Note méthodologique

Alexandre VERDONCK

15 avril 2020

Table des matières

1 Introduction

Cette section nous permettra d'étayer le contexte du prêt bancaire d'un point de vue macro, puis plus spécifiquement le contexte particulier de notre client. Nous analyserons ensuite son besoin de manière formelle et structurée.

1.1 Contexte

Depuis le début des années 2000, les taux d'intérêt des obligations d'état n'a cessé de diminuer avant d'atteindre en 2015 des valeurs négatives en Allemagne, en France et en Suisse notamment.Cet abaissement des taux d'intérêts globaux se répercute jusqu'aux particuliers avec des taux d'intérêts parfois inférieurs à 1%. Ces faibles taux continuent de favoriser l'emprunt à l'échelle globale.

En particulier, une entreprise de prêt à la consommation s'expose à des risques plus importants. A fortiori, lorsqu'on s'intéresse à une base de clients ayant peu ou pas d'historique de prêt comme c'est le cas ici, le risque d'impayé est significativement plus élevé.

Le besoin de qualification du demandeur est essentiel pour assurer la rentabilité de l'entreprise. Cette qualification du demandeur ne doit en outre pas être discriminatoire. Dès lors, pour chaque variable incluse dans le modèle

et considérée comme discriminante l'exploitant doit être en mesure de démontrer une différence significative dans la probabilité de défaillance de payement.

1.2 Analyse du besoin

— Élaboration d'un modèle de prédiction de la probabilité de défaillance de payement, sur base des caractéristiques d'un demandeur.
— Intepretabilité du modèle et de ses prédictions : les variables utilisées ne seront pas "poolées" afin de garder un contrôle sur leur valeurs individuelles.
— Aide à la décision sur base d'une métrique business qui permet de décider d'accord ou non le prêt.

2 Modélisation

Définition formelle du problème L'objectif de l'algorithme est de lier une matrice de caractéristiques X à une variable dépendante binaire y par la probabilité qu'une observation x appartiennent à l'une des deux classes. La fonction $h(x)$ prédit la probabilité que y appartiennent à la classe 1 en fonction du vecteur x_j.

Les données Issues du client, fournissant 7 datasets différents, dont le schéma relationnel est indiqué en figure 1 :

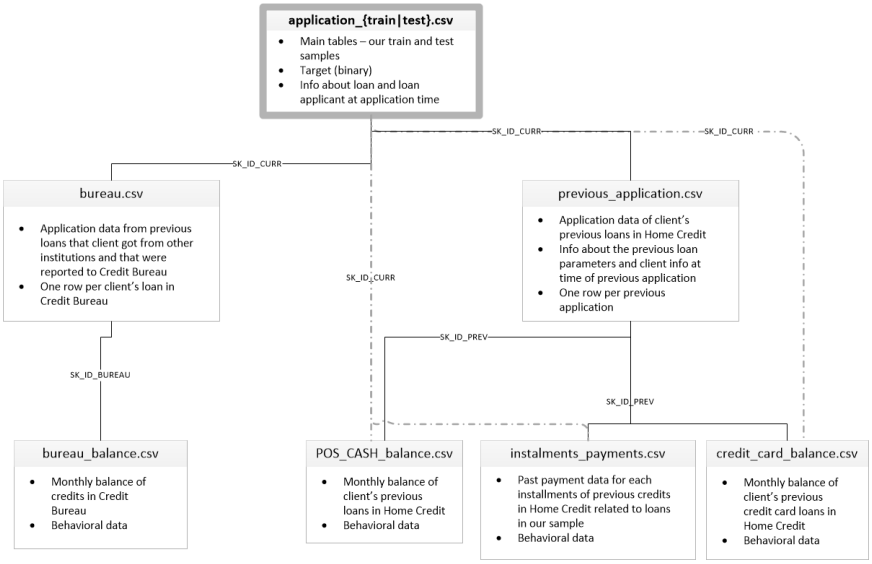

FIGURE 1 – Schéma relationnel de la DB

On dénombre 307511 observation dans le set d'entraînement, le tableau 1 reprend les volumes de données des datasets qui se rapportent au premier :

Dataset	Nombre de lignes	Nombre de variables
bureau.csv	1.716.428	17
bureau_balance.csv	27.299.925	3
previous_application.csv	1.670.214	37
installments_payments.csv	13.605.401	8
POS_CASH_balance.csv	10.001.358	8
credit_card_balance.csv	3.840.312	23

TABLE 1 – Dimensions des différents set de données

La variable dépendante (target) La target est issue du dataset *application_train.csv* et est lié aux index *SK_ID_CURR*. Celle-ci représente le fait que l'emprunt concerné ait eu un défaut de payement critique. (1 = défaut de payement, 0 = pas de défaut)

2.1 Pré-traitement

Le pré-traitement des données suit la procédure développée par Aguiar sur le lien suivant : lien vers le Kernel de Aguiar. L'idée de cette procédure est d'agréger les données autour de l'ID du prêt (SK_ID_CURR) demandé.

L'agrégation génèrent plusieurs statistiques comme le Min, le Max, la moyenne et/ou la médiane.
En concaténant notre train set (125 variables) (avec ces informations, on obtient un total de 674 variables définies sur 307.511 observations.

2.2 Sélection des variables

La sélection de variables peut suivre 3 grands principes, combinables [4] :

1. Sélection filtrante (filter method)
2. Sélection enveloppante (wrapper method)
3. Sélection imbriquées (embedded method)

Le sous-ensemble de variables recherché doit avoir un nombre limité de variables ($p < 50$) et ce pour limiter la complexité du modèle et faciliter son interprétation. **Le modèle développé utilise une stratégie de filtrage parce que c'est la seule méthode qui n'implique pas de processus itératif dans la sélection de variables.** Ce serait ici trop gourmand en temps de calcul puisque nous avons une dataset de forme $(+300.000, +600)$.

La sélection filtrante est effectuée avec un algorithme linéaire de type Lasso [7] dont on rappelle l'équation régularisée :

$$\hat{\beta} = argmin\|y - X\beta\|_2^2 + \lambda\|\beta\|_1$$

Le Lasso tend à annuler les coefficients β là où une régression Ridge tend à les diminuer. Le paramètre λ pénalise les valeurs absolues des coefficients β afin d'abaisser leurs normes. Le résultat obtenu, avec un paramètre $\lambda = 0.95$ est un subset de de 143 features au lieu de 674.

2.3 Algorithme

Du fait du volume de données important et de la grande dimensionalité du dataset, il sera essentiel de choisir un implémentation performante. Nous nous orientons vers un algorithme de Gradient Boosting ayant démontré une performance supérieure dans la grande majorité des applications et certainement lorsque les problèmes à résoudre sont à haute dimensions et non-linéaires.

Nous utiliserons l'implémentation LightGBM, dont la performance en temps est 10x supérieure aux implémentations classique GBDT [1]. Ses principales caractéristiques sont expliquées en section 2.3.1.

2.3.1 Light GBM

L'algorithme Gradient Boosting Decision Tree (GBDT) est très populaire et bénéficie de quelques implémentations efficaces que sont XGBoost et pGBRT. Même si ces implémentations sont largement adoptées dans le domaine et considérées comme des *quick-win algorithms*, leur performance et leur scalabilité sont insatisfaisante lorsque la dimensionalité ou la taille des données augmente.[1]

L'implémentation lightGBM diffère des méthodes précitées grâce à deux techniques :
— **Le Gradient-based One-Side Sampling (GOSS)** exclut une proportion importance des instances (observations) ayant des faibles gradients, c'est à dire qui diffèrent peu. Seule les instances ayant de forts gradients sont utilisées pour l'estimation du Information Gain [8] dans la sélection des features.

— **l'Exclusive Feature Bundling (EFB)** crée des paquets de features mutuellement exclusives (c'est-à-dire qu'elles ne prennent jamais ou rarement des valeurs nulles simultanément) pour réduire leur nombre.

En outre, LightGBM traite nativement les variables catégoriques.

2.3.2 Fonction coût : LogLoss

La fonction coût est une règle de calcul de pénalité associée à une erreur de prédiction. Elle prend en entrée les vraies valeurs y et les valeurs prédites $\hat{y}$. On peut toujours calculer le coût sur une observation (Eq. 1), ou le moyenner sur une série d'observations (Eq. 2) :

$$cost_i = f(y_i, \hat{y}_i) \tag{1}$$

$$cost = \frac{1}{n}\sum_{1}^{n} f(cost_i) \tag{2}$$

Pour la classification binaire, nous avons deux méthodes principales de pénalisation :
— La pénalisation de l'erreur probabiliste $\epsilon_p = f(h(x), y)$,
— La pénalisation de l'erreur binaire $\epsilon_b = f(y, \hat{y})$.

En minimisant cette dernière, on modifie deux paramètres que sont : la surface de modélisation h(x) qui donne la probabilité de défaut de payement mais également le threshold τ au delà duquel un observation est considérée comme un prêt trop risqué (=1). Ce threshold devrait pourtant être variable en fonction du contexte : taux d'intérêts, propension de la banque à prendre des risques en fonction de sa propre situation etc.

Il est donc plus judicieux de créer une fonction coût qui minimise l'erreur probabiliste, permettant d'avoir un qualification précise du prospect. L'exploitant peut alors faire varier le seuil auquel il décide d'accorder le crédit, et ce en fonction des conditions actuelles.

Le choix s'est donc porté sur une fonction coût de type log-loss , illustrée en figure 2.3.2 :

$$H(p,q) \;=\; -y \log \hat{p} - (1-y)\log(1-\hat{p})$$

— Si la vraie valeur est 1, l'erreur vaut : $-y \log \hat{p}$
— Si la vraie valeur est 0, l'erreur vaut alors :
$-(1-y)\log(1-\hat{p})$

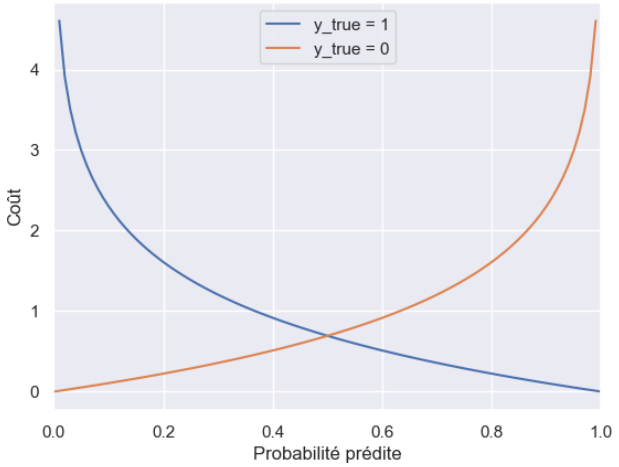

FIGURE 2 – Fonction Coût Logloss

Il est important de noter que, malgré le déséquilibre de classes, il est recommandé de ne pas activer les paramètres d'*imbalance* lorsqu'on prédit les probabilités de classes et non les classes directement [1].

2.3.3 Analyse de la qualité du modèle

1. Documentation de LightGBM : "Note, that the usage of all these parameters will result in poor estimates of the individual class probabilities. You may want to consider performing probability calibration (`https://scikit-learn.org/stable/modules/calibration.html`) of your model.

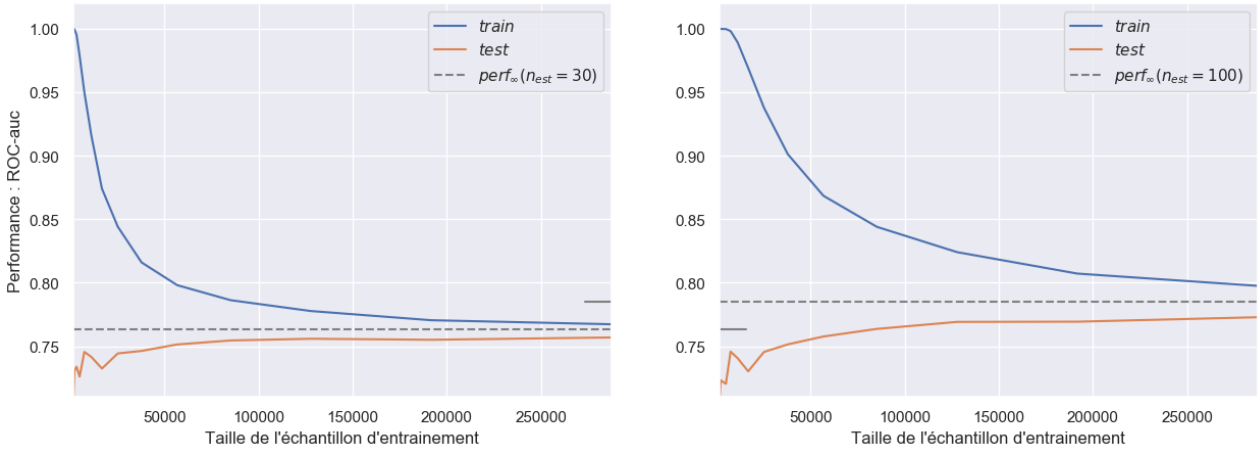

FIGURE 4 – Courbe d'apprentissage du modèle, $n_{estim} = [30, 100]$

La métrique d'évaluation du modèle est la courbe ROC et plus particulièrement l'aire sous sa courbe (AUC-ROC). Cette métrique est évaluée en cross-validation sur 5 folds dont les résultats sont repris dans le tableau suivant :

Fold	ROC auc
1	0.766
2	0.772
3	0.775
4	0.781
5	0.769
moy.	0.774
std.	0.0032

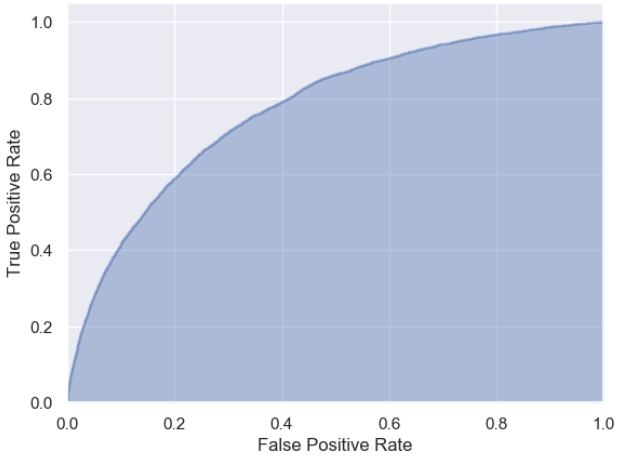

FIGURE 3 – Courbe ROC

2.3.4 Courbe d'apprentissage du modèle

La courbe d'apprentissage (*Learning Curve*) représente l'évolution de la performance d'un algorithme en fonction de son expérience (taille de l'échantillon de données). Par performance on entend ici la métrique ROC-auc, on pourrait également tracer la fonction objectif (logloss) en fonction de l'expérience.

On déduit des informations intéressantes de cette courbe :
— Puisque la courbe *train* fini par se stabiliser, on peut en déduire que son apprentissage est terminé [2]
— La performance maximale du modèle avec des données infinies et sous cette complexité sera au maximum 0.767, c'est-à-dire la valeur asymptotique de la courbe *train*.
— La performance maximale du test set peut être augmentée en augmentant la complexité du modèle, à savoir le n_{est}. Cela aura pour effet que l'apprentissage sera moins abouti (plus grand écart entre le train set et le test set).

2.4 Assignation

Le modèle entraîné prend la forme d'une surface $h(x)$ qui renvoie la probabilité de défaillance de payement pour chaque client. Le choix d'un seuil de probabilité τ au delà duquel une observation est considérée comme

2. Sous la complexité donnée, à savoir la combinaison (features + paramètres du modèle)

positive donnera alors un vecteur binaire y_{pred}.

Prenons maintenant 2 cas extrêmes :

— Seuil est à $0,0\%$: Tous les demandeurs ayant une probabilité de défaut de payement supérieure à $0,0\%$ (tous) se verront refuser le crédit. Cela engendre un manque à gagner maximal pour l'entreprise. (mais aucune défaut de payement, puisqu'elle n'a pas preté) (FP et TP nombreux, FN et $TN = 0$)

— Seuil à $100,0\%$: Tous les demandeurs ayant une probabilité de défaut inférieure à $100,0\%$ (tous) se voient accorder leur crédit. Cela engendre beaucoup de rentrées (intérêts) mais, par l'absence de contrôle, beaucoup de mauvais payeurs et donc des coûts de recouvrement importants. (FN et TN nombreux, FP et $TP = 0$)

Il existe entre ces deux points un seuil τ_{opt}, correspondant à un niveau de risque, qui maximise le profit. Ces concepts sont illustrés à la figure 2.4.

L'équation du coût total, calculé par rapport à un coût relatif de chacune des erreurs FP et FN correspond à :

$$TotalCost(\tau) = \frac{FP(\tau)}{n} * CFP + \frac{FN(\tau)}{n} * CFN$$

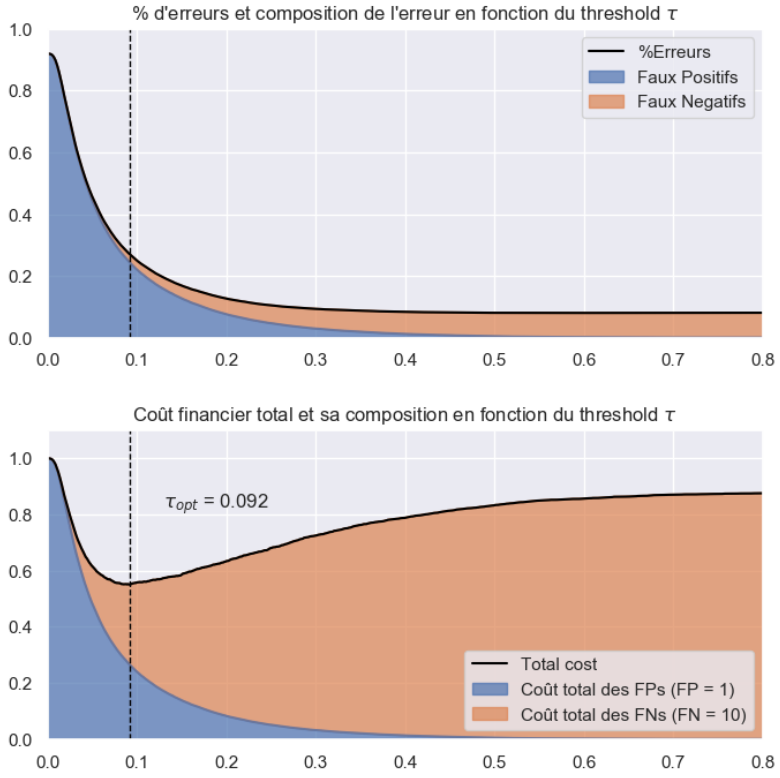

FIGURE 5 – Démonstration du threshold optimal en fonction de la performance du modèle et du rapport de coûts

Le minimum de cette équation dépend évidemment du coût de chacun des types d'erreur. Le graphe 2.4 nous montre que pour un rapport de coût de $\frac{CFN}{CFP} = \frac{1}{10}$, on accordera un crédit aux personnes présentant une probabilité de défaillance inférieure à $9,2\%$.

2.5 Inteprétabilité du modèle

L'interpretabilité du modèle est construite au travers du module SHAP (SHapley Additive exPlanations) de Lundberg et Lee [2].

Le but de SHAP est d'expliquer la prédiction de chaque instance x_i en calculant la contribution de chaque feature sur la prédiction. Le *SHAP Explanation* calcule les valeurs Shapley grâce à la théorie des jeux coopératif. Chacune des valeur des features agit comme un joueur dans une coalition. Les valeurs de Shapley nous disent comment distribuer équitablement le *Payout* (= prédiction) entre les features.

Les sections suivantes montrent deux visualisations : d'abord, celle de l'importance relative des features (dit *Summary Plot* en figure 7) et ensuite un exemple de *Force Plot* qui montre les contributions des features pour une prédiction. (figure 6)

2.5.1 Explication de la prédiction

Les flèches rouges représentent les variables qui, pour ce client, tendent à augmenter la probabilité d'impayés. A l'inverse, les flèches bleues sont les variables pour lesquelles le client à des valeurs rassurantes et qui tendent alors à abaisser cette probabilité. L'ensemble de ces compositions positives et négatives s'additionnent pour déplacer la barre de risque de la *base value* au *model output*.

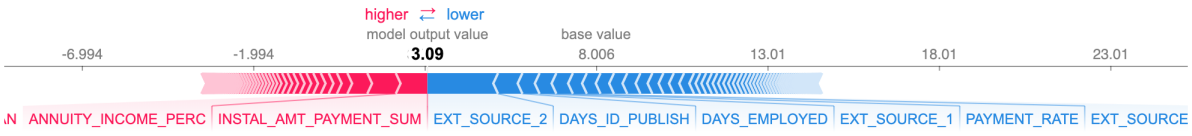

FIGURE 6 – Force Plot dans le DashBoard : probabilité d'impayé [%]

2.5.2 Importance des features

— Les features sont ordonnées du haut vers le bas, dans l'ordre d'importance décroissant.
— La couleur des points correspond à la valeur de cette variable. Ainsi, pour la variable EXT_SOURCE_3, une valeur faible (dans les bleus) contribue à augmenter sa contribution positive (augmenter la probabilité de défaut de payement)
— Pour chaque variable, on trouve une distribution qui correspond aux valeurs SHAP données sur l'échantillon d'entraînement. Par exemple, on voit que souvent la variable EXT_SOURCE_2 a une contribution négative de l'ordre de -0.01. Lorsque sa valeur est faible, on peut toutefois trouver des contribution très positive (jusque $+0.25$ sur la valeur SHAP).

3 Limites et améliorations possible

La stratégie de sélection de features par Regression Lasso répond à un besoin de performance en temps. Néanmoins, elle est limitée à plusieurs niveaux :

1. D'abord, Lasso n'accepte pas les valeurs manquantes qui sont donc imputées par la médiane/valeur la plus courante. Or, LGBM aurait pu astucieusement s'en servir pour augmenter la performance. Le fait de remplacer celle-ci fait gomme tout pattern éventuellement pré-existant. On risque donc de supprimer une variable, parce que sa variance est faible (car souvent imputée par une constante), alors que l'information de l'absence (*missingness*) pourrait servir de noeud dans LGBM.
2. Ensuite, Lasso traite chacune des variables comme indépendante. Dès lors, on peut concevoir que si deux variables sont auto-corrélées et fortement correlées à la target, on ait une répartition du poids entre ces deux variables. Cela aurait pour conséquence de diminuer virtuellement l'importance de la variable, là où Light GBM aurait poolés ces variables séparément (pool mutuellement exclusives, comme expliqué en 2.3.1).
3. De manière similaire, Lasso n'envisage pas les interactions entre les features. Ainsi, si x_1 et x_2 ne sont pas de bon prédicteurs indépendants, il n'est pas exclut que $x_1 * x_2$ soit un bon prédicteur. On risque donc de supprimer des variables dont la combinaison serait intéressante [3].

3. D'où l'intéret de réaliser un feature engineering très exhaustif, avec un maximum de combinaison, de sorte que le Lasso puisse choisir une feature $x_3 = x_1 * x_2$

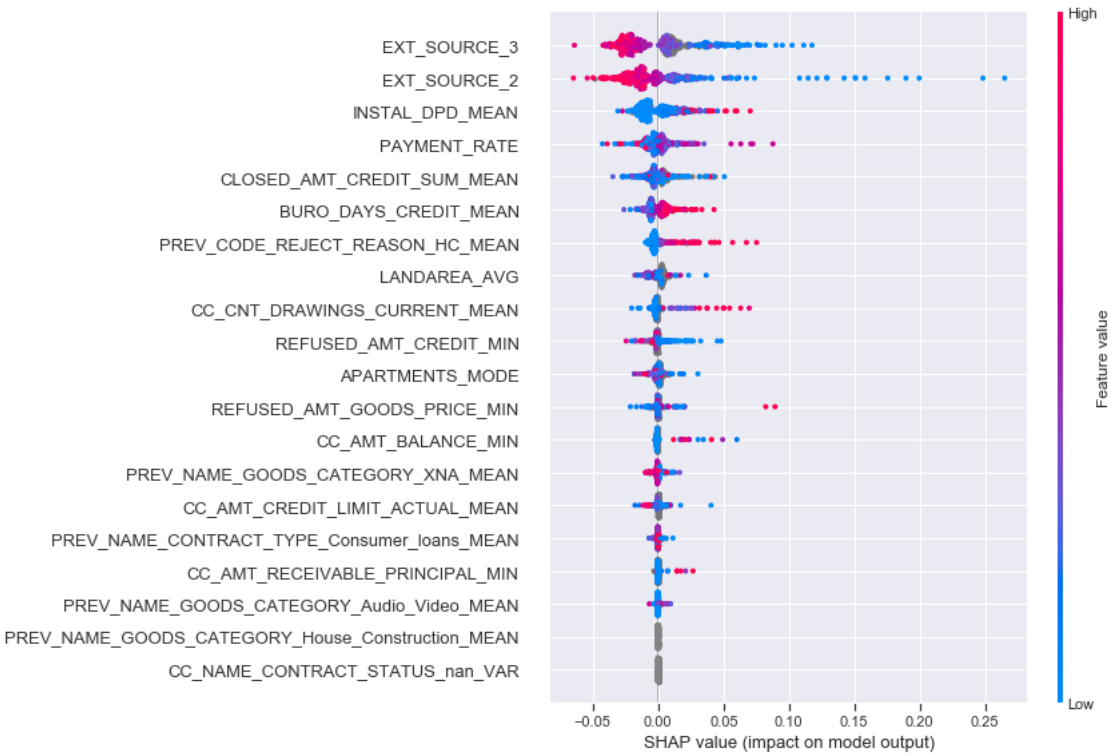

FIGURE 7 – Importance des features dans le modèle (contribution sur la sortie

Les performances sont satisfaisantes, toutefois, il est admis que ce type d'algorithme peut atteindre des niveaux de performance supérieurs [5] grâce à deux modifications principales :

1. Utilisation plus extensive des variables : sélection moins restrictive et donc capacité d'extraire davantage d'informations avec les conséquences suivantes :
   — Augmentation du temps de calcul,
   — Risque d'overfitting accru,
   — Interpretabilité réduite.
2. Recours aux techniques de *Model Stacking* [6], qui concatènent les résultats de plusieurs modèles de nature différentes[4] et réalise une macro prédiction sur base sorties des algorithmes en amont. Les conséquences sont alors :
   — L'augmentation de la performance en tirant profit des qualités locales de chacun des modèles,
   — La réduction de l'interpretabilité puisqu'on combine plusieurs modèles dont le processus décisionnel est différent.

Le dilemme complexité-performance vs. explicabilité peut être résolu par des *local surrogates models* qui modélisent une série de surfaces $\hat{h}(x)$ qui lient, de manière locale, les entrées aux sorties en faisant abstraction de la complexité du modèle . Exemple pour un modèle bivarié, étudié au point $P(x_1, x_2)$:

$$
\begin{aligned}
h(\, x_1,\, x_2\,) &= y \\
h(\, \mathbf{x_1} + \mathbf{dx_1},\, x_2\,) &= y + dy_1 \\
h(\, x_1,\, \mathbf{x_2} + \mathbf{dx_2}\,) &= y + dy_2
\end{aligned}
$$

Si $dy_2/dx_2 > dy_1/dx_1$ cela implique qu'une variation infinitésimale de x_2 à une conséquence plus importante que la même variation sur x_1. Par conséquent, cette feature à une importance supérieure (au point P, d'où l'adjectif "local"). Cela permet de faire abstraction du stacking.

4. Par exemple : une régression logistique, un XGBoost et un LGBM

Références

[1] LightGBM : A Highly Efficient Gradient Boosting Decision Tree, https://papers.nips.cc/paper/6907-lightgbm-a-highly-efficient-gradient-boosting-decision-tree.pdf

[2] Lundberg, Scott M and Lee, Su-In : A Unified Approach to Interpreting Model Predictions, 2017. http://papers.nips.cc/paper/7062-a-unified-approach-to-interpreting-model-predictions.pdf

[3] Théorie des jeux coopératifs : https://fr.wikipedia.org/wiki/Jeu_coop%C3%A9ratif_(th%C3%A9orie)

[4] Principes de sélection de variables : https://en.wikipedia.org/wiki/Feature_selection#Main_principles

[5] Problème similaire de prédiction (binaire), avec score relatifs : https://www.kaggle.com/c/home-credit-default-risk/leaderboard

[6] Principe et avantages du Stacking : https://blogs.sas.com/content/subconsciousmusings/2017/05/18/stacked-ensemble-models-win-data-science-competitions/

[7] Lasso Regression : https://www.thelearningmachine.ai/lle

[8] Information Gain in Decision Trees, https://en.wikipedia.org/wiki/Information_gain_in_decision_trees

A Modèle

A.1 Paramétrage du modèle

paramètres	valeur	paramètre	valeur
boosting type	GBDT	num leaves	15
objective	binary	min child samples	20
max depth	-1	reg alpha	0
n estimators	100	reg lambda	0
metric	auc	subsample	0.9
colsample bytree	0.9	categorical features	True