OpenClassrooms

Note technique

Formation Data Scientist Projet 7

2023

Démarche d'élaboration d'un outil de scoring à destination de la société financière «Prêt à dépenser».

Introduction

Cette note technique détaille la démarche suivie pour développer un modèle de classification binaire permettant de prédire la probabilité de défaut de remboursement d'un client et de classer sa demande de crédit à la consommation comme «refusée» ou «acceptée».

But de la note technique

Présenter clairement toute la démarche d'élaboration du modèle de classification, de l'entraînement à l'analyse du data drift, en passant par l'optimisation et l'analyse des résultats et des métriques utilisées.

Audience cible

Collègues data scientists

Table des matières

Chapter N° 1

Méthodologie d'entraînement du modèle

Analyse exploratoire

Objectif:

Comprendre la structure des données d'entrée, téléchargées sur la plateforme Kaggle (*Données sources* (2018)), afin de déterminer les étapes de nettoyage nécessaires.

Nous avons à disposition 10 fichiers csv (voir table 1.1). Les demandes de crédit à classifier se trouvent dans le fichier application_train.csv avec des variables comportementales (âge, genre, emploi, logement ...) et les classes à prédire : 0 pour crédit accordé, 1 pour crédit refusé.

Le fichier application_test.csv contient également des demandes de crédit à classifier (sans étiquettes) et sera utilisé, non pas comme jeu de test, mais uniquement pour mesurer un éventuel data drift des données.

Les autres fichiers contiennent des informations relatives aux anciennes demandes de crédits faites par chaque client, auprès de la société "Prêt à dépenser" ou d'autres institutions financières.

Doublons : Aucun doublon n'a été détecté dans tous ces fichiers sources.

Valeurs manquantes : Il ne manque aucune valeur dans la colonne 'TARGET' du jeu d'entraînement application_train.csv. Par contre, il manque des valeurs dans de nombreuses colonnes (voir figure 1.1).

Fichier (.csv)	Clé primaire	Description	Format
application_train	SK_ID_CURR	demandes de crédit déposées auprès de la société "Prêt à dépenser".	lignes : 307,511 colonnes : 122
application_test	SK_ID_CURR	demandes de crédit déposées auprès de la société "Prêt à dépenser".	lignes : 48,744 colonnes : 121
bureau	SK_ID_BUREAU	demandes de crédit adressées à d'autres institutions financières	lignes : 1,716,428 colonnes : 17
bureau_balance	SK_ID_BUREAU MONTHS_BALANCE	mensualités associées aux crédits accordés par d'autres institutions financières	lignes : 27,299,925 colonnes : 3
POS_CASH_balance	SK_ID_PREV MONTHS_BALANCE	mensualités associées aux prêts à la consommation déjà accordés par la société "Prêt à dépenser"	lignes : 10,001,358 colonnes : 8
credit_card_balance	SK_ID_PREV MONTHS_BALANCE	mensualités associées aux prêts adossés à une carte de crédit, déjà accordés par la société "Prêt à dépenser"	lignes : 3,840,312 colonnes : 23
previous_application	SK_ID_PREV	anciennes demandes de crédit faites auprès de la société "Prêt à dépenser"	lignes : 1,670,214 colonnes : 37
installments_payments		échéancier des crédits déjà accordés par la société "Prêt à dépenser"	lignes : 13,605,401 colonnes : 8
sample_submission	SK_ID_CURR	exemples de résultats de classification	lignes : 48,744 colonnes : 2
HomeCredit_columns_description		description des variables de tous ces jeux de données	

TABLE 1.1 – Description des fichiers sources

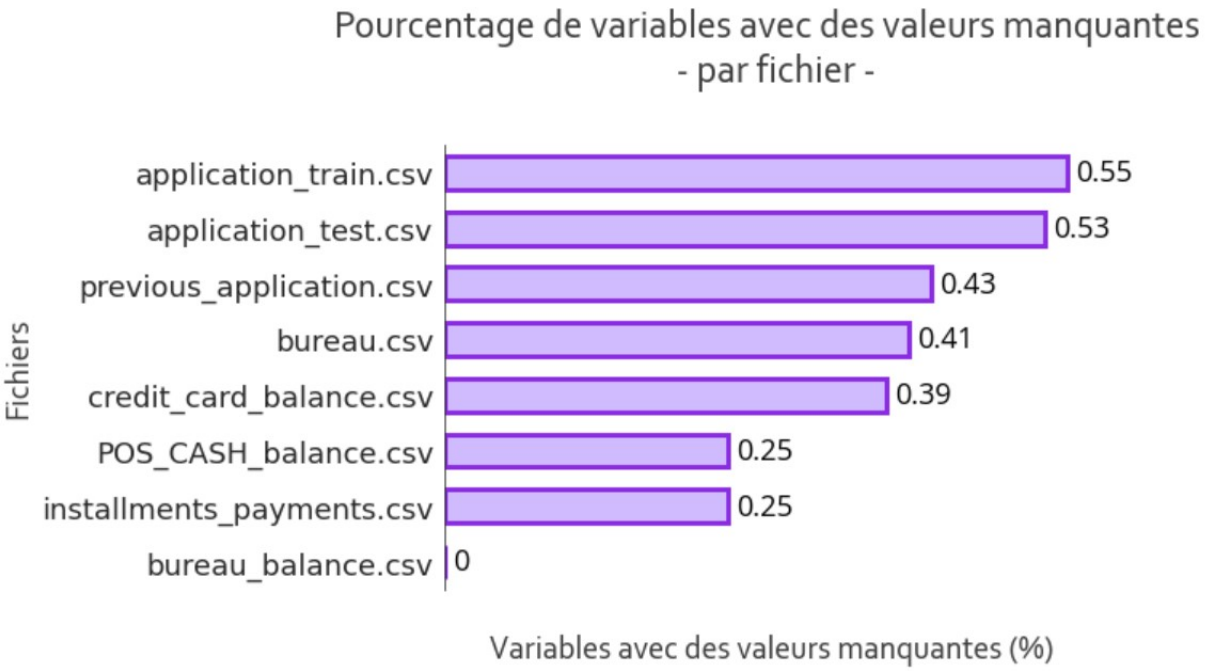

FIGURE 1.1 – Histogramme des valeurs manquantes

Traitement des données

Objectif: Agrégation, Nettoyage, Feature Engineering

Obtenir un unique jeu de données propre avec une ligne par demande à classifier (SK_ID_CURR).

Un *Kernel Kaggle* (2018) qui répond bien aux besoins de la mission a été utilisé pour cette étape. Chacun des fichiers csv présentés précédemment est nettoyé et agrégé de sorte à obtenir uniquement des jeux de données avec une ligne par 'SK_ID_CURR' (identifiant unique associé à chaque demande de crédit à classifier). De nouvelles variables sont également créées. (Voir annexe A , tableaux récapitulatifs du nettoyage de chaque fichier).

Puis tous les fichiers sont fusionnés en utilisant une jointure à gauche sur la clé 'SK_ID_CURR'. Le fichier contenant le jeu d'entraînement, application_train.csv, est le fichier le plus à gauche auquel tous les autres sont joints. L'ordre de fusion des autres fichiers n'a pas d'importance.

On obtient ainsi un nouveau jeu de données avec une ligne par demande de crédit adressée à la société «Prêt à dépenser», soit 307 507 lignes et 458 colonnes. L'identifiant 'SK_ID_CURR' est utilisé comme index.

Enfin, la colonne contenant les classes à prédire est enregistrée dans un fichier à part.

On répète ces mêmes étapes pour nettoyer le fichier application_test.csv.

Préparation des données

Objectif:

Obtenir un jeu d'entraînement digeste pour les algorithmes et les capacités de calcul

L'ordinateur utilisé pour élaborer le modèle n'a pas suffisamment de ressources pour travailler sur le jeu de données complet. Toute cette partie et l'entraînement du modèle ont donc été fait avec un échantillon représentatif, i.e. respectant le fort déséquilibre de la cible : 92% de classe 0 et 8% de classe 1. Cet échantillon correspond environ à la moitié du jeu de données complet.

Séparation du jeu de données :

Le jeu de données est séparé en deux avec stratification, à l'aide de la méthode train_test_split de **scikit-learn** : 70% des données serviront à l'entraînement et 30% à tester les modèles entraînés.

$$train\_test\_split(X, y, test\_size = 0.3, random\_state = 8, stratify = y) \tag{1.1}$$

Transformations :

Une instance de la classe OrdinalEncoder de la librairie scikit-learn est utilisée pour transformer en données numériques les variables catégorielles qui ne prennent que deux valeurs différentes. Ces deux valeurs seront respectivement remplacées par des 0 et des 1.

Concernant les 35 autres variables catégorielles, l'utilisation de la classe OneHotEncoder de la librairie scikit-learn fut envisagée dans un premier temps. Cependant, les temps de calcul étant trop long (plus de 30 minutes), la méthode *get_dummies* de la **librairie pandas** a finalement été retenue. En effet, cette méthode est bien plus rapide (1 minute) pour un résultat similaire.
La classe GetDummiesTransformer, appelant la méthode *get_dummies*, a donc été créée. Cette classe permet, comme la classe OneHotEncoder, de transformer des variables catégorielles en variables numériques, en créant, pour chaque variable, une colonne par catégorie présente dans le jeu d'entraînement. Ces nouvelles colonnes sont remplies avec des 0 et des 1 : 1 si l'individu concerné appartient à la catégorie, 0 sinon.

Les variables numériques sont, quant à elles, centrées-réduites à l'aide d'une instance de la classe RobustScaler de scikit-learn, qui soustrait la médiane et divise par l'écart inter-quartile.
De plus, une instance de la classe SimpleImputer, toujours de scikit-learn, a été utilisée pour imputer par la médiane et tester certains modèles qui ne gèrent pas les valeurs manquantes.

Enfin, une méthode de ré-échantillonnage est utilisée pour traiter le déséquilibre des classes. Elle est présentée plus en détail dans la partie 2.

Entraînement

Pipeline

Pour éviter la **fuite de données** du jeu d'entraînement vers le jeu de test, les transformations présentées précédemment (1) sont placées avec le modèle dans un objet Pipeline (voir figures 1.2 et 1.3)

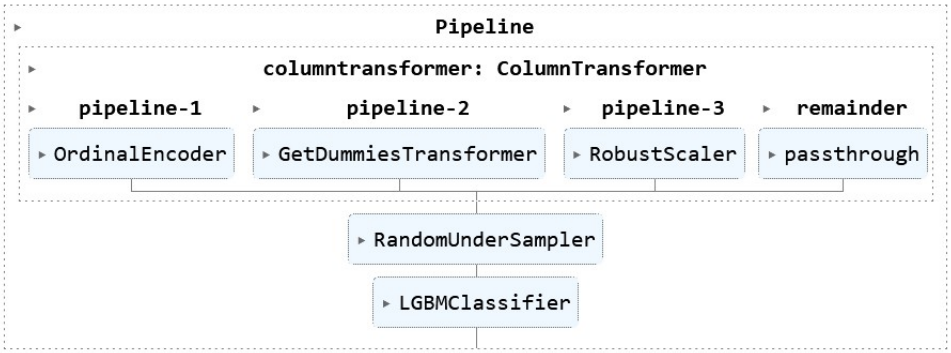

FIGURE 1.2 – Objet Pipeline modèle LGBM

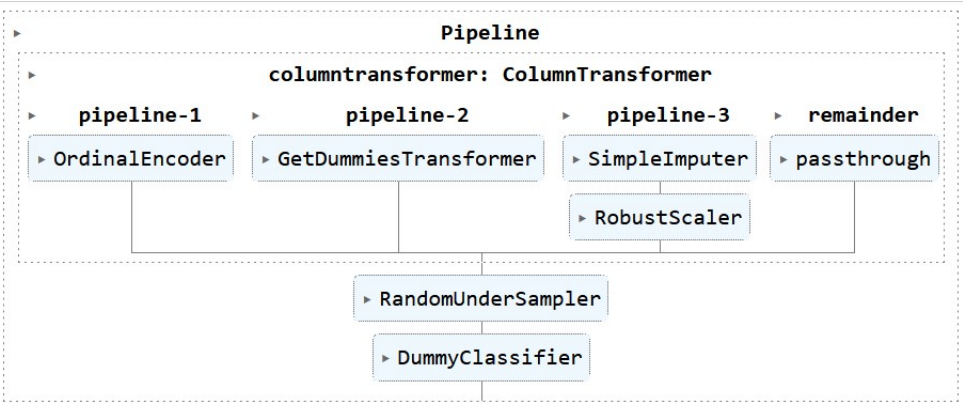

FIGURE 1.3 – Objet Pipeline avec SimpleImputer

Validation croisée

L'entraînement comporte une petite recherche d'hyperparamètres par validation croisée à l'aide de la classe GridSearchCV de **scikit-learn**. Pour chaque combinaison d'hyperparamètres testée, le jeu d'entraînement est séparé en 3 parties à peu près égales. Tour à tour chacune des parties est utilisée comme jeu de validation pour calculer un score métier (voir chapitre 3). Le modèle est entraîné avec les données des 2 parties restantes. Les résultats obtenus avec la combinaison d'hyperparamètres qui minimise le score métier moyen sont enregistrés via l'**outil MLFlow Tracking**. L'évaluation finale du modèle est faite sur le jeu de test et également enregistrée via MLFlow Tracking.

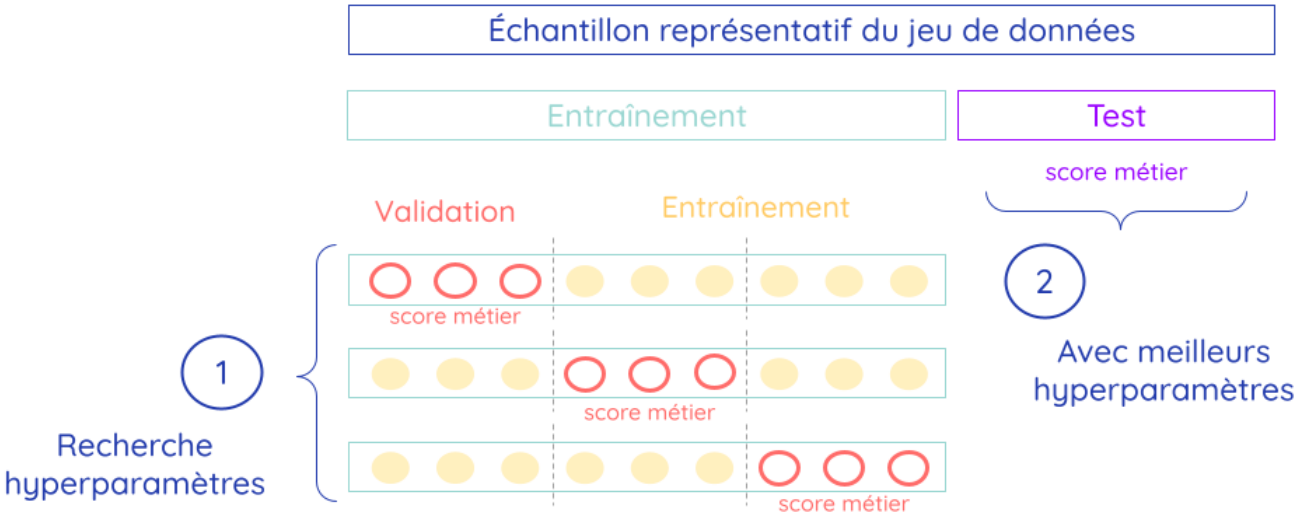

FIGURE 1.4 – Schéma d'entraînement

CHAPTER N° 2

Traitement du déséquilibre des classes

Méthodologie utilisée

Notre jeu de données est fortement déséquilibré car seules 8% des demandes de prêts ont été refusées (classe 1). Un modèle entraîné sur ce jeu déséquilibré risque d'ignorer complètement la classe 1 et donc d'être très mauvais pour identifier les demandes de crédit qu'il vaudrait mieux refuser. Or, ce sont ces demandes que l'on cherche précisément à identifier.

Objectif:

Rééquilibrer le jeu d'entraînement

Deux types de stratégies de ré-échantillonnage peuvent être utilisées :

- Ajouter des individus de la classe 1, qui est sous-représentée (Over Sampling)
- Supprimer des individus de la classe 0 (Under Sampling) ce qui entraîne une perte d'informations.

Stratégie retenue : classe RandomUnderSampler de la librairie python **"Imblearn"**

L'ordinateur utilisé pour entraîner le modèle n'ayant pas les ressources pour traiter un jeu de données trop gros, la stratégie de sous-échantillonnage a été retenue. On supprime simplement au hasard des individus de la classe majoritaire (voir figure 2.1). À noter que cette méthode simple et rapide a l'avantage d'être applicable sur les jeux de données contenant des valeurs manquantes. De plus, la combiner avec une technique de sur-échantillonnage type SMOTE, ne donne pas de meilleurs résultats et augmente grandement les temps de calcul.

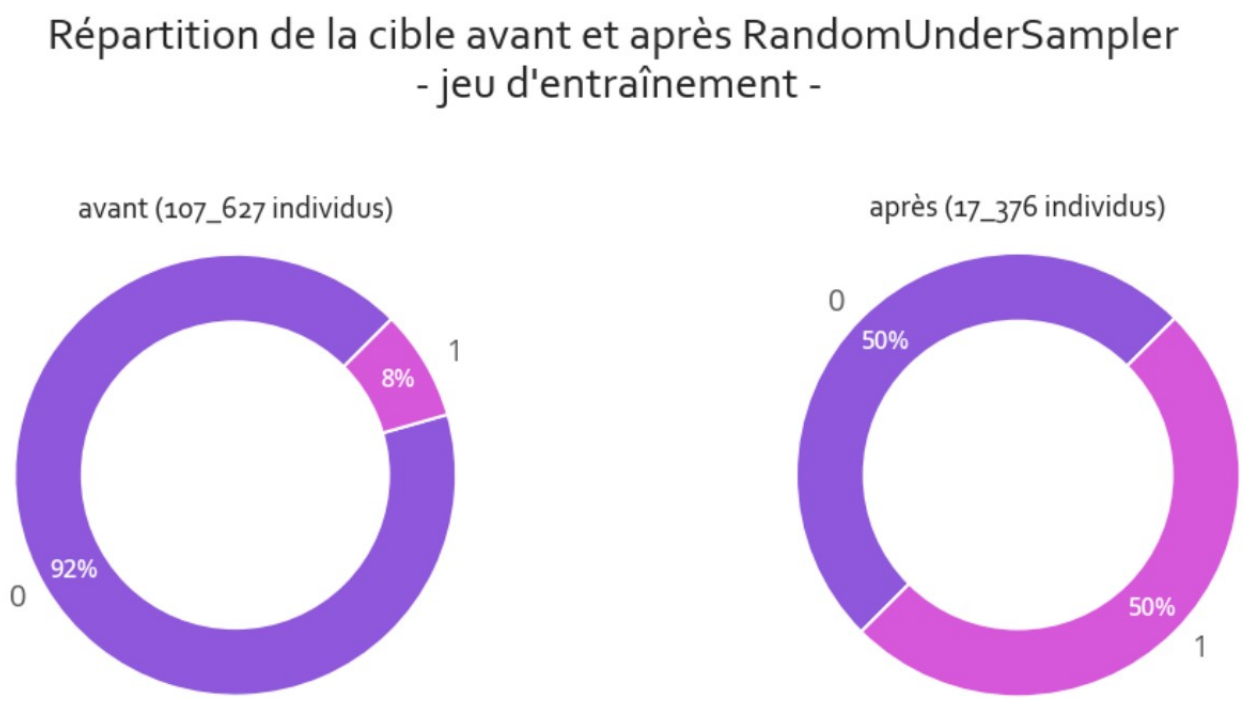

FIGURE 2.1 – RandomUnderSampler sur jeu d'entraînement

CHAPTER N° 3

Optimisation du modèle

Le modèle est optimisé en deux temps :

- lors de l'entraînement, pour choisir le meilleur modèle, avec GridSearchCV de **scikit-learn**
- une fois le modèle choisi, pour limiter le sur-apprentissage, avec la librairie **HyperOpt**

Fonction de coût métier

Pour choisir et optimiser le meilleur modèle de classification, nous allons chercher à minimiser un score métier.

Objectif:

Minimiser le coût d'une prédiction erronée pour la société Prêt à dépenser.

Ce score métier doit prendre en compte le fait que classer un client avec un haut risque de défaut de remboursement en "bon" client entraîne une perte en capital dix fois plus grande que le manque à gagner en marge dû au fait de classer un "bon" client en "mauvais" client.

Autrement dit, le coût d'un faux négatif (FN) pèse 10 fois plus que le coût d'un faux positif (FP) (voir matrice de confusion 3.1). Donc le coût total est égal à 10*FN + FP. On divise par 10 pour obtenir un coût entre 0 et 1.

$$Coût\ Total = \frac{10 * FN + FP}{10} \tag{3.1}$$

Optimisation du seuil :

Ce score métier est calculé en optimisant le seuil au dessus duquel la demande de crédit est rejetée (classe 1). Pour chaque valeur de seuil testée, nous allons :

- remplacer la probabilité d'appartenir à la classe 1 par un label,
  - ◇ 1 si probabilité > seuil
  - ◇ 0 sinon
- calculer un score métier avec ces labels

On conserve le seuil qui minimise le score métier, pour faire des prédictions sur de nouvelles données.

	Clients prédits en défaut (1)	Clients prédits sans défaut de remboursement (0)
Clients réellement en défaut	Vrais Positifs	**Faux Négatifs**
Clients qui remboursent	Faux Positifs	Vrais Négatifs

TABLE 3.1 – Matrice de confusion

Rappel : classe 0 = "bon" client, classe 1 = "mauvais" client

Autres métriques d'évaluation

La sélection du meilleur modèle a été faite en prenant également en compte d'autres métriques, calculées de sorte à rester pertinentes malgré le fort déséquilibre du jeu de données.

Balanced Accuracy : À maximiser

Elle est calculée à partir de deux mesures complémentaires : la sensibilité et la spécificité.

$$Balanced\ Accuracy = \frac{Sensitivity + Specificity}{2} \tag{3.2}$$

La sensibilité, ou recall, correspond à la proportion de positifs (dans notre cas, classe 1, clients en défaut) correctement identifiés par le modèle.

$$Sensitivity = \frac{TP}{TP + FN} \tag{3.3}$$

La spécificité correspond à la proportion de négatifs (classe 0, clients qui rembourseront totalement leur emprunt) correctement identifiés.

$$Specificity = \frac{TN}{FP + TN} \tag{3.4}$$

AUC : À maximiser

Le score AUC mesure l'aire sous la courbe ROC. Un score AUC égale à 0.5 correspond à un classifieur aléatoire, un score de 1 correspond à un classifieur parfait.

La courbe ROC montre, pour chaque seuil de décision possible, comment la sensibilité évolue en fonction de l'anti-spécificité, i.e. 1-spécificité. (voir figure 3.1).

Le score AUC est donc également calculé à partir des deux mesures complémentaires spécificité et sensibilité, ce qui lui permet aussi de prendre en compte le déséquilibre du jeu de données.

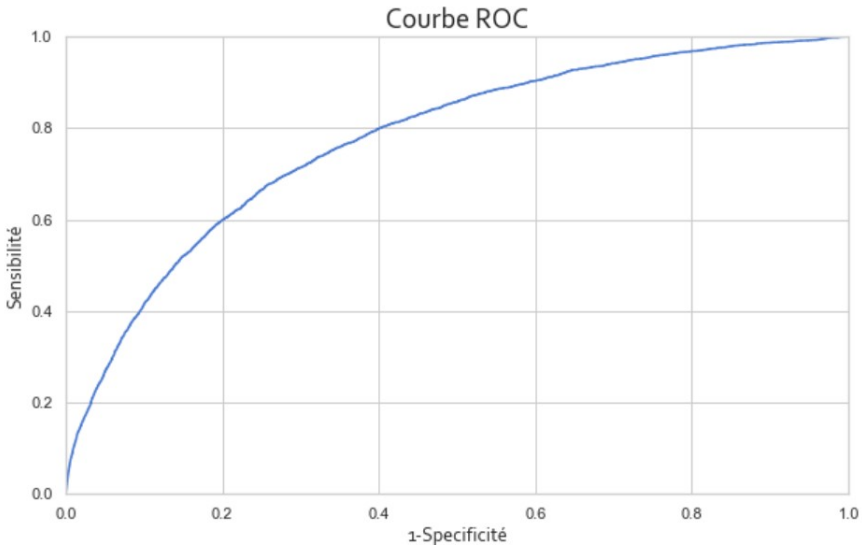

FIGURE 3.1 – Courbe ROC

Algorithme d'optimisation

Une fois le modèle sélectionné (LightGBM), la librairie **HyperOpt** d'**optimisation baysienne** a été utilisée pour trouver la combinaison d'hyperparamètres qui minimise le plus le score métier. Au lieu de tester exhaustivement toutes les combinaisons d'hyperparamètres de l'espace de recherche, cette librairie utilise les itérations précédentes et les probabilités conditionnelles pour déterminer les prochaines combinaisons à tester.

Chapter N° 4

Tableau de synthèse des résultats

Après étude de plusieurs modèles de classification, notre choix s'est porté sur le **LightGBM**.

Ce dernier obtient les meilleurs résultats sur le jeu de test pour le score métier (custom_score) et la balanced accuracy (voir figure 4.1).

Il est plus rapide que les autres modèles (voir figure 4.2) et gère les valeurs manquantes. Nul besoin d'imputer par la médiane donc, ce qui aurait pu beaucoup modifier la distribution des variables.

model	threshold	custom_score	accuracy_score	balanced_accuracy_score	roc_auc_score
lgbm	0.547	0.050655	0.742813	0.703593	0.703593
gboost	0.533	0.050815	0.735355	0.703211	0.703211
catboost	0.523	0.052346	0.717708	0.695082	0.695082
xgboost	0.619	0.054056	0.770650	0.679910	0.679910
adaboost	0.496	0.054381	0.682327	0.685268	0.685268
bagging	0.690	0.062273	0.872089	0.619838	0.619838
random_forest	0.694	0.062936	0.874626	0.615461	0.615461
balanced_random_forest	0.690	0.069696	0.900490	0.570865	0.570865
logistic_reg	0.506	0.077481	0.866149	0.524875	0.524875
dummy	0.001	0.080735	0.919265	0.500000	0.500000

FIGURE 4.1 – Résultats sur le jeu de test

Remarque : "threshold" correspond au seuil optimisé utilisé pour classer une demande en 0 ou 1.

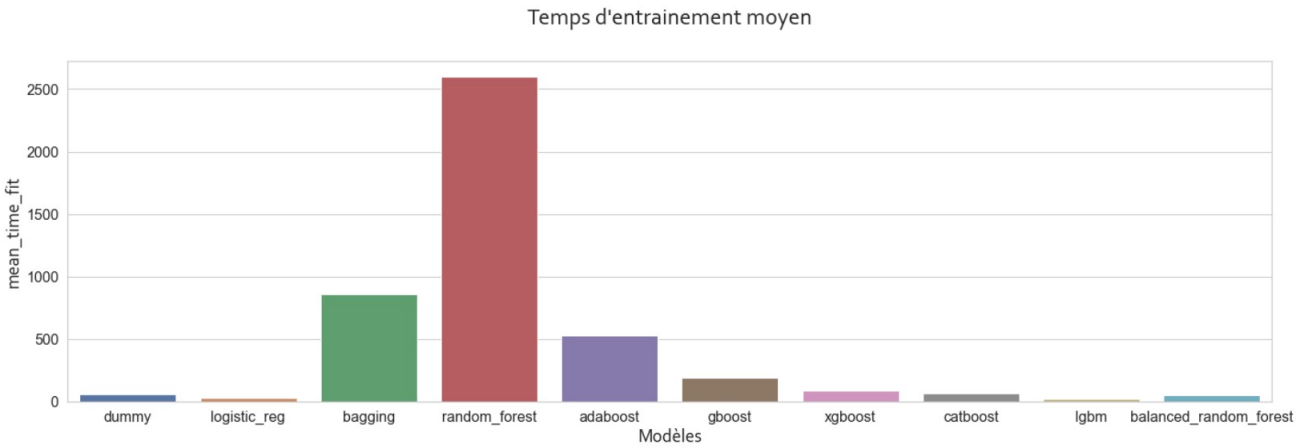

FIGURE 4.2 – Temps d'entraînement des modèles testés

Chapter N° 5

Interprétabilité du modèle

Interprétabilité globale

La **librairie lgbm** permet de facilement classer les features de la plus utilisée par le modèle à la moins utilisée (voir figure 5.1).

L'analyse de ce classement révèle que le modèle utilise en priorité :

- les notes attribuées aux clients par d'autres institutions financières (EXT_SOURCE). Ces notes sont normalisées entre 0 et 1, mais nous n'avons pas plus de détail sur leur source ni sur la façon dont elles ont été calculées.
- le taux de remboursement (PAYMENT_RATE), variable ajoutée lors du feature engineering (montant des annuités/montant du crédit)
- l'âge de l'emprunteur

À noter que seules 233 des 725 features sont utilisées, soit 32% des features. Une piste d'amélioration consisterait donc à supprimer toutes ces features inutiles et à ré-entraîner le modèle.

Interprétabilité locale

Le modèle est destiné à des chargés de relation client qui doivent être en mesure d'expliquer de manière transparente ses prédictions.

Un dashboard permettant d'interpréter les prédictions du modèle a donc été développé et mis en production. Ce dashboard contient un diagramme en cascade, construit à l'aide de la **librairie shap**, qui permet de visualiser l'impact de chaque variable sur une prédiction donnée.

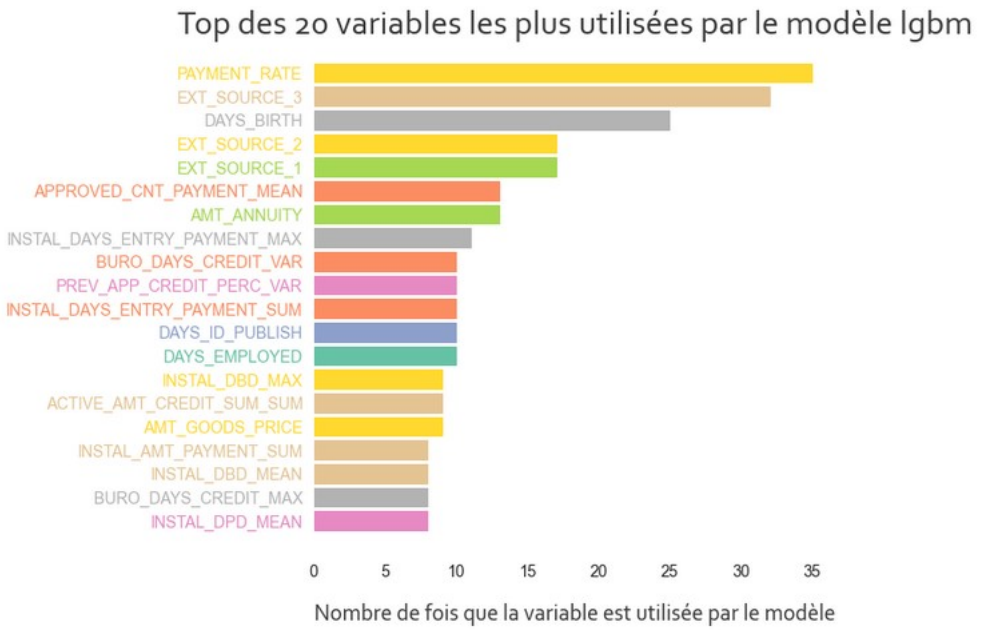

FIGURE 5.1 – 20 features les plus utilisées par le modèle LGBM

Exemple d'interprétation (figure 5.2) :

La faible notation (0.36) attribuée au client par l'institution financière 2 (EXT_SOURCE_2) fait augmenter sa probabilité prédite de défaut de remboursement. À l'inverse, la note plutôt élevée (0.74) attribuée par l'institution financière 3 (EXT_SOURCE_3) fait baisser sa probabilité prédite de défaut.

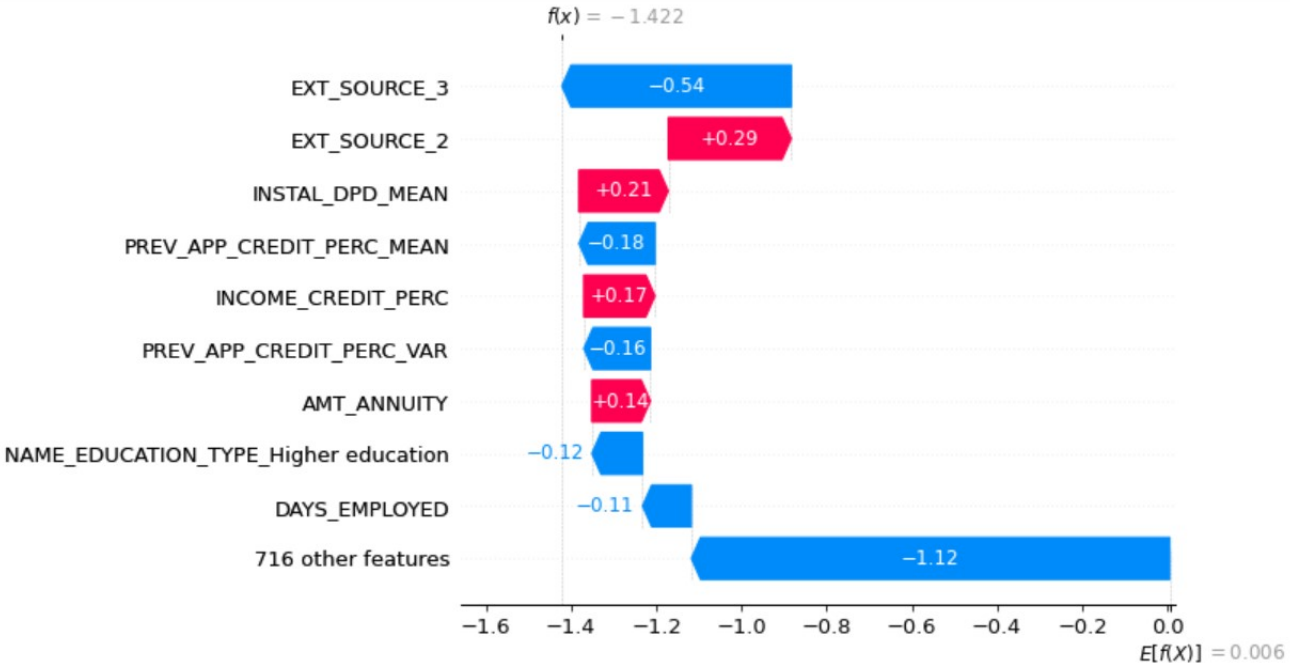

FIGURE 5.2 – Impact de chaque variable sur une prédiction - Visualisation Shap

CHAPTER N° 7

Analyse du datadrift

Définition

Le data drift (ou dérive de données) survient quand les données soumises au modèle en production diffèrent de façon trop importante des données utilisées pour entraîner, tester et valider le modèle avant de le déployer (*Définition du data drift* (2023)).

Si ce phénomène n'est pas identifié à temps, les prédictions du modèle risquent d'être erronées et la société financière "Prêt à dépenser" risque d'accorder des crédits à des clients, qui présentent en réalité une forte probabilité de défaut.

Données d'entrée

La **librairie evidently** a été utilisée pour générer un rapport HTML permettant de détecter et d'explorer les éventuels changements entre :

- les données d'entraînement : un échantillon du jeu de données "application_train.csv".
- et les données de production : un échantillon du jeu de données "application_test.csv".

Remarque : des échantillons ont été utilisés car les temps de calcul sont très longs sur les jeux de données complets (plus de 5 heures).

Algorithme

Pour chaque variable, l'algorithme compare la distribution des valeurs d'entraînement avec la distribution des valeurs de production. Pour déterminer si ces deux distributions sont significativement différentes, il applique le **test statistique** qu'il juge le plus adapté selon le type de la variable, le nombre de valeurs différentes possibles que peut prendre la variable et la taille du jeu d'entraînement. Par défaut, tous les tests sont fait avec un niveau de confiance à 0.95. Dans notre cas, avec un échantillon de 500 observations, les tests statistiques suivants sont utilisés :

- features numériques :
  - ◇ plus de 5 valeurs différentes possibles : Test de Kolmogorov-Smirnov
  - ◇ de 2 à 5 valeurs différentes possibles : Test du chi-2
  - ◇ une seule valeur possible : Test de comparaison de proportions basé sur le Z-score
- features catégorielles :
  - ◇ binaires : Test de comparaison de proportions basé sur le Z-score
  - ◇ plus de 2 valeurs différentes possibles : Test du chi-2

Si la distribution de production est significativement différente de la distribution d'entraînement pour au moins 50% des features, Evidently considère qu'il y a du data drift. À noter que l'on aurait pu implémenter un seuil plus élevé. (Pour plus de détails voir *Documentation Librarie Evidently* (2023)).

Comment lire le rapport

Seuls 13.5% de nos features ont "dérivé" ce qui est bien en dessous du seuil de 50%. On peut donc considérer qu'il n'y a pas de data drift entre ces deux échantillons de données (voir 6.1). Chaque ligne du rapport correspond à une variable, pour laquelle est renseigné :

- Column : le nom de la variable
- Type : le type de la variable
- Reference distribution : la distribution d'entraînement
- Current distribution : la distribution de production
- Data Drift : la présence ou non de datadrift
- Stat Test : le nom du test statistique utilisé pour comparer les distributions
- Drift Score : le résultat du test statistique (dans notre cas, une p-value)

$$p\_value < 0.05 => Drift\ Detected \tag{6.1}$$

Ce rapport est interactif, il est donc possible de trier les variables pour, par exemple, afficher en premier les variables ayant dérivée.

En déroulant chaque variable, il est possible d'afficher un ou deux graphiques interactifs selon le type de la variable :

- Un histogramme représentant les distributions des données d'entraînement et de production
- Un graphique type nuage de points illustrant le drift (uniquement pour les features numériques).
  - ◇ Points rouges : données du jeu de production
  - ◇ Droite verte : moyenne du jeu d'entraînement
  - ◇ Zone verte : écart type du jeu d'entraînement

Parmi les 62 variables présentant du drift, seules 40 sont vraiment utilisées par le modèle. Malheureusement, la variable "PAYMENT_RATE", qui se trouve être la variable la plus utilisée par le modèle, en fait partie. Cette variable ayant un impact très important sur les prédictions, on peut se demander s'il ne vaudrait pas mieux ré-entraîner le modèle en dépit de la conclusion du rapport.

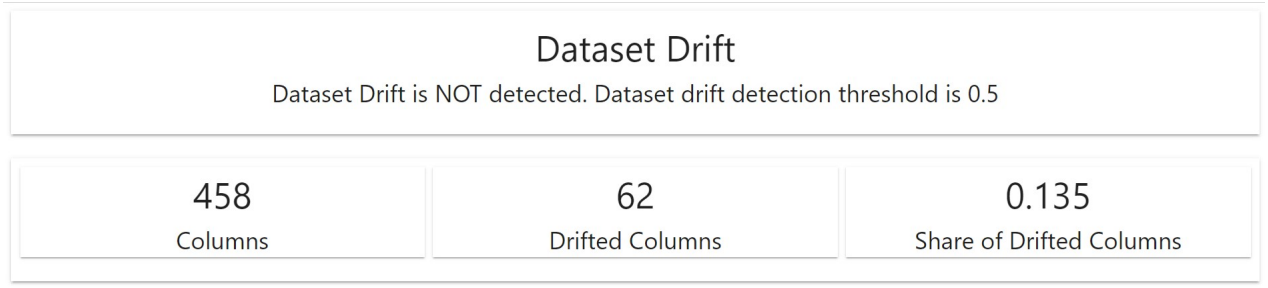

FIGURE 6.1 – Entête du rapport HTML Evidently

Chapter N° 6

Limites et améliorations

Travailler sur le jeu complet :

L'ordinateur utilisé pour entraîner le modèle n'étant pas assez puissant, seule la moitié du jeu de données a été traitée. Le principal axe d'amélioration consisterait donc à entraîner le modèle sur le jeu de données complet. Toujours selon cette même problématique, le nombre de combinaisons d'hyperparamètres testées lors de la phase d'optimisation est assez restreint. Il serait intéressant d'étendre l'espace de recherche et d'augmenter le nombre de tests.

Réduire le nombre de variables :

L'interprétabilité du modèle pourrait être améliorée en réduisant le nombre de variables. On pourrait supprimer progressivement les variables les moins utilisées.

Renommer les variables :

Une piste d'amélioration serait de renommer les 233 variables utilisées par le modèle de sorte à les rendre plus intelligibles pour les chargés de clientèle.

Personnaliser le pré-traitement des données :

Les étapes nécessaires à l'élaboration du modèle (analyse exploratoire, préparation des données, feature engineering) ont été abordées de façon superficielle en réutilisant un kernel Kaggle. Il serait peut-être possible d'améliorer les résultats du modèle en faisant plus de recherches et en créant de nouvelles variables en collaboration avec les équipes métiers.

Intégrer la détection de datadrift au pipeline :

Nous pourrions mettre l'outil de détection du data drift dans le pipeline. En cas de data drift, le modèle serait ré-entraîné automatiquement.

Chapter N° 8

Annexes

Nettoyage des données sources

Fichier Source : application_train.csv ou application_test.csv	
Étape 1 : Nettoyage	Étape 2 : Feature Engineering
CODE_GENDER : si ni F ni M alors supprime DAYS_EMPLOYED : remplace 365243 par nan	Ajout de ratios : DAYS_EMPLOYED_PERC : pourcentage de jours vécus en tant qu'employé. INCOME_CREDIT_PERC : ratio revenu total sur montant du crédit demandé INCOME_PER_PERSON : ratio revenu total sur nombre de membres de la famille ANNUITY_INCOME_PERC : ratio montant des annuités sur revenu total PAYMENT_RATE : ratio montant annuités sur montant du crédit demandé
Fichier nettoyé : une ligne par SK_ID_CURR	

Table A.1 – Nettoyage demandes de prêt à classifier

Fichier Source : bureau_balance.csv et bureau.csv	
Étape 1 : Agrégation et Fusion	Étape 2 : Feature Engineering
Agrège les données mensuelles de bureau_balance.csv par SK_ID_BUREAU (id des demandes de prêt adressées à d'autres institutions financières). Fusionne avec bureau.csv sur SK_ID_BUREAU. Agrège ce nouveau fichier par SK_ID_CURR (id des demandes de prêt à classifier). Fonctions d'agrégation : - variables catégorielles : mode - variables numériques : min, max, var, mean, sum	Ajoute les mêmes variables numériques filtrées sur les prêts encore actifs : ACTIVE_ + nom variable numérique existante : Ajoute les mêmes variables numériques filtrées sur les prêts clôturés : CLOSED_ + nom variable numérique existante
Fichier nettoyé : une ligne par SK_ID_CURR	

TABLE A.2 – Nettoyage demandes de prêt à d'autres institutions financières

Fichier Source : previous_application.csv		
Étape 1 : Nettoyage	Étape 2 : Agrégation	Étape 3 : Feature Engineering
Variables représentant des nombres de jours : remplace les 365243 par nan DAYS_FIRST_DRAWING DAYS_FIRST_DUE DAYS_LAST_DUE_1ST_V DAYS_LAST_DUE DAYS_TERMINATION	Agrège les anciennes demandes de prêts par SK_ID_CURR Fonctions d'agrégation : - variables catégorielles : mode - variables numériques : min, max, var, mean, sum	Ajoute APP_CREDIT_PERC : montant demandé sur montant reçu Ajoute les mêmes variables numériques filtrées sur les anciennes demandes acceptées : APPROVED_ + nom variable numérique existante Ajoute les mêmes variables numériques filtrées sur les anciennes demandes refusées : CLOSED_ + nom variable numérique existante
Fichier nettoyé : une ligne par SK_ID_CURR		

TABLE A.3 – Nettoyage anciennes demandes de prêt

Fichier Source : POS_CASH_balance.csv	
Étape 1 : Agrégation	Étape 2 : Feature Engineering
Agrège les données mensuelles par SK_ID_CURR (id des demandes de prêt à classifier). Fonctions d'agrégation : - variables catégorielles : mode - variables numériques : min, max, size	Ajoute le nombre de mensualités (pour des prêts à la consommation) par demande de prêt actuelle POS_COUNT
Fichier nettoyé : une ligne par SK_ID_CURR	

Table A.4 – Nettoyage mensualités prêts à la consommation

Fichier Source : installments_payments.csv	
Étape 1 : Feature Engineering	Étape 2 : Agrégation
PAYMENT_PERC : ratio montant versé sur montant dû PAYMENT_DIFF : différence montant dû moins montant payé DPD : nombre de jours entre la date due et la date de versement effectif (days past due) DBD : nombre de jours entre la date de versement et la date due (days before due) INSTAL_COUNT : nombre de versements par SK_ID_CURR	Agrège les paiements par SK_ID_CURR (id des demandes de prêt à classifier). Fonctions d'agrégation : - variables catégorielles : mode - variables numériques : min, max, sum, var
Fichier nettoyé : une ligne par SK_ID_CURR	

Table A.5 – Nettoyage échéancier de versements

Fichier Source : credit_card_balance.csv	
Étape 1 : Agrégation	Étape 2 : Feature Engineering
Agrège les cartes de crédit par SK_ID_CURR (id des demandes de prêt à classifier). Fonctions d'agrégation : - variables catégorielles : mode - variables numériques : min, max, sum, var, mean	Ajoute le nombre de cartes de crédit par client CC_COUNT
Fichier nettoyé : une ligne par SK_ID_CURR	

Table A.6 – Nettoyage mensualités prêts adossés à carte de crédit

References

Définition du data drift (2023). https://datascientest.com/definition-data-drift.
Documentation Librarie Evidently (2023). https://docs.evidentlyai.com/reference/data-drift-algorithm.
Données sources (2018). https://www.kaggle.com/c/home-credit-default-risk/data.
Kernel Kaggle (2018). https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features.