Cas pratique 1 – e2

Prédiction des crises cardiaques

Joachim ANDRE

Développeur en Intelligence Artificielle Ecole IA Microsoft By Simplon

Table des matières

[Introduction 3](#_Toc128128053)

[1. Interprétation des indicateurs de performance du modèle existant 3](#_Toc128128054)

[2. Définition des caractéristiques d’améliorations à apporter 6](#_Toc128128055)

[3. Intégration des améliorations 7](#_Toc128128056)

[4. Estimation de charge du besoin d’évolution de l’application 7](#_Toc128128057)

[5. Teste de la non-régression 7](#_Toc128128058)

# Introduction

Un alternant développeur IA, dans le cadre d’un service de cardiologie, a développé un algorithme de prédiction des risques d’attaque cardiaque. Un algorithme de Régression Logistique a été mis en production

Il nous est donc demandé d’améliorer les résultats du modèle existant afin d’avoir de meilleurs résultats, et une meilleure précision.

# Interprétation des indicateurs de performance du modèle existant

Notre dataset est composé de 303 lignes et de 14 colonnes. Voyons les colonnes plus en détails :

* Age : âge du patient
* Sex : sexe du patient
* exng : angine d’effort ( 1 = oui, 0 = non)
* caa : nombre de
* cp : Type de douleurs thoracique :
  + Valeur 1 : angine typique
  + Valeur 2 : angine atypique
  + Valeur 3 : douleur non angineuse
  + Valeur 4 : asymptomatique
* trtbps : tension artérielle au repos ( en mm/Hg)
* chol : cholestérol en mg/dl récupéré via capteur IMC
* fbs : glycémie à jeun > 120 mg/dl (1 = vrai, 0 = faux)
* rest\_ecg : électrocardiogramme au repos :
  + Valeur 0 : normal
  + Valeur 1 : présentant une anomalie de l’onde ST-T (inversion de l’onde T et/ou élévation ou dépression du segment ST > 0.05mV)
  + Valeur 2 : montrant une hypertrophie ventriculaire gauche probable ou certaine selon les critères d’Estes
* thalachh : fréquence cardiaque maximale atteinte
* output : diagnostic de maladie cardiaque (état de la maladie agiographique) :
  + 0 = <50% rétrécissement du diamètre, moins de risque de crise cardiaque
  + 1 = >50% rétrécissement du diamètre, plus de risque de crise cardiaque
* old peak : Dépression ST induite par l’exercice par rapport au repos.
* Slp : Pente du segment ST d’effort maximal :
  + 0 = Sans pente
  + 1 = Plate
  + 2 = descendant
* thall : thalassemia :
  + 0 = null
  + 1 = défaut corrigé
  + 2 = normal
  + 3 = défaut réversible

Le modèle utilisé est une Régression Logistique. C’est un algorithme d’apprentissage supervisé utilisé pour la classification binaire. Il est basé sur une fonction sigmoïde qui transforme la sortie linéaire en une probabilité. C’est un algorithme simple est rapide qui peut être utilisé pour des problèmes de classification binaire avec des ensembles de données petite à moyenne taille.

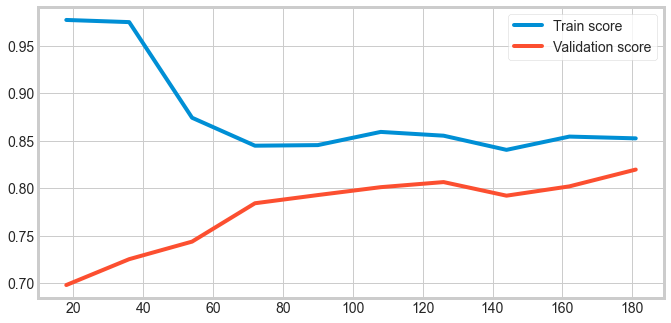
Nous allons analyser les métriques d’origines de notre modèle de Régression Logistique.

Nous pouvons visualiser la conception de notre modèle. Nous pouvons remarquer que le dataset n’est pas standardisé. La standardisation va permettre de mettre à l’échelle les données de manières robustes par rapport aux valeurs aberrantes (outliers). Il y’a deux méthodes : la méthode RobustScaler , qui contrairement à la méthode StandartScaler utilise la moyenne et l’écart-type, RobustScaler utilise les quartiles (Q1 et Q3) pour centrer et réduire les données. En effet la moyenne et l’écart-type sont sensibles aux valeurs aberrantes, alors que RobustScaler est plus robuste. Il est donc plus adapté aux jeux de données contenant des valeurs aberrantes et/ou des écarts importants entre les valeurs.

Une fois le modèle entrainé avec les hypers paramètres suivants :

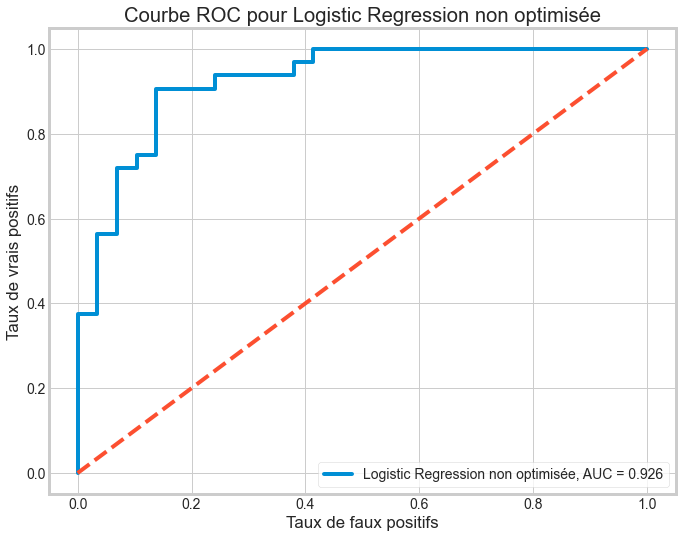
* Lbfgs (Limited-memory Broyden-Fletcher-Goldfarb-Shanno) pour le solver, est un algorithme d’optimisation numérique qui est souvent utilisé pour résoudre des problèmes d’optimisation non linéaire.
* Max\_iter correspond au nombre maximal d’itérations que l’algorithme de descente de gradient effectue pour trouver le minimum de la fonction coût. Si l’algorithme ne converge pas avant d’avoir atteint ce nombre maximal, il s’arrête et renvoie un message d’erreur.

Afin de pouvoir évaluer la performance de ce modèle, j’ai tracé les courbes d’apprentissage (learning curve). La courbe d’apprentissage est construite en traçant l’erreur d’entrainement et de validation/précision en fonction du nombre d’exemples d’entrainement utilisés. Elle permet de détecter si le modèle souffre d’un surapprentissage ou d’un sous apprentissage.



Ici nous pouvons voir que la courbe pour le train score diminue rapidement pour osciller autour des 0.80 et 0.85, alors que la courbe de validation monte rapidement pour rester dans les mêmes niveaux, soit aux alentours de 0.80. Le modèle n’est pas en surapprentissage, cependant on peut se demander si on dataset plus grand ne permettrait pas d’améliorer ce score, ainsi qu’une optimisation des hypers paramètres.

J’ai ensuite tracé une courbe ROC (Receiver Operating Characteristic), qui trace le taux de vrais positifs en fonction du taux de faux positifs. Le score AUC (Area Under the Curve) mesure la probabilité que le modèle classe un exemple positif de manière plus haute qu’un exemple négatif.



On voit ici que le modèle de base à un score AUC de 0.926.

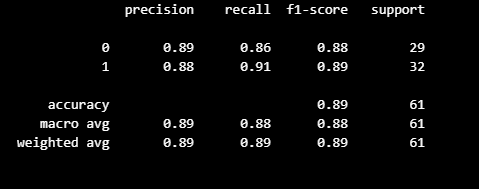
J’ai ensuite fait une matrice de confusion qui permet de comparer les prédictions du modèle avec les véritables étiquettes de la target (colonne « output »). Elle est présentée sous la forme d’une grille 2\*2 avec quatre cases :

* Vrai positif : prédiction correcte de la classe positive
* Faux positif : prédiction d’une classe positive, alors que vraie classe négative
* Vrai négatif : prédiction correcte de le classe négative
* Faux négatif : prédiction d’une classe négative, alors que vraie classe positive



Ici nous pouvons voir que le modèle a prédit 25 vrai positif et 29 vrai négatif.

Pour finir, j’ai fait un rapport de classification avec différents indicateurs de performance.



* Precision : proportion de prédiction positives correctes parmi toutes les prédictions positives.
* Recall : proportion de prédiction positives correctes parmi toutes les observations réellement positives
* F1-score : moyenne pondérée de la Precision et du Recall

# Définition des caractéristiques d’améliorations à apporter

Afin d’améliorer le modèle initial, je vais déjà regarder sa pertinence en le comparant à d’autres modèles de machine Learning de classification, les arbres de décision, un SVC et un Gradient Boosting Classifier.

L’arbre de décision est un algorithme d’apprentissage supervisé utilisé pour résoudre des problèmes de classifications ou de régression. Il utilise une structure arborescente pour représenter des décisions et leurs conséquences.

Le SVC est un algorithme d’apprentissage supervisé, son objectif est de trouver la meilleure frontière de décision (ou hyperplan) qui sépare les différentes classes d’objets dans un espace de grande dimension.

Le Gradient Boosting Classifier est un algorithme d’apprentissage supervisé qui combine plusieurs modèles d’arbres de décision pour former un modèle plus fort. Le modèle entraine successivement des arbres de décision en se concentrant sur les erreurs des prédécesseurs.

Je vais aussi créer une API pour permettre l’utilisation du modèle de prédiction, et sa potentielle intégration dans une application web.

# Intégration des améliorations

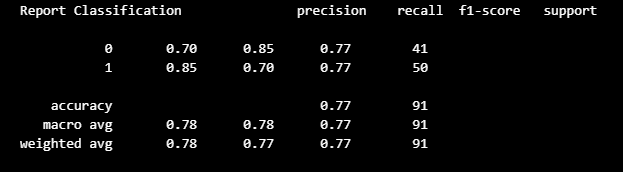
Afin d’intégrer les différents modèles cités si dessus, j’ai donc créé un pipeline pour les 4 modèles.

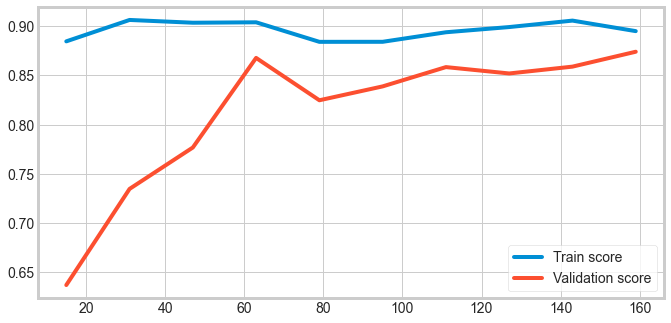
J’ai ensuite utilisé la méthode de GridSearchCV afin d’optimiser les hypers paramètres des modèles.

Afin d’évaluer chacun des modèles j’ai mis en place une matrice de confusion, un rapport de classification et une courbe d’entrainement.

Régression logistique

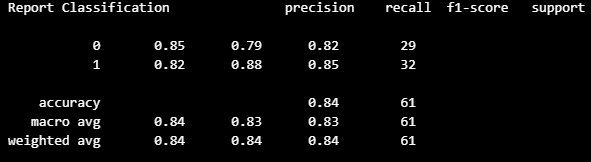


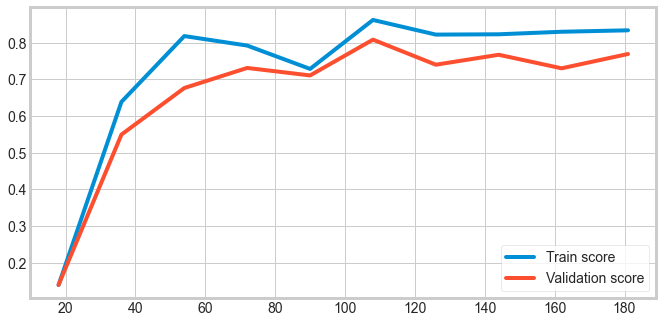




Arbre de décision

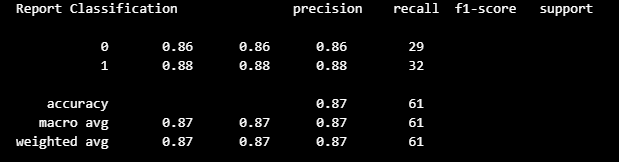


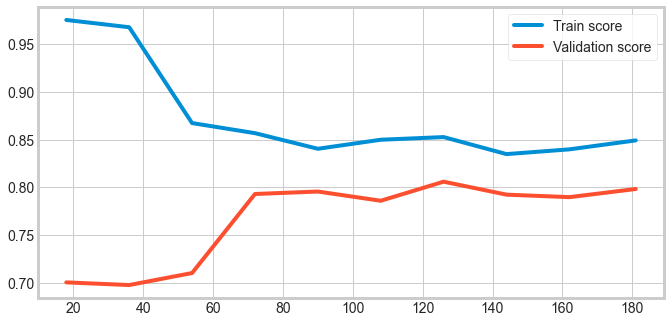




SVC

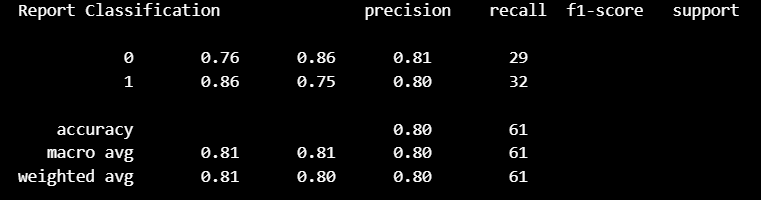


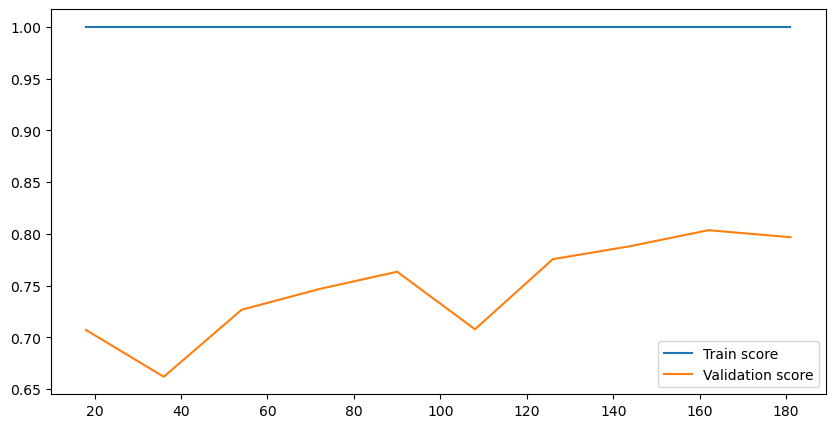




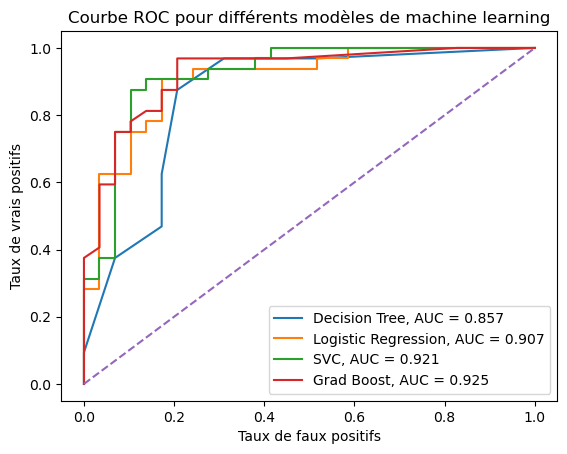
Gradient Boosting Classifier







Ensuite, pour visualiser la robustesse de nos modèles j’ai fait une courbe ROC. La courbe ROC représente le taux de vrais positifs en fonction du taux de faux positifs, pour différents seuils de classification. Ici le modèle de Gradient Boost est celui avec le meilleur score AUC, soit 0.925.



# Estimation de charge du besoin d’évolution de l’application

* Analyse du besoin : Après avoir discuté avec le client et compris ses besoins, vous comprenez que la nouvelle fonctionnalité nécessite des modifications en créant une API, la transformation des données, et la sélection d’un modèle d’algorithme ainsi que son optimisation.
* Identification des composants impactés : Modèle d’algorithme, API.
* Estimation de la complexité de l'intégration : Vous estimez que l'intégration de la nouvelle fonctionnalité est assez complexe car elle nécessite des modifications dans plusieurs composants différents, et implique une analyse minutieuse de l'impact potentiel sur l'existant. Vous attribuez une note de 7 sur 10 pour la complexité.
* Estimation du temps nécessaire : En utilisant les informations obtenues jusqu'à présent, vous estimez que l'intégration de la nouvelle fonctionnalité prendra environ 1 semaine de travail pour une équipe de deux développeurs.
* Prévoir des marges de sécurité : Vous prévoyez une marge de sécurité de 20 % pour tenir compte des imprévus tels que les retards de livraison de composants tiers ou les difficultés imprévues de la réalisation de l'intégration.

En fin de compte, votre estimation de charge pour cette évolution serait d'environ 1 semaines 1/2 de travail pour une équipe de deux développeurs, soit 41 heures de travail en tout. Bien entendu, cette estimation est approximative et peut varier en fonction de nombreux facteurs, mais elle fournit une base solide pour la planification et la gestion des projets.

# Teste de la non-régression