement toute la partie collaboration/fusion n’a pas pu être pratiquée.

## Compétences acquises

* Absolument toutes les étapes nécessaires à la conception, la réalisation et l’évaluation d’un système de machine learning.

## Résultats obtenus

Finalement, un système complet a pu être mis en place avec des systèmes de sauvegarde et de chargement des données prétraitées et du modèle. De plus, des résultats non pas parfait mais relativement correctes ont pu être observés, analysés et améliorés. Dans l’ensemble, je suis plutôt satisfait de mon travail.

# Annexe

1. Enoncé du challenge
2. Codes Python

Date : 22.12.20

Nom de l’étudiant : Spinelli Isaia