Présentation de l'étude de prédiction de maladies cardio-vasculaires

Alexis PLESSIER
M1 Informatique

Le 5 décembre 2021

Problématique : Peut-on obtenir un taux de prédiction assez élevé pour l'appliquer en milieu médical ?

Présentation du DataSet

Le DataSet résulte d'une étude faite par plusieurs Hôpitaux autour du monde afin d'établir un lien entre certaines données et la présence de maladies cardio-vasculaires. Il contient 11 indicateurs et 918 observations ainsi que la présence ou non d'une maladie et est formé comme ci-dessous

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope	HeartDisease
0	40	M	ATA	140	289	0	Normal	172	N	0.0	Up	0
1	49	F	NAP	160	180	0	Normal	156	N	1.0	Flat	1
2	37	M	ATA	130	283	0	ST	98	N	0.0	Up	0
3	48	F	ASY	138	214	0	Normal	108	Y	1.5	Flat	1
4	54	M	NAP	150	195	0	Normal	122	N	0.0	Up	0

DataSet : https://www.kaggle.com/fedesoriano/heart-failure-prediction

Âge : Âge du patient
Sex : Sexe du patient
ChestPainType : Type de douleur thoracique
RestingBP : Battement du cœur au repos (/min)
Cholestérol : Taux de cholestérol (mm/dl)
FastingBS : Glycémie à jeun (mg/dl)

RestingECG : Électrocardiogramme au repos
MaxHR : Battement du cœur maximal (/min)
ExerciseAngina : Angine de poitrine suite à un exercice
Oldpeak : Valeur du pic le plus haut sur l'électrocardiogramme
ST_Slope : Type de pic sur l'électrocardiogramme

Pré-Processing (1/2)

Choix de la technologie : Pour mener à bien ce projet, j'ai décidé de coder en **Python**, car c'est un langage adapté au problème avec des bibliothèques complètes, mais également, car c'est un langage que je maitrise bien.
La première étape est d'effectuer un travail de Pré-Processing sur notre DataSet, le but ici est de nettoyer et arranger les valeurs présentes sans les modifier ou modifier leurs liens entre elles.

Gestion des valeurs manquantes

Une première étape est de gérer les valeurs manquantes. Nous avons alors plusieurs choix qui s'offre à nous : On peut supprimer chaque ligne présentant une valeur manquante ou bien lui attribuer une valeur parmi différents procédés. Par chance, notre DataSet est bien couplet.

```
Nombre de valeurs manquantes :
 Age               0
Sex                0
ChestPainType      0
RestingBP          0
Cholesterol        0
FastingBS          0
RestingECG         0
MaxHR              0
ExerciseAngina     0
Oldpeak            0
ST_Slope           0
HeartDisease       0
dtype: int64
```

Modification des valeurs qualitatives

La deuxième étape va être à présent de modifier les valeurs qualitatives (qui expriment des qualités ou des états uniques comme le sexe par exemple) en valeurs quantitatives. Ceci est nécessaire pour la compréhension de la machine qui apprend. Nous allons donc modifier les valeurs pour qu'elle soit représentées de manière numérique. D'après notre DataSet, nous allons faire ça avec 5 indicateurs

```python
#Remplace les valeurs de Sex en INT
df['Sex'].replace(['M','F'],[0,1], inplace = True)

#Remplace les valeurs de ChestPainType en INT
df['ChestPainType'].replace(['TA','ATA','NAP','ASY'],[0,1,2,3], inplace = True)

#Remplace les valeurs de RestingECG en INT
df['RestingECG'].replace(['Normal','ST','ST-T','LVH'],[0,1,2,3], inplace = True)

#Remplace les valeurs de ExerciseAngina en INT
df['ExerciseAngina'].replace(['N','Y'],[0,1], inplace = True)

#Remplace les valeurs de ST_Slope en INT
df['ST_Slope'].replace(['Up','Flat','Down'],[0,1,2], inplace = True)
```

Pré-Processing (2/2)

Préparation des ensembles d'entrainement et de test

Pour bien définir sur quoi nous allons entrainer notre modèle et sur quoi nous allons le tester, nous devons couper notre DataSet en deux parties :

- **y** qui sera la colonne "HeartDisease" et sera notre *target*.
- **X** qui sera tout notre DataSet sauf la colonne "HeartDisease" et qui seront nos *features*.

Le but ici est de séparer le résultat à ce qui nous amène au résultat.

	Age	Sex	ChestPainType	RestingBP	Cholesterol	FastingBS	RestingECG	MaxHR	ExerciseAngina	Oldpeak	ST_Slope
0	40	0	1	140	289	0	0	172	0	0.0	0
1	49	1	2	160	180	0	0	156	0	1.0	1
2	37	0	1	130	283	0	1	98	0	0.0	0
3	48	1	3	138	214	0	0	108	1	1.5	1
4	54	0	2	150	195	0	0	122	0	0.0	0

DataFrame **X**

0	0
1	1
2	0
3	1
4	0

DataFrame **y**

Normalisation des données

Il est souvent plus intéressant de mettre les données à une échelle connue que de les laisser brutes. Ces opérations de mise à l'échelle modifie les données, mais pas le lien et le rapport de la distance qu'elles ont entre elles, et ceci permet au modèle d'apprendre plus facilement et plus rapidement. J'ai choisi d'utiliser deux procédés qui sont le MinMax et le Standard. Après certains essaies le plus efficace s'est avéré être le **MinMaxScaler**. Son fonctionnement est simple, la valeur la plus petite à pour valeur 0 et la plus grande 1. Les valeurs entre ces deux là sont des rapports entre elles.

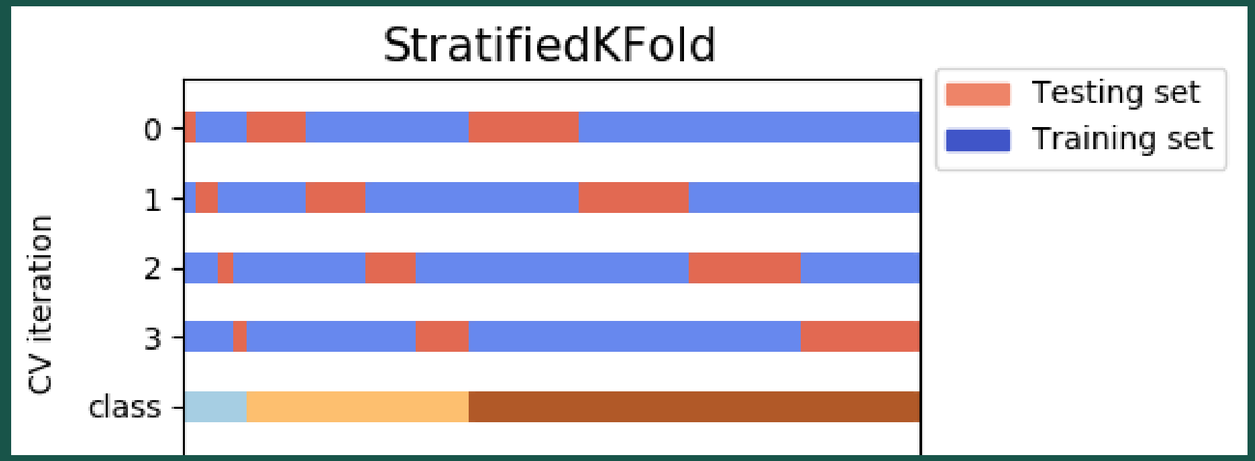
```
[[0.24489796 0.         0.33333333 ... 0.         0.29545455 0.        ]
 [0.42857143 1.         0.66666667 ... 0.         0.40909091 0.5       ]
 [0.18367347 0.         0.33333333 ... 0.         0.29545455 0.        ]
```

3 premières et 3 dernières valeurs du DataFrame **X** normalisé avec la méthode MinMax

Cross-Validation

Maintenant que nous avons nos ensembles sur lesquels entrainer nos modèles, nous devons définir des ensembles d'entrainement et des ensembles de tests. J'ai décidé de prendre 80% des données pour entrainement et 20% pour tester. Cependant, avec cette méthode, il peut arriver que nous passons à côte de quelque chose, car l'ensemble d'entrainement et de test sont toujours les mêmes, c'est pourquoi nous utilisons la **cross-validation**.

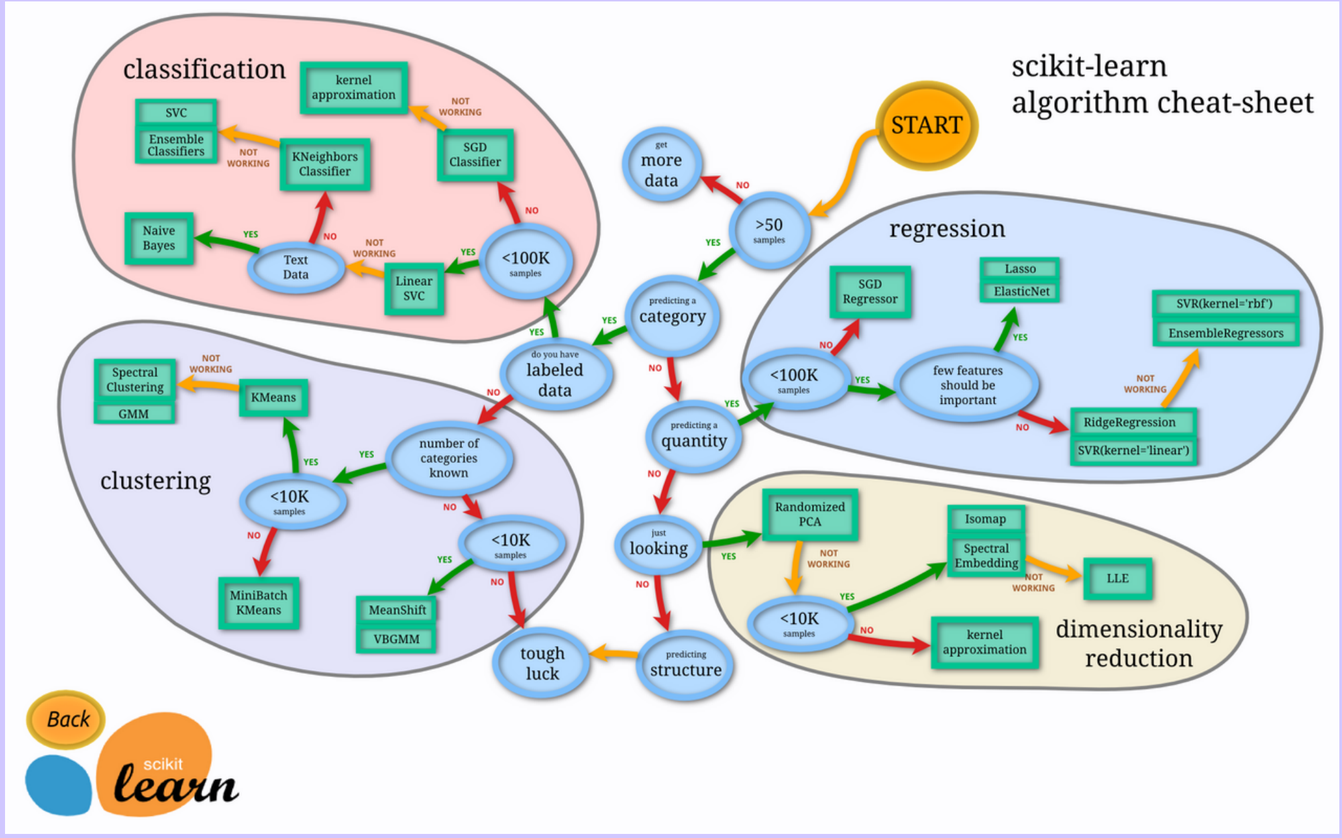
La cross-validation agit sur l'ensemble d'entrainement, il va découper ce dernier en autant de morceaux **N** que nous voulons et en définir un qui ne sera pas un ensemble pour entrainer, mais pour valider. Il va faire ça pour tous les morceaux ce qui nous permettra d'avoir **N** résultats de validations avec **N** ensemble d'entrainements différents pour enfin renvoyer la moyenne de bonne prédiction de chaque validation. Il existe plusieurs méthodes de découpages, cependant j'ai décidé de garder ce paramètre par défaut et d'utiliser la méthode **StratifiedKFold**.



Test des classifieurs

Selon le nombre de données que nous avons, le nombre de features et notre objectif, il existe de nombreux classifieurs plus ou moins spécialisé dans leur domaine. Je pourrais choisir d'utiliser un classifieur qui me parait le plus adapté, mais en réalité, ce que les professionnels font et bien … Ils en testent plusieurs ! Et c'est finalement le choix le plus logique, car on cherche à obtenir la précision la plus élevée peu importe que la méthode paraisse adaptée ou non. J'ai donc décidé de tester 6 classifieurs :

- **RandomForest**
- **AdaBoost**
- **Support Vector Machine**
- **K Near Neighbors**
- **Gradient Boosting**
- **Bagging**



Interprétation des résultats (1/2)

Matrice de confusion : L'exécution des apprentissages du classifieur (par la trainSet) sur le testSet va nous donner des résultats et notamment une matrice de confusion. Cette matrice de confusion de taille 2x2 va nous donner les effectifs prédis avec succès surses cases [0,0] et [1,1] et les echecs sur [0,1] et [1,0] :

Matrice de confusion type

n=165	Predicted: NO	Predicted: YES
Actual: NO	50	10
Actual: YES	5	100

```
RandomForest
[[ 66   7]
 [  9 102]]
```

```
AdaBoost
[[ 65   8]
 [ 11 100]]
```

```
SVM
[[ 66   7]
 [  9 102]]
```

```
KNN
[[ 67   6]
 [ 11 100]]
```

```
GB
[[64  9]
 [12 99]]
```

```
BC
[[66  7]
 [19 92]]
```

Rapport de classification : Les matrices de confusions sont utiles, mais on ne sait pas exactement quoi choisir au premier regard bien que cela nous donne une première idée. On va donc pouvoir générer un rapport de classification plus détaillé pour nous aider à décider quel classifieur choisir. Comme on cherche la précision la plus élevée, la variable **accuracy** (en F1) est ici la plus décisive.

	precision	recall	f1-score	support
0	0.88	0.90	0.89	73
1	0.94	0.92	0.93	111
accuracy			0.91	184
macro avg	0.91	0.91	0.91	184
weighted avg	0.91	0.91	0.91	184

Rapport RandomForest

	precision	recall	f1-score	support
0	0.86	0.89	0.87	73
1	0.93	0.90	0.91	111
accuracy			0.90	184
macro avg	0.89	0.90	0.89	184
weighted avg	0.90	0.90	0.90	184

Rapport AdaBoost

	precision	recall	f1-score	support
0	0.88	0.90	0.89	73
1	0.94	0.92	0.93	111
accuracy			0.91	184
macro avg	0.91	0.91	0.91	184
weighted avg	0.91	0.91	0.91	184

Rapport SVM

	precision	recall	f1-score	support
0	0.88	0.90	0.89	73
1	0.94	0.92	0.93	111
accuracy			0.91	184
macro avg	0.91	0.91	0.91	184
weighted avg	0.91	0.91	0.91	184

Rapport KNN

	precision	recall	f1-score	support
0	0.84	0.88	0.86	73
1	0.92	0.89	0.90	111
accuracy			0.89	184
macro avg	0.88	0.88	0.88	184
weighted avg	0.89	0.89	0.89	184

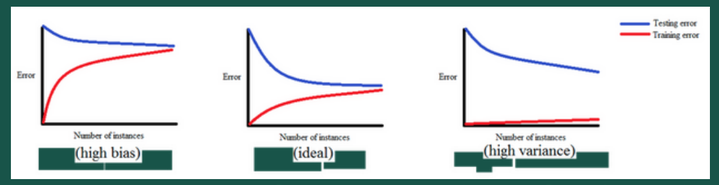
Rapport GB

	precision	recall	f1-score	support
0	0.78	0.90	0.84	73
1	0.93	0.83	0.88	111
accuracy			0.86	184
macro avg	0.85	0.87	0.86	184
weighted avg	0.87	0.86	0.86	184

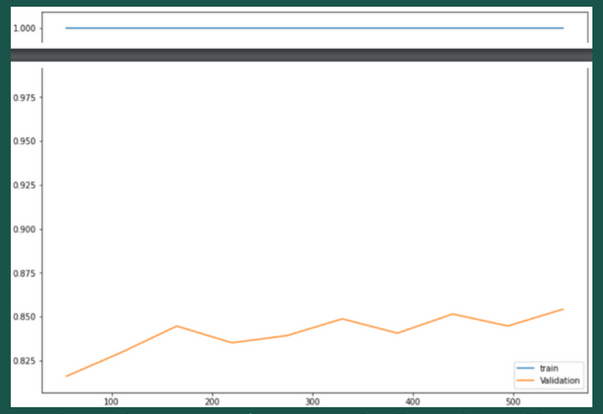
Rapport BC

Interprétation des résultats (2/2)

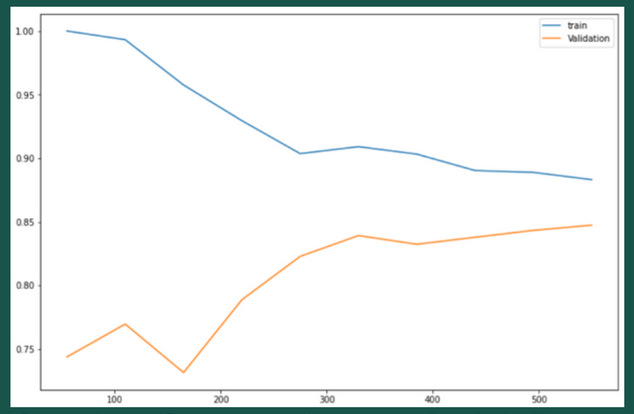
À la suite des rapports de classification, nous pouvons éliminer les deux derniers. Avant de désigner un modèle trop rapidement, il faut vérifier si notre modèle ne s'est pas **trop** entrainé. Car oui il est possible qu'il s'entraine trop et oublie toute notion de généralisation. Si notre modèle se spécifie trop, sa précision sera élevée sur le testSet mais lors de vrai cas, il sera inefficace, c'est ce que l'on appelle **l'overfitting**. En opposition à l'overfitting, si notre modèle n'arrive pas à apprendre des généralités, il est en cas d'**underfitting**. Pour éviter ces situations, il faut regarder la **courbe d'apprentissage** de chaque modèle



On vise le modèle central (modèle ideal)

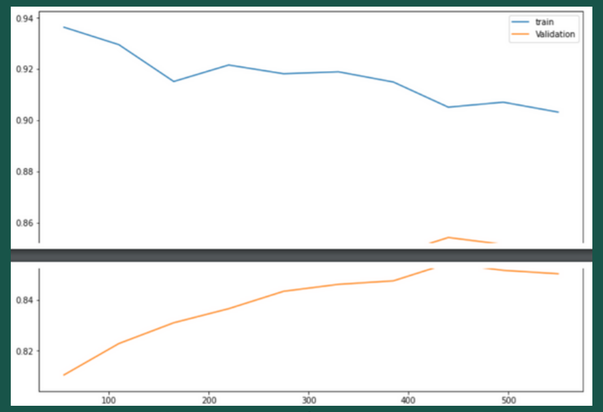


Courbe d'apprentissage de Random Forest

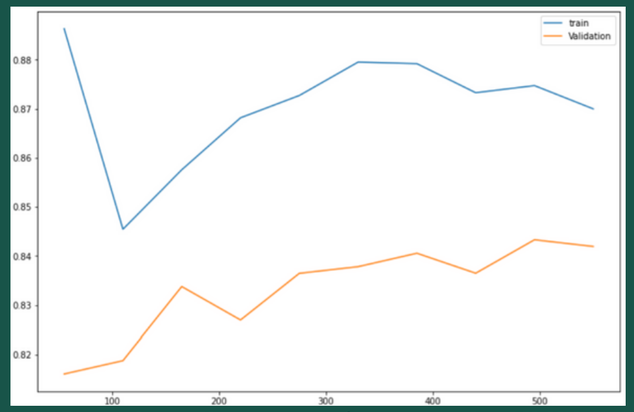


Courbe d'apprentissage de AdaBoost

Avec Les résultats ci-contre, nous constatons que le **Random Forest** et le **KNN** s'entrainent mal sur le testSet. Un graphique ressort tout particulièrement et c'est celui du modèle utilisant le classifieur **AdaBoost**. Nous allons donc utiliser ce dernier afin d'essayer de l'optimiser.



Courbe d'apprentissage de SVM



Courbe d'apprentissage de KNN

GridSearchCV

Maintenant que nous avons choisi le modèle final (**AdaBoost**), nous pouvons nous demander s'il est améliorable. Lors de l'exécution du modèle sur le testSet, la méthode possédait des *hyper-paramètres* par défaut. Est-ce qu'en modifiant leurs valeurs, nous pouvons obtenir un meilleur résultat ? Pour répondre à cette question, nous allons utiliser la méthode **GridSearchCV**, qui consiste à entrainer le modèle avec différentes valeurs pour différents hyper-paramètres de notre méthode afin de trouver la valeur de **accuracy** (valeur qui nous intéresse) la plus élevée comme ci-dessous :

```
#AdaBoost

hyper_params = {'n_estimators' : [1,10,50,100,500],
                'learning_rate' : [0.0001,0.001,0.01,0.1,1.0]}


grid = GridSearchCV(AdaBoost,hyper_params,scoring='accuracy',cv=4)
```

On décide ici de travailler sur les hyper-paramètres **n_estimators** et **learning_rate** de la méthode AdaBoost avec les valeurs rentrés pour optimiser la variable **accuracy** et une **cross-validation** en 4 parties.

Résultats :

```
[[ 67   6]
 [ 10 101]]
```

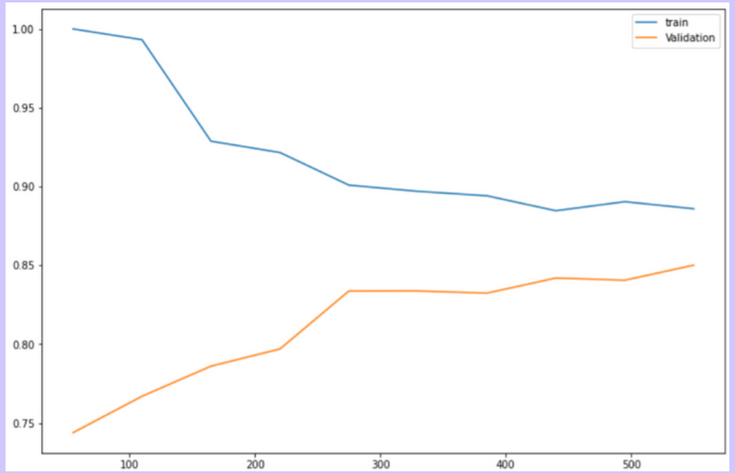
Matrice de confusion finale

```
{'learning_rate': 0.1, 'n_estimators': 500}
              precision    recall  f1-score   support

           0       0.87      0.92      0.89        73
           1       0.94      0.91      0.93       111

    accuracy                           0.91       184
   macro avg       0.91      0.91      0.91       184
weighted avg       0.91      0.91      0.91       184
```

Rapport de classification final



Courbe d'apprentissage finale

Nous avons une amélioration de la précision qui passe à **0.91** avec les hyper-paramètres : **{"learning_rate" : 0.1, "n_estimators" : 500**

Conclusion

Tout au long de ce projet, je me suis rendu compte que chaque étape était primordiale à la bonne compréhension du problème et à son analyse. Le pré-Processing est l'une des plus importantes et n'est pas à sous-estimer, pour l'exemple, j'ai développé le même modèle sans normalisation des données et la précision obtenue était de **74%**.

Au-delà de la préparation des données, il est important d'avoir une visualisation de l'apprentissage que l'on fait pour être sûr de la solidité de notre modèle.

Pour répondre à la question, notre modèle entrainé à une précision de **91,3%**. Ce résultat est très bon dans certains cas, mais il faut remettre du contexte à ce nombre. Le thème principal ici est **la santé d'un être vivant**, nous ne pouvons pas prendre le risque de détecter une maladie inexistante à 1 personne sur 10 comme nous ne pouvons pas passer à coter d'une maladie d'une personne sur 10. Même un taux de précision de *98%* serait trop faible, car le sujet étudié est trop important pour la prédiction seule actuellement. Cependant, il peut servir à indiquer ou épauler des professionnels de la santé, c'est une possibilité.