**Année 2020-2021**

**Rapport de projet E2 – Développeur en Intelligence Artificielle**

Optimisation d’un programme d’I.A existant

« Analyse des données de marketing en banque »

*Etude réalisé par*

Audrey FOUGERE



[1. Introduction 3](#_Toc69329249)

[2. Le modèle 3](#_Toc69329250)

[2.1. Principe 3](#_Toc69329251)

[2.2. Notations 3](#_Toc69329252)

[2.3. Le modèle LOGIT 3](#_Toc69329253)

[3. Analyse du dataset « bank\_full.csv 3](#_Toc69329254)

[3.1. Analyse exploratoire des données 3](#_Toc69329255)

[3.2. Preprocessing de nos données 4](#_Toc69329256)

[4. Construction d’un modèle de classification 4](#_Toc69329257)

[4.1. Partitionnement des données d’entraînement 4](#_Toc69329258)

[4.2. Prédictions sur les échantillons de test 5](#_Toc69329259)

[4.3. Analyse des différentes métriques 5](#_Toc69329260)

[5. Eviter le Surapprentissage/ Overfitting 5](#_Toc69329261)

[5.1. Définition 5](#_Toc69329262)

[5.2. Séparation des observations à la main 5](#_Toc69329263)

[6. Réglages du modèle/ Optimisation des hyperparamètres 6](#_Toc69329264)

[6.1. Introduction 6](#_Toc69329265)

[6.2. Evaluer la robustesse de notre modèle par la validation croisée 6](#_Toc69329266)

[6.3. Mise en place d’une meilleure performance de mon modèle 6](#_Toc69329267)

[6.4. Analyse des métriques après Optimisation des Hyperparamètres 7](#_Toc69329268)

[6.5. Sauvegarde de mon modèle de classification 7](#_Toc69329269)

[7. Amélioration de l’application d’intelligence artificielle grâce à l’ajout d’une évolution fonctionnelle 7](#_Toc69329270)

[7.1. Intégration de mon modèle d’IA dans une application existante 7](#_Toc69329271)

[7.2. Estimation de charge au regard du besoin fonctionnelle 7](#_Toc69329272)

[7.3. Intégration d’une évolution fonctionnelle dans mon modèle 7](#_Toc69329273)

[7.4. Test de la non-régression de l’application suite à l’intégration de cette évolution 7](#_Toc69329274)

# Introduction

Mon projet E2 vise à analyser des données de marketing en banque autour de la souscription d'un dépôt à long terme. Il a été créé dans le cadre du programme Data IA de l'école Simplon Microsoft à Nantes. Le jeu de données utilisé est issu du site UCI : <https://archive.ics.uci.edu/ml/datasets/Bank+Marketing> Le fichier csv utilisé pour faire ces analyses est le suivant : bank-full.csv. Cet ensemble de données va être utilisé pour prédire si un nouveau client est susceptible de souscrire à un dépôt à long terme. La *classification* est un exemple de technique d'apprentissage automatique supervisé, ce qui signifie qu'elle repose sur des données qui incluent des valeurs de caractéristiques connues ainsi que des valeurs d’étiquettes connues. Un algorithme de classification est utilisé pour adapter un sous-ensemble de données à une fonction qui peut calculer la probabilité pour chaque étiquette de classe à partir des valeurs d'entité. Les données restantes sont utilisées pour évaluer le modèle en comparant les prédictions qu'il génère à partir des entités aux étiquettes de classe connues.

# Le modèle

## Principe

La régression logistique est une méthode d'apprentissage supervisée très utile pour expliquer / prédire une variable discrète (à l'opposé de continu) lorsque le jeu de données comporte de nombreuses variables catégorielles.

## Notations

Soit Y la variable à prédire / expliquée (dans notre cas, il s'agit de la souscription ou non à l'option de dépôt à long terme) et X = (X1, X2, ..., Xn) les variables explicatives

Dans le cadre de la régression logistique binaire, la variable Y prend deux modalités {0,1}. Les variables Xi sont exclusivement binaires ou continues.

- Soit Ω un ensemble de n échantillons (ici n = 45211) comportant n1 (respectivement n0)) observations correspondant à la modalité 1 (respectivement 0) de Y (dans notre cas, n1 = 5289 et n0 = 39922).

- P(Y=1) est la probabilité a priori pour que Y = 1. Pour simplifier on la notera p(1)

- P(X|1) est la distribution conditionnelle des X sachant la valeur prise par Y.

- La probabilité a posteriori d'obtenir la modalité 1 de Y (resp. 0) sachant la valeur prise par X est p(1|X) (resp. p(0|X))

## Le modèle LOGIT

On désigne par le terme LOGIT de p(1|X) l'expression suivante :

- il s'agit bien d'une régression car on cherche à montrer une relation de dépendance entre une variable à expliquer tet une série de variables explicatives

- il s'agit d'une régression logistique car la loi de probabilité est modélisée par une loi logistique (<https://fr.wikipedia.org/wiki/Fonction_logistique_(Verhulst)>)

La fonction logistique est intéressante car elle est beaucoup moins sensible aux outliers dans le cas d’une classification binaire

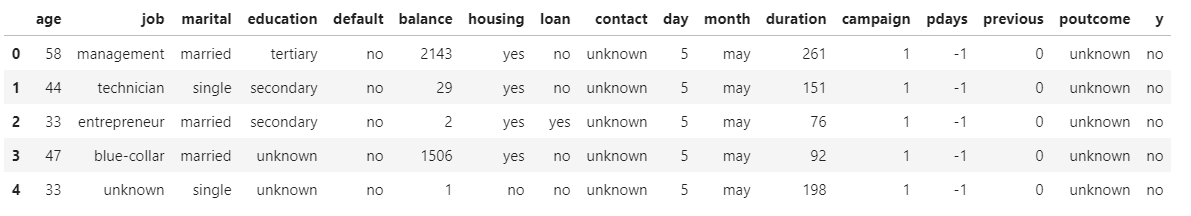
Expliquer / prédire une variable binaire consiste alors à estimer au mieux les coefficients *bi* ci-dessus. Ensuite on peut retrouver la probabilité conditionnelle par la formule suivante :

Il existe plusieurs méthodes de résolutions / d'estimations de ces paramètres (voir principe du maximum de vraisemblance et algo méthode de résolution numérique Newton-Raphson). Nous allons nous concentrer plus sur l'interprétation des résultats dans le cas d'une classification plutôt que sur les méthodes de résolution.

# Analyse du dataset « bank\_full.csv

## Analyse exploratoire des données

Après import des librairies : numpy, pandas, Preprocessing, statmodels et sklearn et du jeu de données « bank\_full.csv ». On utilise la méthode *read\_csv* pour convertir le csv en dataframe pandas.



On a 45211 observations pour 17 variables

- La variable d'intérêt (cible) est "y", représentant le taux de souscription à un dépôt de long terme

- On observe 7 variables numériques : "age", "balance", "day", "duration", "campaign", "pdays" et "previous" (même si "day" pourrait être considéré comme une variable catégorielle)

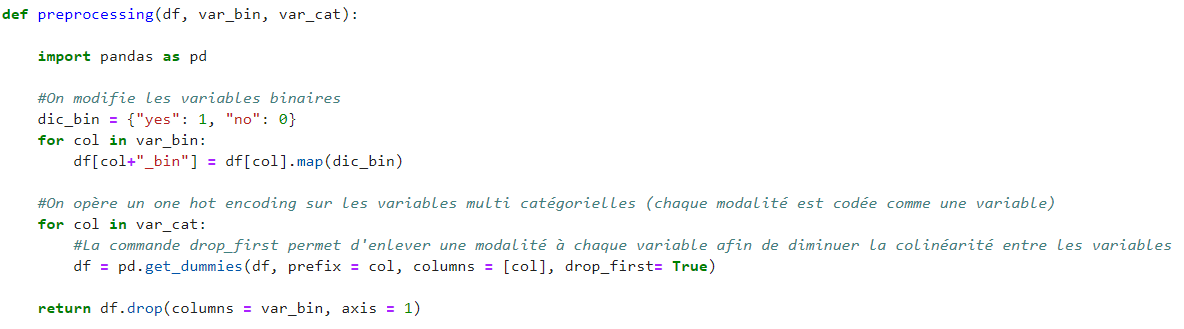
- On observe 10 variables catégorielles :

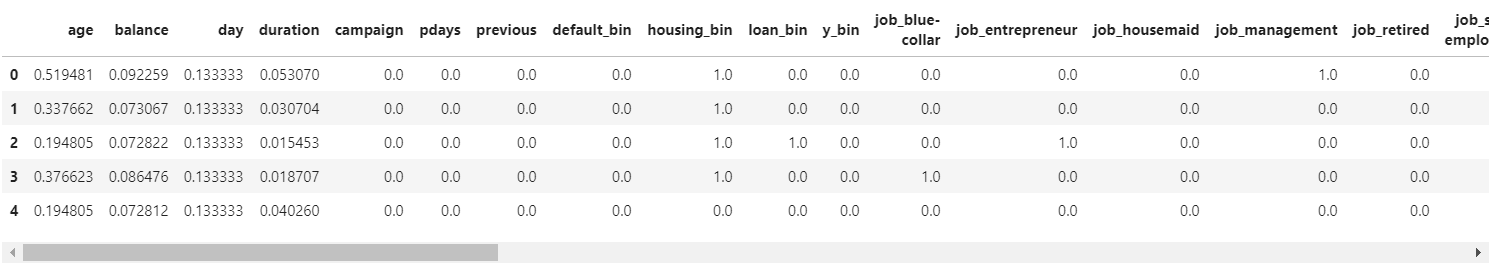
\* 4 variables binaires : "default", "housing", "loan" et "y"

\* 6 variables avec strictement plus de 2 modalités : "job", "marital", "education", contact", "month" et "poutcome"

## Preprocessing de nos données

Ensuite, on passe par une étape de prétraitement des données ou preprocessing des données. On a besoin de variables exclusivement binaires ou continues. On met en place un One-hot encoding, chaque modalité de variables multi catégorielles est transformée en variable binaire.

On met en place cet encoding avec le script python preprocessing ci-dessous :

On introduit une étape de normalisation de type min max afin d'avoir toutes les variables entre 0 et 1. En l'occurrence, cela n'apportera pas énormément dans le sens où l'on veut appliquer une régression logistique (moins sensible aux échelles) mais il serait primordial d'en mettre une en place si on appliquait un modèle linéaire.

# Construction d’un modèle de classification

## Partitionnement des données d’entraînement

Lors de l’ajustement des modèles, je souhaite m’assurer de deux choses :

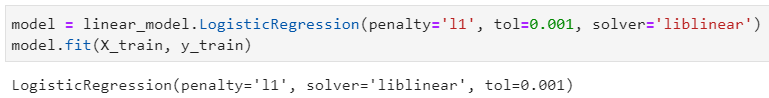
* Trouver le meilleur modèle (en termes de paramètres du modèle)
* Généraliser le modèle c’est-à-dire le faire fonctionner correctement sur des données invisibles (sur lequel il ne s’est pas entraîné)

Une image contenant table

Description générée automatiquementPendant cette étape, nous séparons nos données en deux sous-ensembles : les données d’entrainement et les données de test. Notre modèle va s’entrainer sur les données d’entrainement afin de faire des prédictions sur les données de test (données sur lesquelles mon modèle ne s’est pas entraîné

Tout d’abord, je mets en place une régression logistique de base : avec une division des données en 70%/30% et un entrainement sur l’ensemble des données train et testez la précision sur l’ensemble des données test.

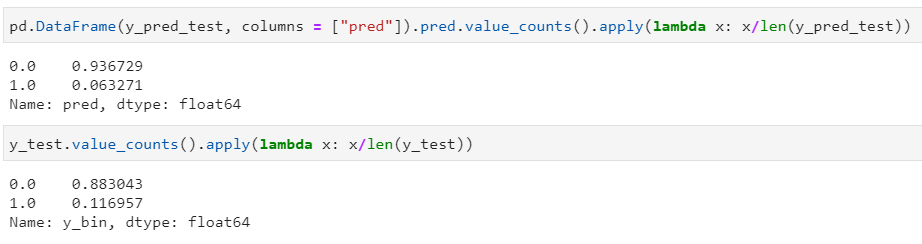
Notre dataset ne contient pas beaucoup d'observations (seulement quelques dizaines de milliers), on peut donc se dire qu'un solver « liblinear » serait approprié pour résoudre un tel problème.



## Prédictions sur les échantillons de test

En faisant des prédictions sur l'échantillon de test, on se rend compte que le modèle prédit deux fois moins de personnes susceptibles de prédire qu'il y en a dans l'échantillon. Le modèle a tendance a être "frileux" (prédire moins de 1 que de 0) pour avoir une meilleur accuracy. Il est possible de changer ça en jouant sur l'hyperparamètre class\_weight



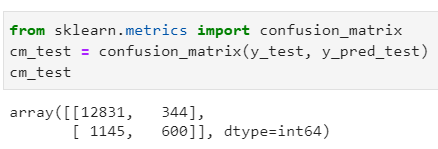


## Analyse des différentes métriques

Plusieurs paramètres d’un algorithme de classification nous permettent d’évaluer si un modèle est optimale :

* L’accuracy
* La precision (également appelée valeur predictive positive)
* Le recall (également appelé sensibilité)
* Le F1-score, calculé à partir de la precision et du rappel du test.
* La courbe ROC

En apprentissage automatique supervisé, la matrice de confusion est une matrice qui mesure la qualité d’un système de classification. Chaque ligne correspond à une classe réelle, chaque colonne correspond à une classe estimée. Un des intérêts de la matrice de confusion est qu’elle montre rapidement si un système de classification parvient à classifier correctement

Une image contenant table

Description générée automatiquementRegardons de plus près cette matrice :

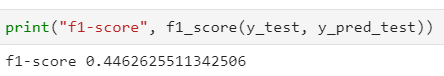
True Negative 12831

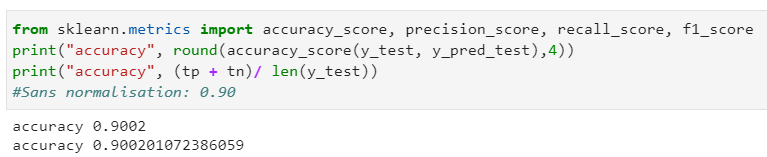
True Positive 600

False Positive 344

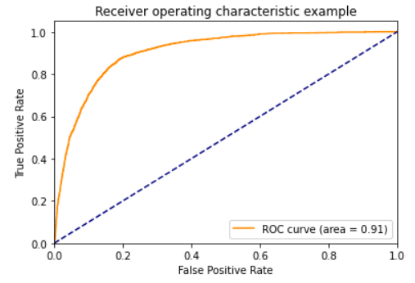
False Negative 1145

La precision = TP / TP + FP = 600 / (600 + 344) = 0.6356

Le recall = TP / TP + FN = 600 / (600 + 1145) = 0.03438



Un autre indicateur qui peut être utilisé pour évaluer notre modèle est la courbe ROC, également appel courbe de sensibilité/spécificité. C’est une mesure de la performance d’un classificateur binaire. Graphiquement, on représente la courbe ROC sous la forme d’une courbe qui donne le taux de vrais positifs en fonction du taux de faux positifs.



**AUC** signifie "aire sous la courbe ROC". Cette valeur mesure l'intégralité de l'aire à deux dimensions situées sous l'ensemble de la courbe ROC (par calculs d'intégrales) de (0) à (1).

Les valeurs d'AUC sont comprises dans une plage de 0 à 1. Un modèle dont 100 % des prédictions sont erronées a un AUC de 0. Si toutes ses prédictions sont correctes, son AUC est de 1.

Dans notre cas précis, notre AUC = 0.91

# Eviter le Surapprentissage/ Overfitting

## Définition

Lorsque le dataset est fractionné, il peut y avoir un surapprentissage ou alors un sous-ajustement de notre modèle. L’objectif est bien évidement d’éviter ces deux problèmes d’ajustement. Le surajustement signifie que notre modèle que nous avons formé s’est en quelque sorte « trop bien » entraîné et est maintenant trop ajusté à l’ensemble des données. Il a appris par cœur les données sans vraiment comprendre.

Cela peut se produire quand le modèle est trop complexe (c’est à dire trop de caractéristiques, variables par rapport au nombre d’observations). Ce modèle aura une grande précision pour les données de formation, mais probablement pas assez précis pour de nouvelles données/observations. Quant au sous-ajustement, cela signifie que le modèle ne s’ajuste pas aux données d’apprentissage et manque donc les tendances que peuvent traduire les données. Le modèle ne peut pas non plus être généralisé à de nouvelles données. C’est le résultat d’un modèle trop simple (c’est-à-dire pas assez de variables indépendantes).

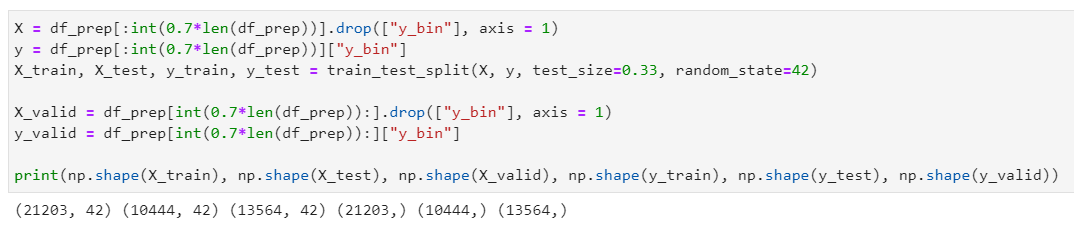
Une image contenant texte, horloge

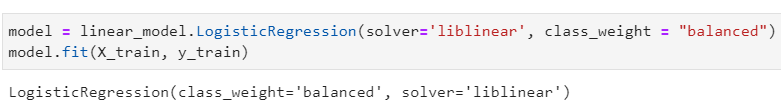
Description générée automatiquementVoici un schéma représentant un surajustement, un sous-ajustement et un modèle « parfait »

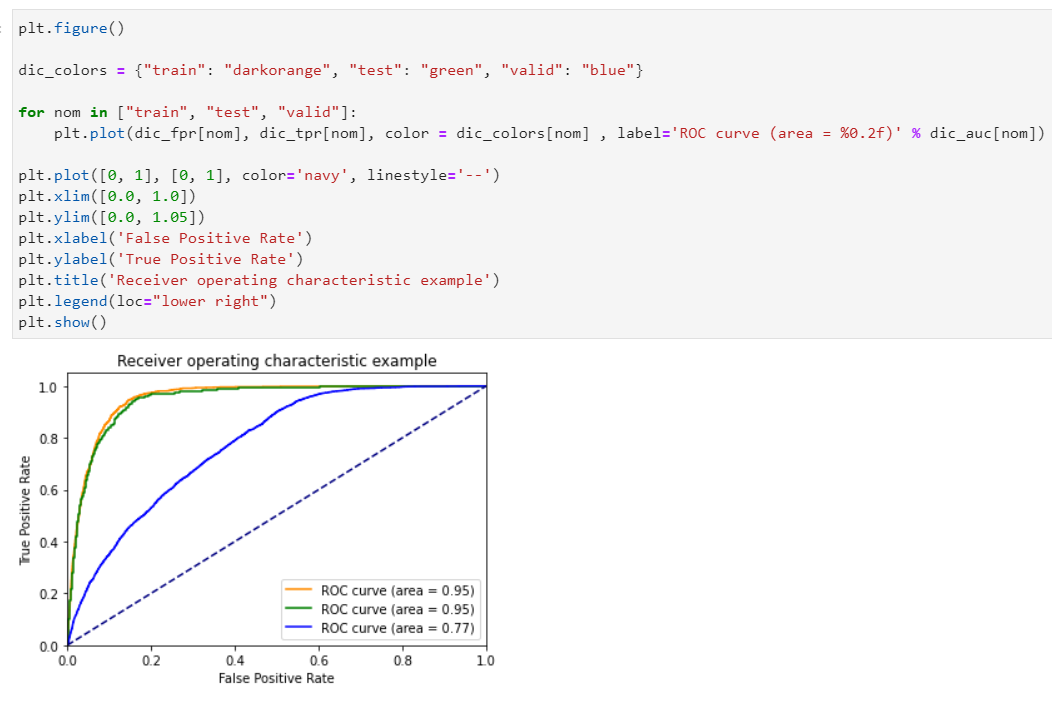
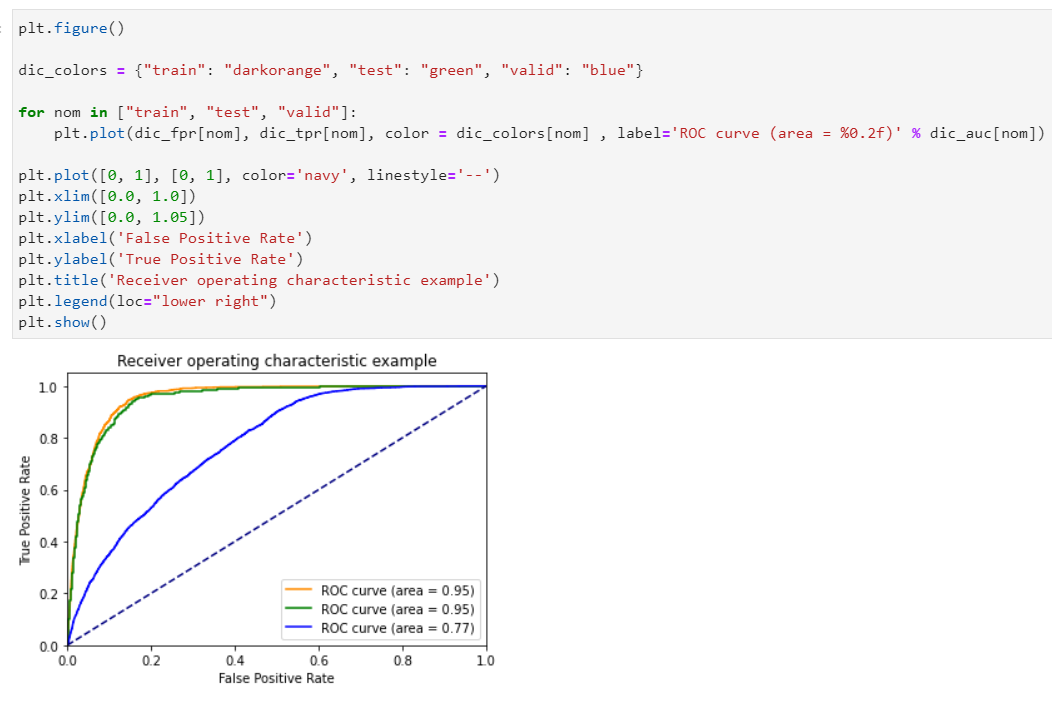
Dorénavant, pour pallier ce souci de surapprentissage, on va créer trois bases de données : une base d'apprentissage, une de test et une dernière de validation pour laquelle on est sûr que le modèle n'a jamais vu les données.

# Séparation des observations à la main

Le découpage suivant donne une répartition inégale des 0 et 1, il permet de voir comment le modèle s'adapte à des taux de souscription variable en fonction des échantillons.







# Réglages du modèle/ Optimisation des hyperparamètres

## Introduction

Il s'agit de rechercher le "meilleur" set d'hyperparamètres pour notre modèle. On essaie donc de chercher le meilleur arbitrage entre performance et robustesse du modèle. La performance consiste à avoir des scores élevés pour les différents métriques testées et la robustesse signifie avoir des métriques similaires au cours des différents apprentissages de la cross validation. Lors de l'optimisation des hyperparamètres, il est possible de spécifier le score que l'on cherche à maximiser. Il semblerait qu'il y ait deux stratégies possibles pour la banque :

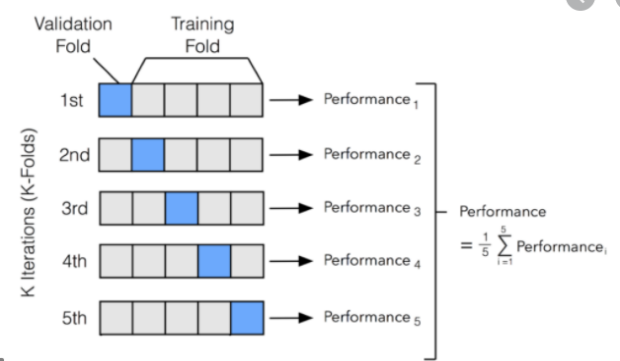
- contacter un maximum de personnes pour avoir le plus de souscriptions possibles (avoir le recall le plus grand possible)

- appeler un nombre limité de clients mais particulièrement bien ciblés (avoir la meilleure précision possible).

N'ayant pas d'indications concernant la stratégie que la banque cherche à adopter, on a choisi de maximiser le **f1-score**.

## Evaluer la robustesse de notre modèle par la validation croisée

Il est acquis qu’un modèle doit être évalué sur une base de test différente de celle utilisée pour l’apprentissage. Mais la performance est peut-être juste l’effet d’une aubaine et d’un découpage particulièrement avantageux. Pour être sûr que le modèle est robuste, on recommence plusieurs fois. On appelle cela la validation croisée ou cross validation. Concernant notre modèle, nous découpons la base de données en cinq segments de façon aléatoire. On en utilise quatre pour l’apprentissage et un pour tester. On recommence cinq fois. Si le modèle est robuste, les cinq scores de test seront sensiblement égaux.



## Mise en place d’une meilleure performance de mon modèle

Afin de régler les hyperparamètres de mon algorithme de classification le plus performant, c’est-à-dire la régression logistique ; j’ai utilisé la recherche par quadrillage avec GridSearch. J’ai positionné une liste de possibilité pour chacun des hyperparamètres et j’ai entraîné mon modèle sur chacune des combinaisons pour calculer son score. A la fin, nous conservons uniquement le meilleur paramétrage.C’est une technique intéressante et performante mais qui a un très gros inconvénient : il faut être patient car mon modèle est entraîné sur toutes les combinaisons. Cependant, l’avantage, c’est que je n’ai à le faire qu’une seule fois.

Pour faire ces tests, j’ai utilisé la librairie Scikit-Learn qui fournit la fonction GridSearchCV et j’ai joué sur quatre hyperparamètres qui me semblaient pertinents : le solver, la pondération des classes, la pénalité et le C. Pour faire ce « Grid Search » avec Scikit-Learn cela il suffit de créer un dictionnaire Python (ici « grid ») avec les hyperparamètres à paramétrer et surtout les valeurs que nous voulons tester. J’ai mis en place une validation croisée à cinq folds représenté par cv=5.

Voici le résultat de notre tuning d’hyperparamètres avec GridSearchCV

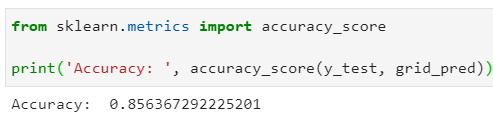
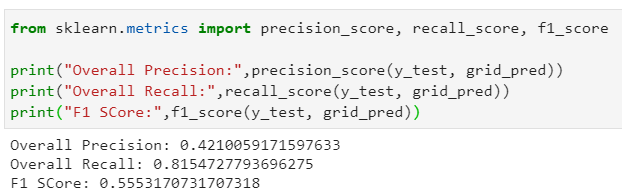


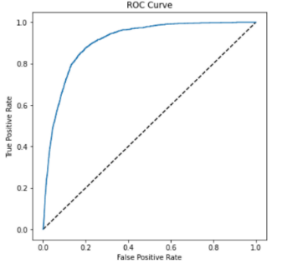
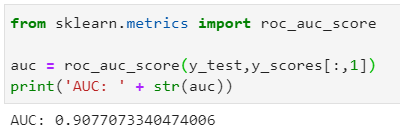
On remarque que notre cible « f 1 » est passé de 0.45 à 0.55, on a gagné 0.10 en f1-score ce qui est conséquent.

Notre meilleur set d’hyper paramètres se compose : du C = 0.001, class\_weight = « balanced », penalty = « l2 » et solver = « sag »

## Analyse des métriques après Optimisation des Hyperparamètres

Voici une synthèse des métriques : on constate une légère diminution de notre performance du modèle avec une diminution de 0.04. En revanche notre f1-score a augmenté de 0.10 et notre AUC est passé de 0.91 à 0.90.



 .

On observe que les modifications des hyperparamètres ont effectivement bien amélioré le niveau de performance de notre modèle d’Intelligence Artificielle

## Sauvegarde de mon modèle de classification

On effectue une sauvegarde de notre modèle à l’aide de la librairie pickle ; cette librairie permet de sauvegarder des structures de données dans un fichier pour pouvoir les recharger plus tard. Dans notre exemple, notre modèle est sauvegardé au format pickle (Le module pickle permet la sérialisation des objets mémoire en chaînes (et réciproquement). Il est utile pour la persistance et le transfert de données sur un réseau. La manipulation des données utilise un fichier par sérialisation, où les données sont stockées sous forme de chaînes.)

# Amélioration de l’application d’intelligence artificielle grâce à l’ajout d’une évolution fonctionnelle

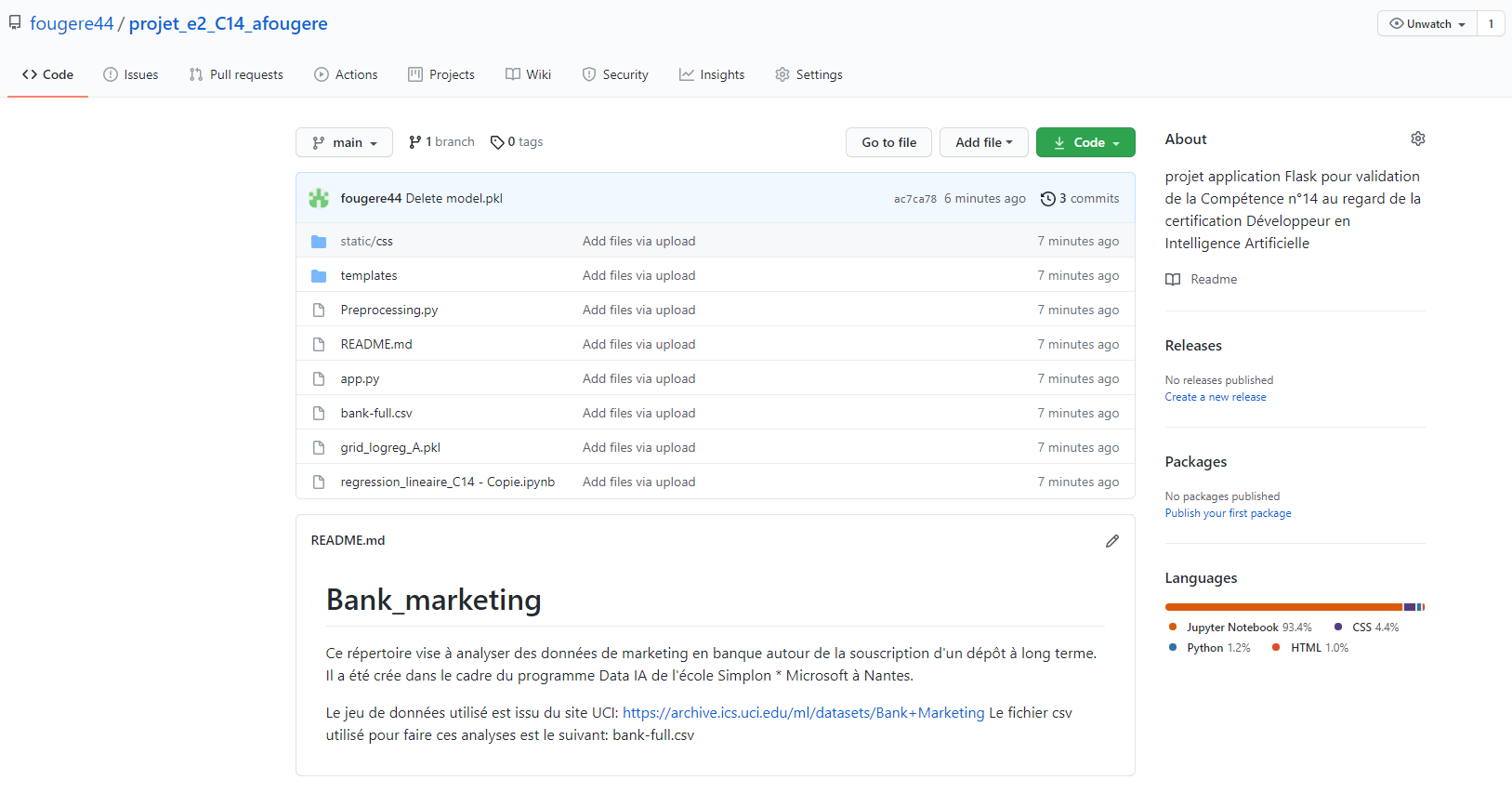
## Intégration de mon modèle d’IA dans une application existante

J’ai repris mon modèle d’intelligence artificielle de régression logistique lié au dataset « bank-full » et effectué l’entrainement sur trois variables :

* Age
* Balance
* Duration

J’ai ensuite utilisé la méthode dump afin de sauvegarder mon modèle appelé « grid\_logreg\_A.pkl » puis un load pour pouvoir télécharger et intégrer mon modèle au format pickle dans une application flask déjà existante.

J’ai créé un repository public sur mon GitHub personnel que vous pouvez aller explorer en suivant ce lien 🡪 <https://github.com/fougere44/projet_e2_C14_afougere.git>



## 

## Estimation de charge au regard du besoin fonctionnelle

Communiquer le besoin utilisateur/clientEstimer la charge nécessaire à cette intégration

## Intégration d’une évolution fonctionnelle dans mon modèle

Intégration dans le dataframe d’une nouvelle catégorie : puis faire un fit et dump du model, qui implique une évolution dans l’application (nouvelle fonctionnalité par suite de l’ajout de la catégorie)

## Test de la non-régression de l’application suite à l’intégration de cette évolution

Quels outils utilisés ?