Concevoir et déployer un modèle de *credit scoring*

Problématique

Le projet consiste à mettre en œuvre un outil de *credit scoring* pour calculer la probabilité qu'un client puisse rembourser le crédit qu'il demande et aider la société financière et les assurances à prendre leur décision lors du traitement de la demande de prêt. De plus, ce projet doit aussi répondre à une demande grandissante de transparence de la part des clients vis-à-vis des décisions d'octroi de crédit.

Objectifs

1. Cet outil doit permettre de définir la probabilité de défaut de remboursement d'un crédit sur la base d'informations relatives au client.

2. Il doit également offrir un certain niveau de transparence et de simplicité concernant les données et leurs traitements en vue d'implémenter des méthodes d'interprétabilité du modèle sous la forme d'un dashboard interactif à destination des chargés de relation client ainsi que leurs clients.

3. Une simulation du data drift est aussi réalisée en vue de déterminer une période de maintenance pour conserver des performances de prédictions optimales.

Compétences acquises

- Sélectionner et adapter un **kernel Kaggle**.
- Gérer le **déséquilibre entre 2 classes** pour l'entraînement des modèles.
- Élaborer la structure d'un **pipeline** capable de répondre au besoin métier et **optimiser les hyperparamètres** des différents algorithmes qui le composent.
- **Évaluer les modèles de classification** avec des indicateurs techniques et métier.
- **Évaluer l'importance relative de chaque feature** dans le résultat de prédiction du modèle **(SHAP)**.
- **Simuler le data drift** dans le temps **(Evidently)** et proposer une période d'entretien du modèle dans le but de maintenir ses performances.
- **Déployer le modèle en back-end dans le Web via une API Flask**.
- **Concevoir et déployer une interface graphique (un dashboard streamlit) interactive** en front end pour permettre aux chargés de relation client d'envoyer les requêtes au modèle et d'interpréter les résultats de prédiction.
- Réaliser des **tests unitaires**.
- Utiliser un **logiciel de gestion de versions (Git)** pour assurer l'intégration du modèle.

Table des matières

I) Données

1) Structure et caractéristiques

Le jeu de données est constitué de toute sorte d'informations relatives aux clients comme leurs crédits en cours, leurs genres ou encore leurs dates de naissances. Il se découpe en de multiples sous-datasets qu'il va falloir regrouper et traiter. Pour ce faire le kernel Kaggle revu de jsaguar a été utilisé comme fondation à ce travail pour sa clarté et son efficacité apparente.

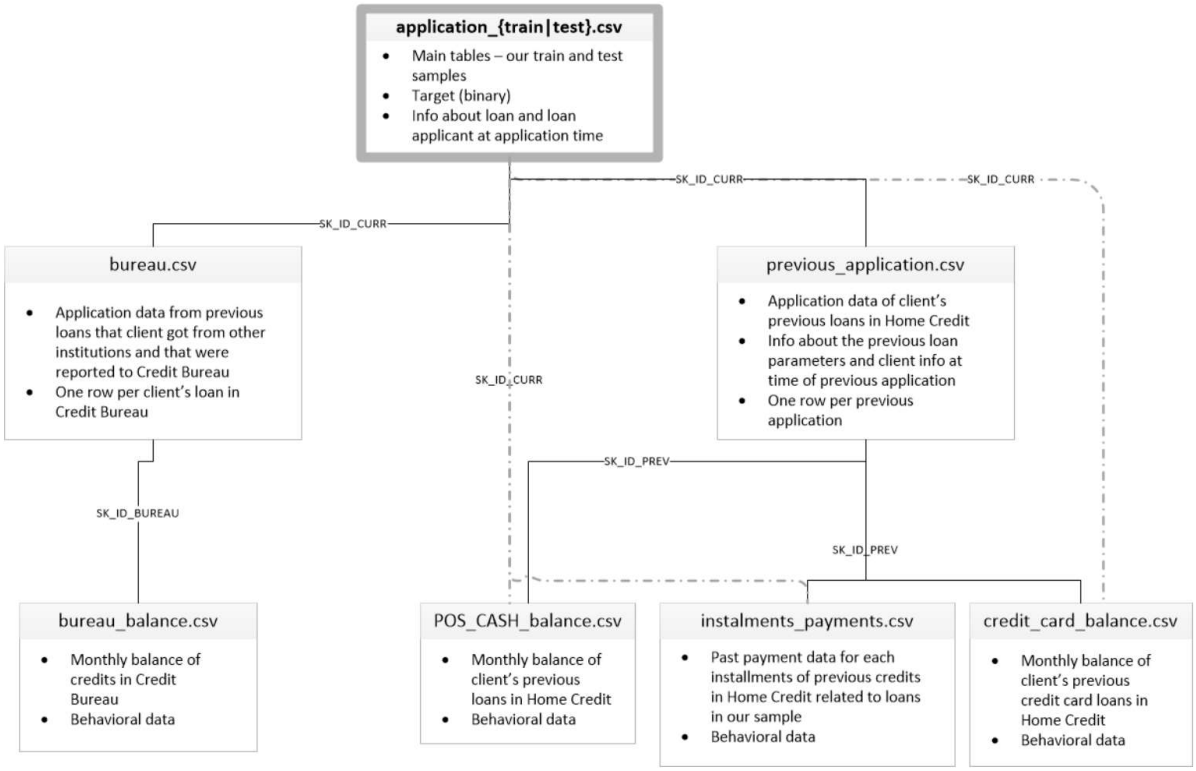


Figure 1 : *Schéma des différents datasets qui constituent le jeu de données.*

Une fois tous les sous-datasets agrégés, en un dataset global avec des features supplémentaires ajoutées par le kernel de jsaguar (ramenant le nombre de features à un total de 622), une dizaine de features se retrouvent complètement vides et à peu près 40 % des features sont à moins de 50 % remplies.
Le nombre de features étant important seules les features remplies à plus de 75 % sont conservées ramenant leur nombre à 360. Ceci permet à la fois de réduire le temps de traitement des données par le modèle ainsi que de limiter le biais induit par l'imputation des valeurs manquantes restantes.

⇨ On obtient ainsi de meilleures performances :
- Un temps de traitement réduit.
- Une certaine fiabilité entre les données et les clients physiques qu'elles représentent.

Une partie du dataset concerne des crédits échus et, par conséquent, comprend des étiquettes cibles (une classe binaire attribuée à chaque dossier client pour indiquer si le crédit a été remboursé ou non) qui pourront être utilisées pour entraîner le modèle de classification.
Toutefois, une autre partie du jeu de données, moins conséquente, ne contient pas d'étiquette. Il s'agit des dossiers en cours pour lesquels il faut réaliser un classement prédictif avec le modèle entraîné.

2) Gestion du déséquilibre des classes

Parmi les données correspondantes au dataset d'entraînement, on peut remarquer que le taux de crédits remboursés est un peu plus de 10 fois supérieur au taux de défauts de remboursement.

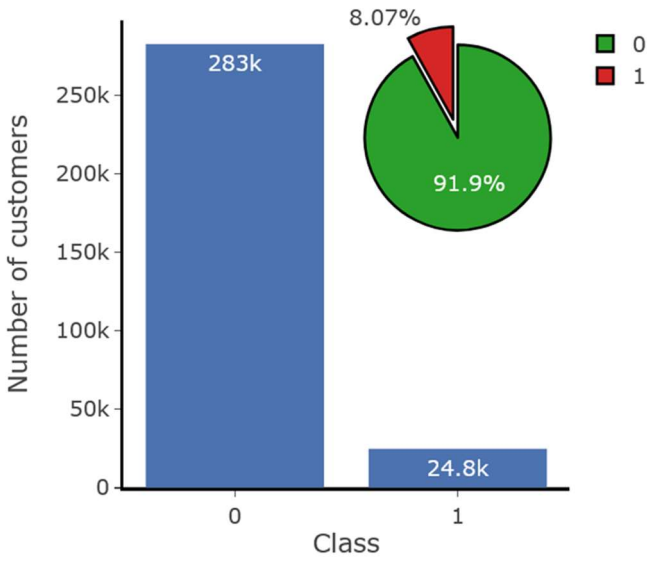


***Figure 2** : Déséquilibre entre le nombre de clients qui ont remboursé leurs prêts (classés 0) et ceux qui sont en défaut (classés 1).*

Deux approches sont possibles pour traiter ce déséquilibre lors de l'entraînement du modèle :

- *Data-level :* L'utilisation d'une méthode de rééchantillonnage *(Resampling)* sur le jeu de données d'entraînement, qui contrairement à l'attribution de poids, va générer des données fictives (ou en retirer des réelles) à l'une ou l'autre des classes pour les rééquilibrer. Cette étape supplémentaire modifie le dataset et allonge le temps de traitement, particulièrement dans le cas du suréchantillonnage (*Oversampling* : ajout d'échantillons fictifs basés sur ceux déjà existant dans la classe minoritaire). Dans le cas du sous-échantillonnage (*Undersampling* : retrait de données de la classe majoritaire), les données retirées peuvent faciliter l'entraînement du modèle en définissant mieux les nuages de points représentant les prédictions possibles par la suppression des outliers ou des chevauchements mais elles peuvent aussi exacerber le phénomène de surentraînement *(overfitting)* du modèle.

- *Algorithm-level :* Cette approche repose sur des adaptations des modèles de machine learning classiques afin qu'ils soient en mesure de mieux gérer le déséquilibre comme : l'attribution d'un poids qui donnera davantage d'importance aux clients en défaut de remboursement, moins nombreux. L'avantage de cette méthode est la rapidité de traitement pour le modèle qui n'a pas à réaliser une étape de rééchantillonnage et un jeu de données inchangé, en conséquence, potentiellement plus fiable.

a) La méthode *data-level*

i. Rééchantillonnage aléatoire

Les méthodes de rééchantillonnage sont multiples. La technique la plus simple est le rééchantillonnage aléatoire. Elle consiste soit à retirer des points, de façon aléatoire, des classes majoritaires dans le cas d'un sous-échantillonnage (*undersampling*) ou à en ajouter des fictifs, de façon aléatoire, dans le cas d'un suréchantillonnage (*oversampling*). Bien que cette technique puisse se montrer efficace, son utilisation peut conduire à la perte d'informations importantes, en particulier si le dataset est petit, induite par sa nature aléatoire. Toutefois, il existe d'autres algorithmes de rééchantillonnage plus sophistiqués et performants.

ii. Sous-échantillonnage

- **Centroïdes des clusters (*Cluster centroids*)**

Cet algorithme identifie les clusters de la classe majoritaire et les remplace par les centroïdes leurs centroïdes. De cette façon, cette technique préserve la distribution originale de la classe majoritaire tout en réduisant sa taille.

- **Tomek's links**

Tomek's links est une approche plus fine de sous-échantillonnage de la classe majoritaire et choisit de manière plus intelligente les points à supprimer. Cet algorithme cherche à apparier les points de la classe majoritaire les plus proches de ceux de la classe minoritaire. Ces paires de points sont les *Tomek's links*. Une fois la paire générée, le point de la classe majoritaire, qui en fait partie, se voit retirer. Ainsi, ce sont les points de la classe majoritaire les plus proches de ceux de la classe minoritaire qui sont sélectionnés en priorité pour être supprimés. Cette méthode permet de mieux isoler les points appartenant à chacune des 2 classes et de réduire la variance de la classe majoritaire en supprimant des outliers éventuels. In fine, les 2 classes sont mieux définies l'une par rapport l'autre facilitant leurs distinctions par le modèle.
NB : *À noter que l'identification des Tomek's Links est réalisée de façon très similaire à l'identification des plus proches voisins par l'algorithme K-NN.*

- **Near Miss**

Il s'agit d'une technique simple mais efficace qui réduit le nombre d'échantillons de la classe majoritaire tout en conservant les échantillons les plus pertinents. Cet algorithme sélectionne les points de la classe majoritaire à éliminer selon leurs proximités avec ceux de la classe minoritaire pour ne conserver qu'un sous-ensemble d'entre eux. Cependant, il diverge dans sa manière de calculer les proximités entre les points des 2 classes et propose 3 méthodes différentes :

- **NearMiss-1 :** sélectionne le point de la classe majoritaire dont la distance moyenne, qui le sépare des 3 points de la classe minoritaire les plus proches, est la plus réduite.
- **NearMiss-2 :** sélectionne le point de la classe majoritaire dont la distance moyenne, qui le sépare des 3 points de la classe minoritaire les plus éloignés, est la plus courte.
- **NearMiss-3 :** est un hybride des deux premières variantes et combine leurs points forts.

Ainsi, cette technique permet de préserver les informations de la classe majoritaire tout en réduisant sa taille. L'un des principaux avantages de Near Miss est sa facilité de mise en œuvre et son efficacité en termes de calcul.

iii. Suréchantillonnage

- **SMOTE & ADASYN**

Comme on a pu le voir dans le ***.i***, plutôt que de réduire la classe majoritaire, il est possible d'augmenter la classe minoritaire par suréchantillonnage.
Les méthodes de suréchantillonnage les plus utilisées, de nos jours, sont SMOTE (*Synthetic Minority Oversampling TEchnic* ou Suréchantillonnage Minoritaire Synthétique) et ADASYN (*ADAptive SYNthetic Sampling* ou Échantillonnage Synthétique Adaptatif) qui est une version améliorée de SMOTE. Il génère plus d'échantillons synthétiques dans les régions de l'espace des features où la densité des observations minoritaires est faible et moins d'échantillons dans les régions où la densité est plus élevée. Ces 2 algorithmes utilisent les points déjà présents dans la classe minoritaire pour générer

artificiellement de nouveaux points (dit synthétiques). Ainsi, bien que les points générés soient fictifs, ils demeurent crédibles. Toutefois, un inconvénient du suréchantillonnage et la possible création d'exemples synthétiques ambigus s'il existe un fort chevauchement entre les classes puisque le modèle de suréchantillonnage ne tient pas compte de la classe majoritaire.

iv. Choix pour les tests dans le projet

Lors des tests de différents classifieurs, couplés à différentes techniques de gestion du déséquilibre des classes, c'est **SMOTE-NC** qui a été sélectionnée pour suréchantillonner la classe minoritaire à un ratio de 1 : 1 avec la classe majoritaire. Comme décrit précédemment, cette méthode se base sur la proximité des points représentant les échantillons (les clients dans notre cas) appartenant aux classes à suréchantillonner dans le but d'en créer de nouveaux crédibles bien que fictifs.
Quant à l'algorithme de sous-échantillonnage, c'est **Near Miss (variante 3)** proposé par la bibliothèque **imbalanced-learn** qui a été sélectionnée.
NB : *Dans un projet ultérieur qui chercherait à optimiser au mieux les performances du modèle mis en place dans ce projet-ci, il pourrait être intéressant de tester les performances des classifieurs après combinaison d'une étape de suréchantillonnage et d'une étape de sous-échantillonnage réalisées sur les 2 classes.*

b) La méthode *algorithm-level*

i. Apprentissage sensible aux coûts (*Cost-sensitive*)

L'une des méthodes que propose l'approche *algorithm-level* consiste à affecter un poids plus important à la classe minoritaire : l'apprentissage sensible aux coûts (*Cost-sensitive*). En pratique, il est spécifié au modèle que de bien classer un point de la classe minoritaire est plus important que de bien classer un point de la classe majoritaire. De cette manière, une erreur de classification d'un point appartenant à la classe minoritaire est considérée par le modèle comme étant plus grave qu'une erreur de classification d'un point appartenant à la classe majoritaire. Les modèles de machine learning cherchant à optimiser un paramètre prédéfini en optimisant le gain, amène l'algorithme à donner plus d'importance à la classe minoritaire et améliorer ses prédictions, en conséquence. Par défaut, il est judicieux de choisir les poids (coefficients) de départ proportionnellement à l'écart relatif entre la classe majoritaire et les classes minoritaires (Dans notre cas l'écart relatif entre nos 2 classes est d'à peu près 1000 %. En conséquence, une erreur de classification sur la classe minoritaire devrait avoir 10 fois plus d'importance).

ii. Apprentissage à une classe (*One-class classification*)

Dans certains cas, une autre méthode peut se montrer particulièrement efficace : l'apprentissage à une classe (*One-class classification*). Au lieu d'entraîner un modèle sur les 2 classes, il est entraîné sur la classe majoritaire. De cette façon, il devient capable de prédire avec précision si un point fait bien partie de la classe ou non. Bien que les modèles d'apprentissage à une classe ne soient pas conçus pour compenser le déséquilibre des classes, ils peuvent apporter des gains de performances significatifs.

> ⇨ In fine, la méthode de gestion du déséquilibre des données sélectionnée est basée sur l'attribution de poids différents entre les 2 classes car plus simple à mettre en place, plus rapide, sans étape d'altération du jeu de données et avec des performances supérieures à celles obtenues par le rééchantillonnage des classes.

II) Modélisation

1) Méthode de classification

a) Principe

Rappelons que la probabilité de remboursement d'un crédit est la donnée déterminante pour accepter ou non son attribution à un client qui en fait la demande.

En appliquant un seuil à cette valeur, il est possible de classer la demande du client de façon négative ou positive en lui attribuant une valeur binaire (0 ou 1) selon la probabilité de remboursement qui lui a été prédite par rapport à ce seuil.

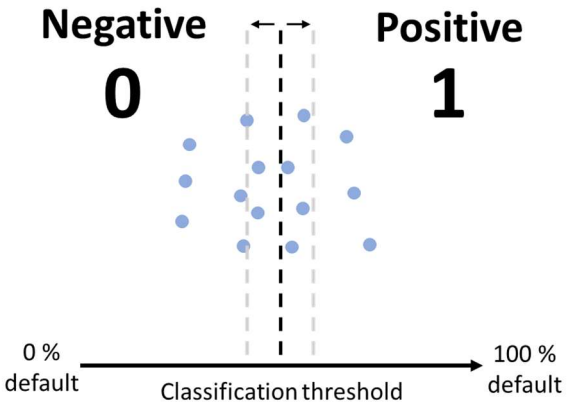


Figure 3 : *Principe de la classification binaire selon un seuil prédéfini. Les points représentent la probabilité de défaut des clients.*

Si la probabilité est inférieure, on considère que le crédit sera remboursé et le dossier est classé négatif (0). Inversement, si la probabilité est supérieure au seuil, on considère que le crédit ne sera pas remboursé, le dossier est positif (1).

⇨ Il s'agit donc d'un problème de classification binaire dont le résultat dépend du seuil fixé.

b) Mesures

Pour évaluer le modèle on peut comparer les valeurs prédites avec les valeurs réelles et mettre en place un tableau indiquant le nombre de valeurs correctement prédites ou non (la matrice de confusion).

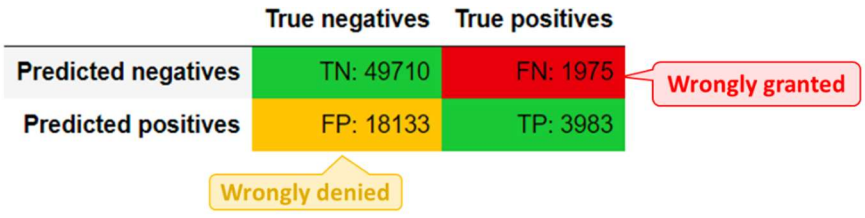


Figure 4 : *Matrice de confusion montrant le nombre de clients qui ont été bien ou mal classés par le modèle avec :*

- *Vrai positif (« True positive » ou TP) : Prédiction positive qui s'avère juste.*
- *Vrai négatif (« True negative » ou TN) : Prédiction négative qui s'avère juste.*
- *Faux positif (« False positive » ou FP) : Prédiction positive qui se révèle fausse.*
- *Faux négatif (« False negative » ou FN) : Prédiction négative qui se révèle fausse.*

⇨ Ainsi, notre tableau d'évaluation montre le nombre de valeurs binaires (0 ou 1) bien ou mal classées selon un seuil de classification prédéfini.

⇨ Pour un modèle de classification optimal, il est donc nécessaire de bien déterminer ce seuil par rapport aux prédictions du modèle et, indirectement, des données relatives aux clients.

c) Indicateurs techniques de performance

Les valeurs de ce tableau peuvent être converties en indicateurs qui caractérisent le modèle :

- La **sensibilité ou rappel** (= TP / (TP + FN)) : représente la proportion de positifs réels qui sont bien détectés par le modèle, autrement dit que la probabilité que la prédiction soit positive lorsqu'elle devrait l'être. Dans notre cas, cet indicateur représente la capacité du modèle à détecter les dossiers de crédits non remboursés (codés 1).

  ⇨ Plus le rappel est fort, plus les cas positifs réels ont bien été détectés et moins on observe d'erreurs de types II (faux négatifs) (en général plus graves que les erreurs de type I).

- La **spécificité** (= TN / (TN + FP)) : représente la proportion de négatifs réels qui sont bien détectés par le modèle, autrement dit que la probabilité de la prédiction soit négative lorsqu'elle devrait l'être. Dans notre cas, elle représente la capacité du modèle à détecter les dossiers de crédits remboursés (codés 0).

  ⇨ Plus la spécificité est élevée, plus les cas négatifs réels sont bien détectés et moins on observe d'erreurs de types I (faux positifs).

- La **précision** (= TP / (TP + FP)) : représente la proportion de vrais positifs sur l'ensemble des positifs détectés par le modèle. Dans notre cas, elle représente la capacité du modèle à détecter les vrais dossiers non remboursés (codés 1).

  ⇨ Plus la précision est forte, moins on observe de faux positifs dans les prédictions du modèle et moins il y a d'erreurs de types I.

NB : *La spécificité est complémentaire de la sensibilité. En termes de mesures, la première devrait être faite seulement sur les vrais négatifs et la seconde seulement sur les vrais positifs. Ainsi, Ensemble, la sensibilité et la spécificité d'une prédiction ou d'un test donnent une appréciation de sa validité intrinsèque. Prises séparément, elles ne veulent rien dire. Par exemple, un test avec une sensibilité de 95 % n'a aucune valeur si sa spécificité n'est que de 5 %. En effet, si la somme de la sensibilité et de la spécificité est égale à 100 % les prédictions ne sont aucunement corrélées aux classes à prédire (dans notre cas au remboursement des crédits). Le modèle ne fonctionnerait pas du tout.*

⇨ Il est en général nécessaire de trouver le meilleur équilibre entre ces deux indicateurs pour minimiser le nombre de mauvaises prédictions et maximiser les performances du modèle.

Il est possible de représenter les évolutions de la sensibilité et de la spécificité en fonction du seuil de classification (la fonction d'efficacité du récepteur ou *Receiver Operating Characteristic* abréviée courbe ROC).

d) AUROC

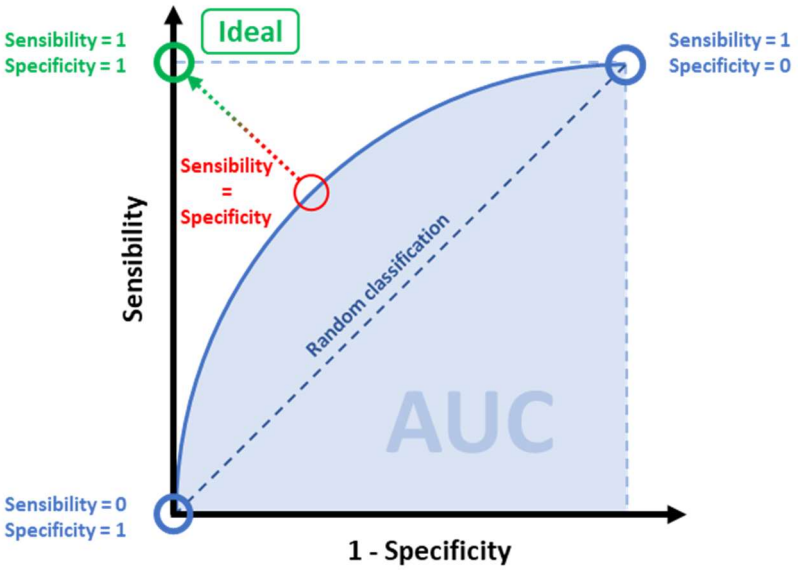


***Figure 5** : Principe d'une courbe ROC et du calcul de son aire (AUC) associée.*

Une courbe ROC caractérise le classifieur qui a produit les résultats sous forme de probabilités par variation du seuil de classification.

Un modèle idéal a une sensibilité et une spécificité de 1. Autrement dit, meilleurs sont les indicateurs, plus la courbe se rapproche de cet idéal (indiqué en vert) et plus le modèle est performant.
Il est possible de mesurer cette performance par l'aire sous la courbe qui prend l'acronyme anglais AUC pour *Area Under the Curve.*

⇨ C'est pourquoi, la mesure AUROC est utilisée pour évaluer des modèles de classification et représente à ce titre un critère de sélection pour nos modèles.

e) Importance relative entre les erreurs de type I et II

L'indicateur précédent, bien que pertinent dans l'hypothèse où les éléments de la matrice de confusion sont de même importance, n'est pas toujours suffisant pour mesurer la performance d'un modèle dans le cas d'un déséquilibre des classes important.

Si la demande d'un client est refusée à tort, à cause d'un risque de défaut de paiement jugé trop élevé, la société de crédit perd ce client et donc le bénéfice potentiel qui lui est associé. En revanche, si elle accorde un crédit à un client qui finalement s'avère ne pas le rembourser alors, la perte est beaucoup plus importante puisqu'elle correspond au total du prêt qui reste à rembourser.

⇨ Il s'agit donc, de trouver le meilleur compromis entre le nombre de crédits accordés à tort, coûteux pour l'entreprise (les faux négatifs aussi appelés erreurs de type II) et le nombre de crédit refusés à tort sur des clients solvables (les faux positifs aussi appelés erreurs de type I) qui représentent un manque à gagner certain.

⇨ Il existe un indicateur approprié pour dans ce genre cas : La mesure F_β.

f) Le score F_β

Le score F_β permet d'associer le rappel et la précision en un seul indicateur tout en permettant d'accorder une importance différente entre les erreurs de type I et les erreurs de type II en réglant la valeur du β.
Il s'exprime tel que :

$$F\beta = (1 + \beta^2) * \frac{precision * recall}{(\beta^2 * precision) + recall}$$

Avec :

- β < 1 : plus d'importance est donnée à la précision (autrement dit aux faux positifs ou erreurs de type I).
  Ex : β = 0.5 signifie que les FP ont 2 fois plus d'importance que les FN.
- β = 1 : autant d'importance est attribuée à la précision et au rappel.
- β > 1 : plus d'importance est donnée au rappel (autrement dit aux faux négatifs ou erreurs de type II).
  Ex : β = 2 signifie que les FN ont 2 fois plus d'importance que les FP.

NB : *Bien qu'il soit possible de choisir la valeur de β, généralement ce sont les valeurs 0.5, 1 et 2 qui sont les plus utilisées.*

Bien que cet indicateur existe, il est souvent plus intéressant de créer un score métier spécifique à la problématique à résoudre.

g) Fonction de coût métier

Il est ainsi possible de définir une fonction de coût en accordant des poids différents aux éléments de la matrice de confusion lors de l'entraînement du modèle.

Un poids de -1 est affecté à chaque faux positif (FP) et un poids de -10 aux dossiers identifiés comme faux négatifs (FN) car estimés 10 fois plus néfastes pour l'accroissement des bénéfices de l'entreprise. Quant aux dossiers identifiés comme réellement négatifs (TN) ou réellement positifs (TP), ils sont considérés comme normaux et voient donc leurs poids associés rester nuls.
NB : *Ces coefficients se basent sur des hypothèses à ajuster avec des interlocuteurs métier. En dehors d'ajuster les coefficients d'importance de FP et FN, il pourrait aussi être intéressant de comptabiliser l'influence des vrais positifs et négatifs en quantifiant le coût que représente chaque ouverture de dossier par exemple.*

Le gain de classification est alors défini par la fonction suivante :

Gain = 0 x TP + 0 x TN - 1 x FP - 10 x FN

Ainsi, l'optimisation métier du classifieur consiste à maximiser le gain (normalisé pour la circonstance) en faisant varier le paramètre seuil de classification mais également les paramètres du classifieur.

Cette opération est réalisée lors de l'optimisation des hyperparamètres avec la méthode Hyperopt par l'intermédiaire d'un dictionnaire de paramètres (dont chacun possède une gamme de valeurs à combiner) passés à la fonction objective avec pour métrique d'évaluation, le gain normalisé à maximiser.

⇨ L'optimisation métier résulte en un Gain normalisé de 0,71 et une valeur de F2 de 0,44.

2) Modèle

a) Contraintes de construction du modèle

Le modèle doit :

- Être capable de gérer des données déséquilibrées.
- Être capable de gérer des données catégorielles encodées de façon ordinales.
- Être peu sensible aux éventuels outliers (falcultatif).
  NB : *Pas nécessaire s'ils ont tous déjà été gérés au cours du feature engineering.*
- Posséder un classifieur qui intègre des méthodes de classification binaires et qui est capable de traiter un nombre important de données dans un temps d'exécution raisonnable.
- Ne pas être trop complexe pour qu'il reste interprétable. Notamment, dans le but d'être transparent avec les clients et permettre aux chargés de relation client d'interpréter les prédictions faites par le modèle.

b) Classifieurs testés

Comme nous avons pu le voir, le classifieur de notre modèle doit être capable de répondre à un certain nombre de critères. Ces contraintes ont amené à la sélection suivante :

- Un dummy classifier qui ne cherche pas de schéma entre les données pour faire office de ligne de base de comparaison.
- Une régression logistique pour représenter un classifieur simple.
- Les classifieurs ensemblistes les plus connus, capables de gérer les contraintes liées à nos données : la forêt d'arbres de décision aléatoire (« random forest »), le XGBoost, et le LightGBM. Leurs robustesses et leurs mécanismes de fonctionnement simple à comprendre en font des classifieurs tout désignés pour établir une première itération de notre *credit scoring*.

c) Validation et test du modèle

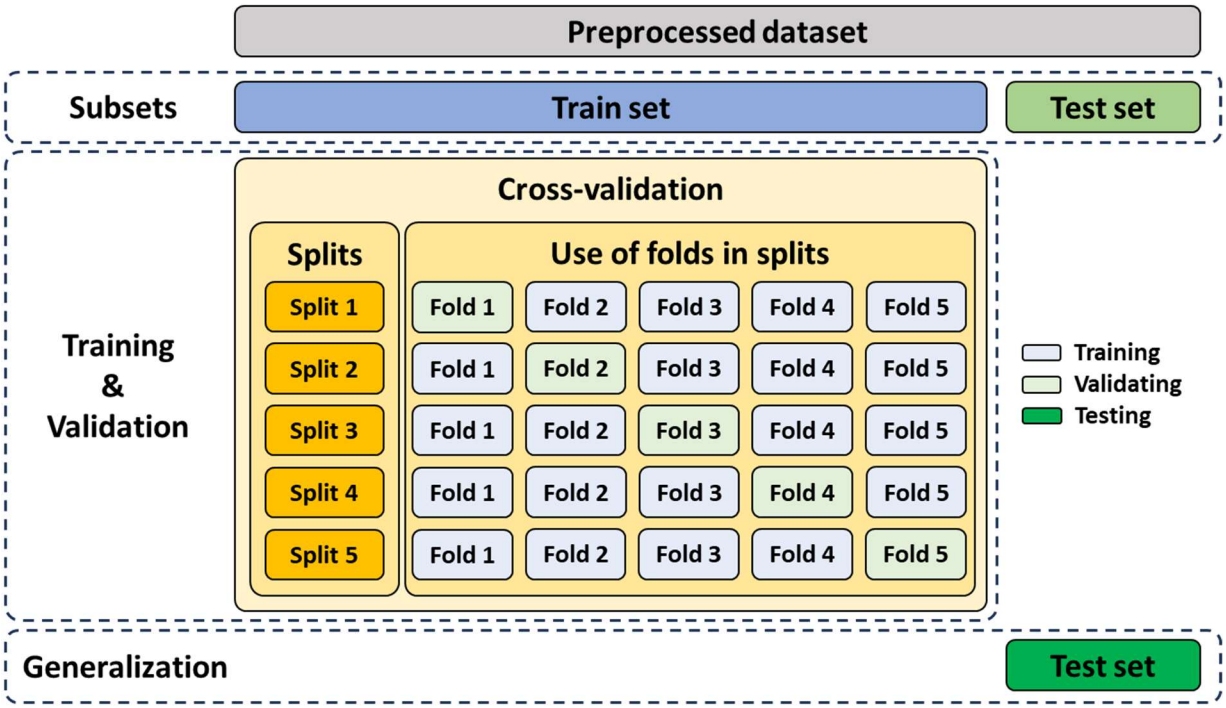


Figure 8 : *Représentation schématique des différentes étapes d'entraînement et de validation du modèle.*

Le jeu de données obtenu après le feature engineering (décrit au début de ce document) est séparé en 2 sous-datasets stratifiés (les 2 classes sont en proportion équivalentes dans chaque pli) :

- Un *train set* : Jeu de données d'entraînement avec 80 % des données.
- Un *test set* : Jeu de données de test avec les 20 % restantes.

Que ce soit lors de l'optimisation des hyperparamètres ou lors de l'étape de validation, les prédictions sont validées par validation croisée (*cross-validation*) sur 5 plis (*folds*) stratifiés déclinés en 5 *splits* créés à partir du *train set* et successivement testés. Seul le test de généralisation sur le *test set* n'est pas cross-validé.

d) Optimisation des hyperparamètres

Deux techniques d'optimisation ont été mises en place :

1. Une première basée sur une optimisation aléatoire au moyen de la fonction RandomizedSearchCV() de scikit-learn pour comparer les différents modèles testés entre eux. De cette façon, on obtient rapidement une première combinaison d'hyperparamètres pour lesquels chaque modèle peut afficher un score proche de l'optimal.

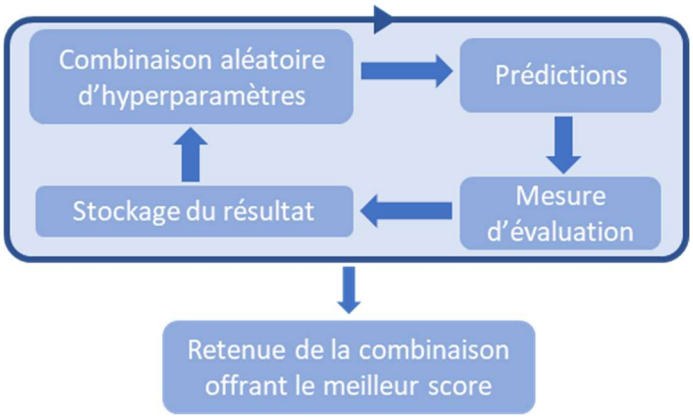


Figure 6 : *Principe de fonctionnement d'un algorithme d'optimisation aléatoire d'hyperparamètres.*

2. Une fois un modèle sélectionné, il passe une seconde étape d'optimisation pour cette fois-ci essayer d'atteindre la meilleure combinaison d'hyperparamètres possible. Cependant, l'optimisation des hyperparamètres n'est plus aléatoire. L'algorithme tient compte des résultats obtenus avec la combinaison d'hyperparamètres précédentes pour définir la nouvelle combinaison à essayer.

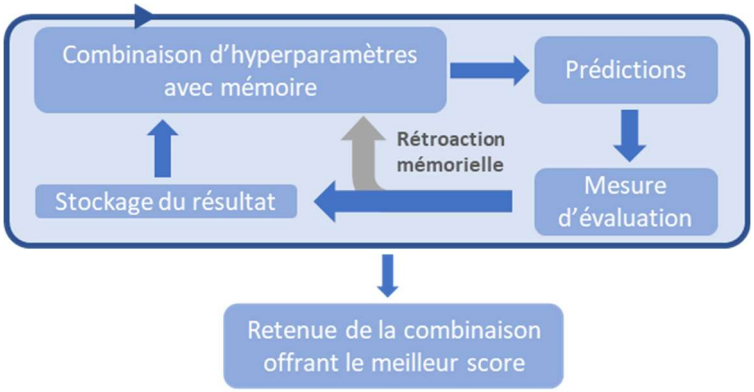


Figure 7 : *Principe de fonctionnement d'un algorithme d'optimisation bayesien d'hyperparamètres.*

⇨ Ainsi, pour cette étape, la fonction d'optimisation aléatoire de scikit-learn est remplacée par la fonction d'optimisation bayesienne d'Hyperopt.

e) Synthèse des résultats

i. Glossaire des abréviations

- **cv :** Cross-validation.
- **clf :** Classifier.
- **scl :** Data scaled (Données normalisées).
- **rsp :** Data resampled (Données Rééchantionnées).
- **wt :** Weight (Poids différents attribués aux 2 classes).
- **opt :** Optimized.
- **osp :** Oversampled.
- **usp :** Undersampled.

ii. Résultats

Tableau 1 : *Résultats obtenus par cross-validation sur le jeu de données d'entraînement.*

Models	Job score	F2 score	AUROC score	Accuracy	Processing time (s)
No class imbalance management + No scaling					
dummy_clf	0.528	0.080	0.500	0.854	0.5
logi_reg_clf_opt	0.611	0.347	0.651	0.564	14.9
rf_clf_opt	0.653	0.373	0.701	0.698	6318
xgb_clf_opt	0.654	0.374	0.701	0.698	6390
lgbm_clf_opt	0.687	0.414	0.746	0.716	8.2
No class imbalance management + Scaling					
scl_dummy_clf	0.528	0.080	0.501	0.854	2.1
scl_logi_reg_clf_opt	0.702	0.434	0.769	0.689	14.7
scl_rf_clf_opt	0.654	0.374	0.703	0.700	6061
scl_xgb_clf_opt	0.664	0.391	0.722	0.668	27.2
scl_lgbm_clf_opt	0.686	0.416	0.747	0.679	9.2
Cost-sensitive + Scaling					
scl_wt_dummy_clf	0.528	0.080	0.501	0.854	2.3
scl_wt_logi_reg_clf_opt	0.703	0.435	0.769	0.709	16.0
scl_wt_rf_clf_opt	0.644	0.377	0.696	0.597	5.7
scl_wt_xgb_clf_opt	0.668	0.394	0.723	0.688	5.1
scl_wt_lgbm_clf_opt	0.706	0.438	0.770	0.682	10.3
Oversampling + Scaling					
scl_osp_dummy_clf	0.532	0.000	0.498	0.919	737
scl_osp_logi_reg_clf_opt	0.672	0.395	0.725	0.721	741
scl_osp_rf_clf_opt	0.663	0.390	0.717	0.662	993
scl_osp_xgb_clf_opt	0.668	0.397	0.722	0.654	735
scl_osp_lgbm_clf_opt	0.647	0.380	0.699	0.595	703
Undersampling + Scaling					
scl_usp_dummy_clf	0.532	0.000	0.498	0.919	7.8
scl_usp_logi_reg_clf_opt	0.652	0.380	0.706	0.641	11.3
scl_usp_rf_clf_opt	0.643	0.369	0.684	0.645	14.9
scl_usp_xgb_clf_opt	0.663	0.387	0.714	0.690	8.6
scl_usp_lgbm_clf_opt	0.661	0.388	0.713	0.656	8.6
Hyperparameter fine tuning on the selected model : Cost-sensitive + Scaling					
scl_wt_lgbm_clf_fine_opt	0.708	0.441	0.769	0.717	10.3

Bien que les scores techniques et métier montrent les mêmes tendances, on peut noter quelques particularités :

➢ **Observations**

- Sans aucune transformation des données, les modèles ensemblistes sont plus performants malgré leur lenteur en fonction de leurs combinaisons d'hyperparamètres (à l'exception de LightGBM).

- La mise à l'échelle des données n'influence pas beaucoup le score obtenu pour les modèles ensemblistes mais améliore grandement les performances du classifieur de régression logistique.

- Parmi les différentes techniques de gestion du déséquilibre des données testées, il semble que la technique de gestion du déséquilibre par attribution de poids soit la plus performante avec LightGBM (les scores F2 et métier montrent les meilleurs résultats). Cette technique présente également l'avantage de ne pas nécessiter d'étapes supplémentaires pour transformer les données. Par conséquent, contrairement au suréchantillonnage, l'entraînement est plus rapide et aucune donnée artificielle, potentiellement non représentatives de la réalité, n'est présente. De même, comme nous l'avons vu précédemment, le sous-échantillonnage implique de réduire le nombre d'entrées des classes majoritaires pour les ajuster à la classe la plus minoritaire, réduisant la taille du dataset d'entraînement et augmentant les risques de surentraînement du modèle en conséquence. L'attribution de poids différents aux différentes classes évite également ce problème.

➢ **Interprétations**

Les performances de la régression logistique peuvent dépendre de la manière dont les imputations ont été effectuées dans le cahier précédent. Une sélection plus stricte des caractéristiques (en fonction de leur taux de remplissage, par exemple) pourrait modifier les performances de ce classifieur. Une réduction de dimension telle que l'ACP permettrait également de sélectionner les caractéristiques les plus pertinentes.
NB : *Néanmoins, l'ajustement fin des caractéristiques avec le modèle afin d'obtenir le meilleur score possible n'entre pas dans le cadre de ce projet.*

Bien que la régression logistique soit très proche en termes de performances, le classifieur LightGBM présente d'autres propriétés intéressantes. Il est capable de gérer des caractéristiques catégorielles ordinales, sa nature basée sur des arbres décisionnels en fait aussi un bon candidat pour l'interprétabilité du modèle et il est moins sensible aux valeurs aberrantes que la régression logistique. En outre, il est plus rapide, plus efficace en termes de mémoire et sa précision est généralement meilleure que celle des autres classifieurs de type *gradient boosting*.

⇨ Dans l'ensemble, les observations semblent cohérentes compte tenu de la nature et de la sensibilité de chaque type de modèle testé. Le classifieur LightGBM montre de bonnes performances dans toutes les situations, dont les meilleures lorsqu'associées à la méthode de *cost-sensitive* pour gérer le déséquilibre des classes, et répond à tous les critères de classification requis pour ce projet.

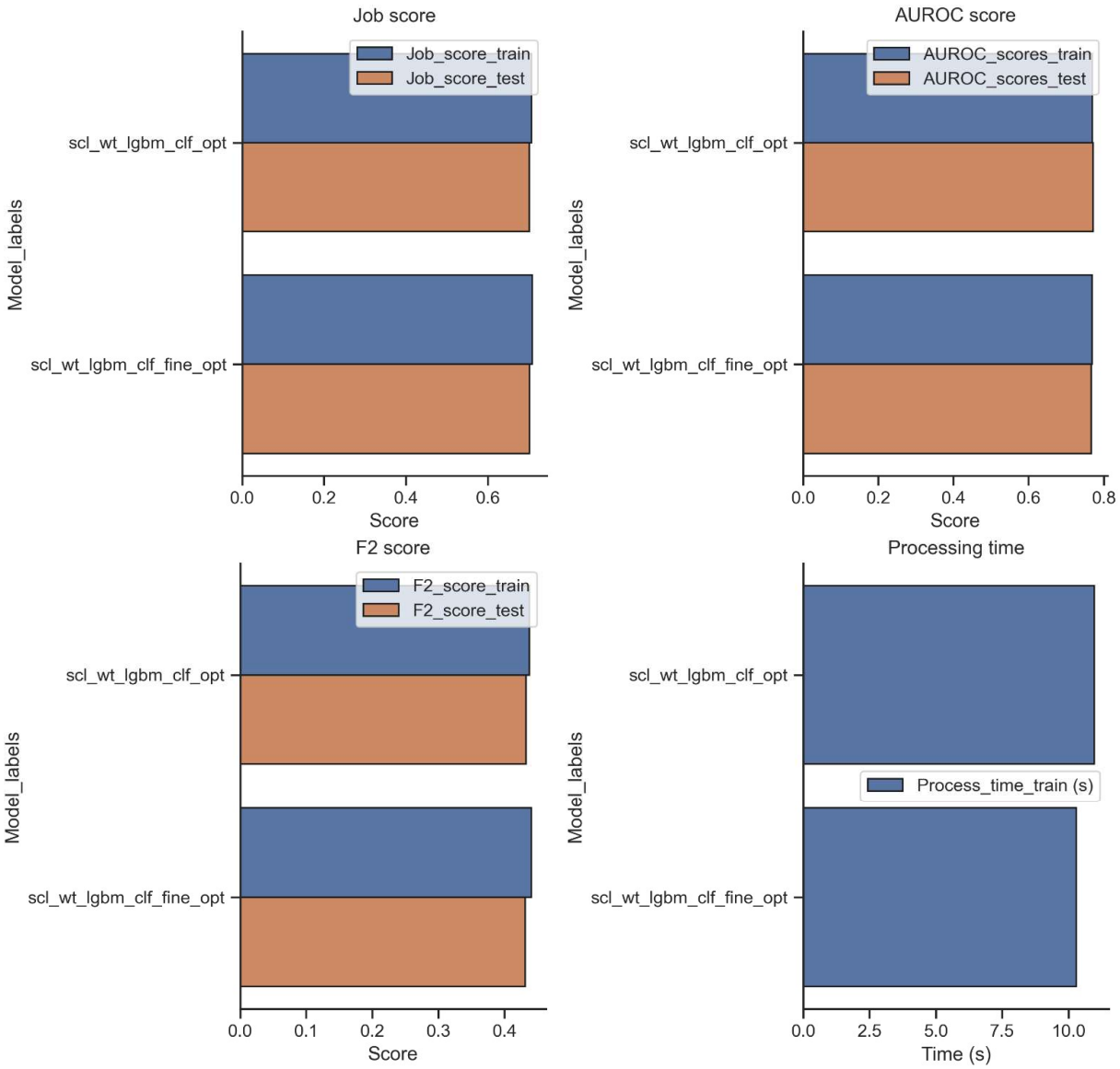


Figure 3 : *Résultats obtenus après optimisation poussée des hyperparamètres du modèle sélectionné (LightGBM) par validation croisée sur le jeu d'entraînement et sans cross-validation sur le jeu de test.*

Après 500 itérations, une légère variation de la performance peut être observée (~0,5 %) sur tous les indicateurs et le temps de traitement de la validation croisée a diminué d'environ 10 %.

⇨ L'optimisation poussée des hyperparamètres n'a pas apporté d'amélioration significative dans l'espace de valeurs défini. Peut-être que le fait d'être plus stricte avec la sélection des caractéristiques et l'imputation de leurs valeurs manquantes pendant le feature engineering ou d'étendre la plage de valeurs pour un plus grand nombre d'estimateurs pourrait améliorer les performances au détriment du temps d'entraînement dans le second cas.
On peut aussi noter que le modèle montre également une bonne capacité à généraliser, quel que soit l'indicateur, avant et après l'optimisation poussée.

III) Interprétabilité du modèle

***NB :** Les graphiques présentés ci-dessous sont tirés du dashboard. Il a été jugé que pour des personnes non averties, il est plus intuitif de comprendre que plus la probabilité est élevée plus elles ont de chances d'obtenir un prêt.*

⇨ Ainsi, les concepts vus jusqu'ici, de dossiers classés positifs (pas assez fiables) ou négatifs (assez fiables), voient leurs valeurs inversées.

1) Importance des features

Les modèles testés offrent tous une méthode permettant d'extraire les features ayant le plus d'influence sur la classification. Cependant, après de multiples recherches, il s'avère qu'il existe des techniques plus générales (adaptées à la plupart des modèles) et, semble-t-il, souvent plus fiables comme SHAP par exemple, qui est utilisé dans ce projet.

La méthode SHAP (*SHapley Additive exPlanations*) consiste à calculer la valeur de Shapley pour toutes les features de tous les individus. Autrement dit, la moyenne de l'impact d'une feature sur la prédiction renvoyée par le classifieur pour toutes les combinaisons de variables possibles.

⇨ La somme des effets de chaque variable explique alors la prédiction.

Ce score permet de disposer d'une information relative à l'importance globale des features dans le modèle qui peut à la fois être débattue et validée avec les experts métier pour améliorer le modèle et favoriser la transparence vis-à-vis des clients sur les éléments qui ont le plus influencé la prédiction.

Un autre avantage de SHAP est sa capacité à pouvoir offrir en plus d'une analyse locale une analyse globale. En effet, en moyennant les valeurs absolues de shapley de chaque feature au niveau local, il est possible de remonter à l'importance globale de celles-ci.

2) Interprétabilité globale

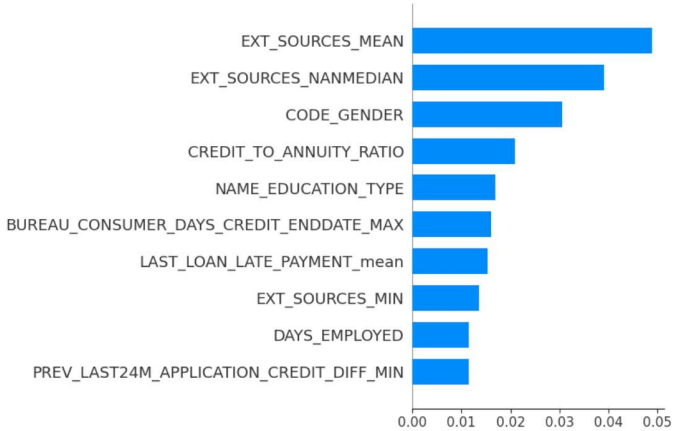


***Figure 4 :** Extrait des features globalement les plus influentes sur les prédictions du modèle, classées selon leurs scores attribués par l'algorithme SHAP.*

La figure ci-dessus, montre les 10 features les plus influentes sur les résultats de prédiction du modèle d'après les valeurs absolues de shapley.
Cette classification des features, par leur degré d'influence sur le classifieur, représente un premier niveau d'interprétation du modèle.

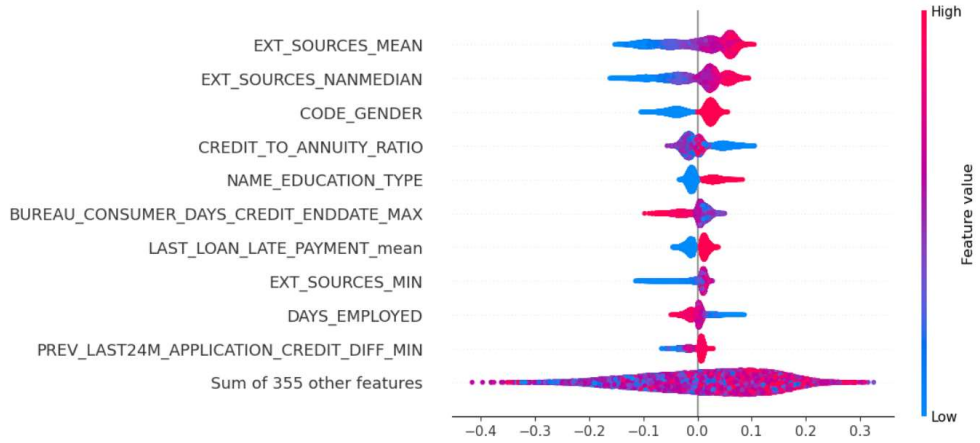


Figure 5 : *Distribution des dossiers des clients du dataset selon les features globalement les plus influentes sur les prédictions du modèle. Les couleurs représentent les valeurs que peuvent prendre ces features.*

Les points affichés sur la représentation *« beeswarm »* représentent les dossiers dont la couleur varie en fonction de la valeur de la feature (petite à grande valeur : bleu à rouge). Ces points sont positionnés sur l'axe des abscisses selon leurs valeurs SHAP caractérisant l'impact des valeurs que prend la variable pour chaque client sur son importance dans les prédictions.

Ex : Beaucoup de points rouges localisés sur la partie droite du graphique montrent que les valeurs élevées contribuent positivement à l'estimation de remboursement d'un crédit et par conséquent, à l'attribution de ce dernier. Si l'on regarde la variable CODE_GENDER par exemple, on peut remarquer que les femmes (encodées par la valeur 1) ont plus de chance de rembourser leurs prêts que les hommes (encodés par la valeur 0) d'après le modèle.

3) Interprétabilité locale

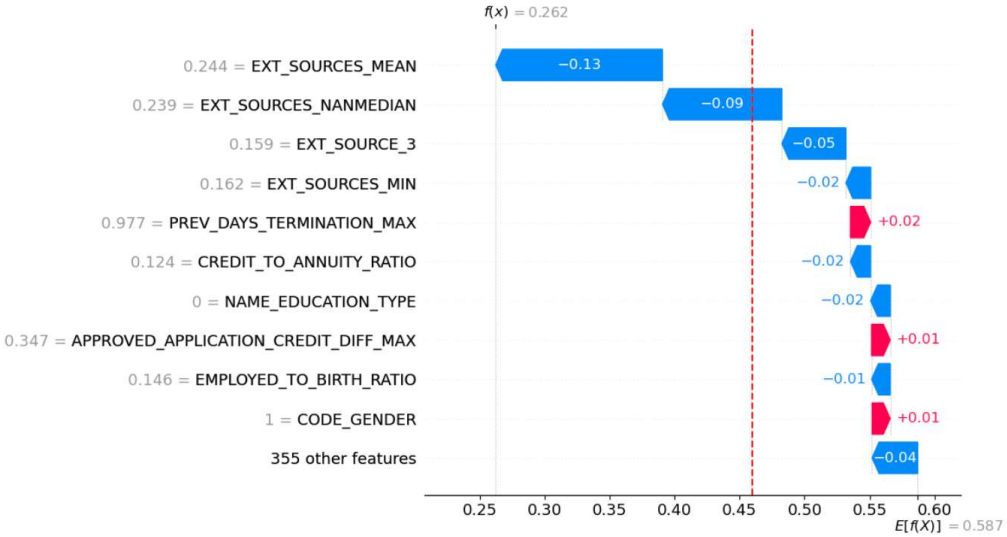


Figure 6 : *Extrait des features les plus influentes sur les prédictions du modèle pour un client anonyme qui se voit refuser sa demande de prêt, classé selon le score attribué par l'algorithme SHAP. Le trait rouge représente le seuil de classification.*

Le graphique permet d'observer quelles sont les features qui semblent le plus influencer la recommandation du modèle pour un client tout en montrant de façon simple, de quelle manière chaque feature a pu contribuer au résultat délivré par le modèle en partant de la valeur moyenne de shapley calculée sur tous les clients du jeu de données d'entraînement :

- **En rouge :** les features qui ont influencé positivement l'attribution d'un crédit au client.
- **En bleu :** les features qui ont influencé négativement l'attribution d'un crédit au client.

IV) Analyse de data drift

1) Data drift

Entre le jeu de données d'entraînement celui des nouveaux clients à classer, on observe un drift d'environ 18 %. Il serait sans doute possible de le réduire et rester plus proche du profil des données d'entraînement en traitant les dossiers de demande de prêt plus rapidement.

⇨ Cependant, on peut considérer que pour le moment cette valeur est normale.

2) Simulation de drift dans le temps

NB : *Une feature est considérée driftée si son profil de valeurs change de plus de 5 %.*

La simulation se déroule sur un peu moins d'une quarantaine d'années. Elle débute à partir des 20 premières années de données, puis mesure l'influence sur le profil global des données déjà enregistrées lors de l'ajout de données supplémentaires (le data drift) de plus en plus récentes par itération de 2 ans.

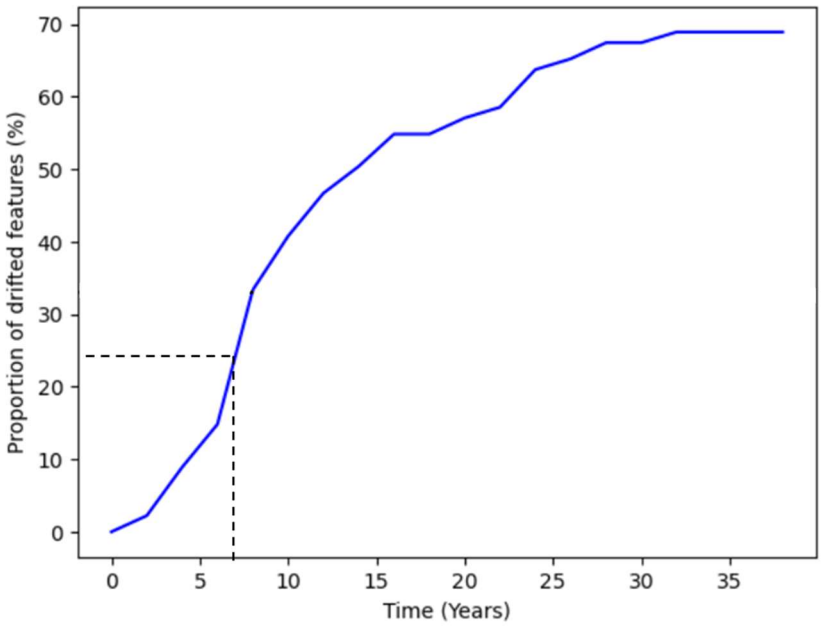


Figure 7 : *Simulation de data drift des 135 features les plus influentes sur les prédictions du modèle (influence relative supérieure à 0.1 % d'importance).*

Sur la figure, on peut remarquer que le profil des données change de plus en plus rapidement au cours des 10 premières années pour atteindre 33 % de drift aux alentours de 8 ans (dont 13 % entre la 6e et la 7e année).
Dans le but de conserver des performances optimales, il est nécessaire d'établir un seuil de drift maximal, qui une fois atteint, indique la nécessité d'effectuer une maintenance sur notre outil (i.e. de réétalonner / réentaîner le modèle sur le nouveau profil de données).

Une méthode pour estimer ce seuil consiste à observer, parmi les features détectées comme ayant driftées, celles faisant partie du top 10 des features les plus influentes. Il est ainsi possible de considérer que lorsque 2 ou 3 features très influentes pour le modèle ont déjà drifté, il est préférable d'effectuer une maintenance.

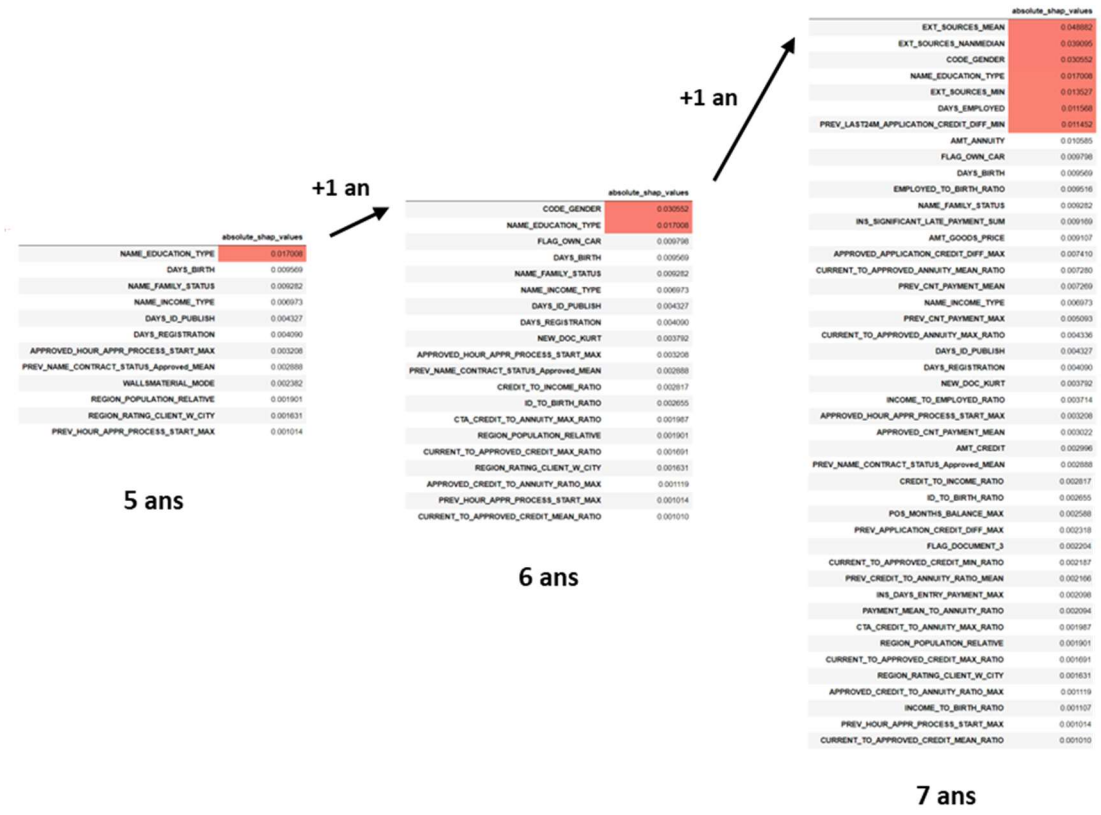


Figure 8 : *Features ayant drifté au cours des 7 premières années. En rouge les features faisant partie du top 10 des features les plus influentes sur les prédictions du modèle d'après les valeurs SHAP.*

Comme le montre cette figure, on peut remarquer qu'entre 6 et 7 ans beaucoup de features commencent à drifter et notamment une proportion significative des features du top 10.

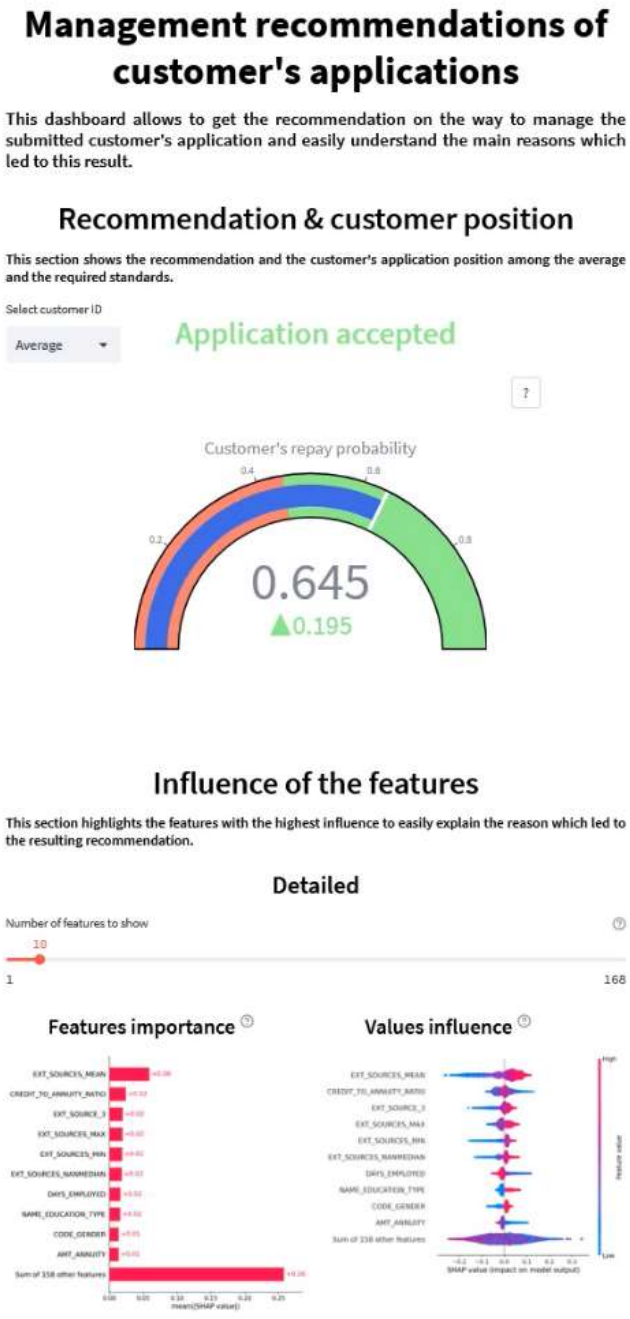
> ⇨ D'après ces 2 figures, il semble judicieux de fixer une première date de maintenance entre la 6e et la 7e année correspondant à un drift global d'environ 25 %. Toutefois, une discussion avec les experts métier permettrait d'ajuster ou de confirmer cette échéance.

V) Dashboard interactif

Le dashboard est composé de 2 parties :

- Un **back end** : API flask qui va s'occuper de charger le modèle, faire la prédiction correspondant à la requête reçue (le client sélectionné), interpréter la prédiction et renvoyer les résultats à la partie front-end.

- Un **front end** : Interface interactive créée avec streamlit affichant les résultats provenant de l'API flask et à partir de laquelle le chargé de client émet les requêtes vers la partie back-end.



Ces deux parties sont hébergées en ligne sur Heroku et communiquent l'une avec l'autre. Elles sont accessibles séparément par leurs propres adresses web de type https après déploiement.

NB : *Le dashboard n'est déployé que pour les tests pour limiter les coûts. Une vidéo explicative du fonctionnement du dashboard est aussi disponible.*

VI) Perspectives d'amélioration

La réalisation du modèle a nécessité la conception de nombreux blocs de transformation et de traitement des données dans son architecture (comme en attestent les multiples notebooks). Chacun fait appel à des méthodes paramétrables et les résultats sont dépendants des paramètres choisis.

⇨ Ainsi, les choix des paramètres utilisés à chaque étape peuvent être changés ou affinés.

1) Sélection des variables

Les informations disponibles relatives à l'importance des variables devraient être débattues avec les experts métier en vue de définir les stratégies techniques à tester dans les différents blocs concernés :

- Seuil des valeurs manquantes.
- Techniques d'imputation.
- Corrélations entre variables.
- Réduction de dimensions et seuil de variance.

2) Équilibrage des données

- Essayer d'autres techniques d'attribution des poids lors de l'apprentissage sensible aux coûts.
- L'équilibrage des données introduit des données artificielles potentiellement incohérentes dans le cas du suréchantillonnage ou retirer des données importantes dans celui du sous-échantillonnage. Des tests supplémentaires peuvent être réalisés en variant les algorithmes et leurs paramètres. Il est aussi possible de combiner les techniques de suréchantillonnage et de sous-échantillonnage pour combiner les avantages des 2.

3) Fonction coût métier

Communiquer les règles métier et les critères financiers relatifs aux pertes et profits permettrait d'affiner davantage la fonction d'évaluation du gain pour l'approcher encore plus des contraintes métier réelles.

4) Classifieurs et optimisation des hyperparamètres

Plusieurs classifieurs ont été testés avant de retenir LightGBM. Cependant, il est aussi possible d'en essayer d'autres ou d'en combiner plusieurs par *stacking*. L'empilement de modèles peut améliorer les performances en combinant les avantages de chacun.

Pour encore affiner les performances du modèle, il est possible de continuer son optimisation avec davantage d'itérations sur l'algorithme d'Hyperopt ou d'ajouter davantage d'hyperparamètres, voire éventuellement, de sélectionner d'autres plages de valeurs pour les hyperparamètres testés. Cependant, il est certainement plus prometteur de revenir sur l'EDA et le feature engineering afin de tester les performances du modèle sur d'autres choix d'imputation des valeurs manquantes ou de sélection et de pré-processing des features.

5) Interprétabilité du modèle

La méthode SHAP offre des résultats intéressants, en particulier pour l'interprétabilité locale. Par contre, son intégration dans une application front-end comme streamlit demeure assez mal prise en charge et demande souvent l'ajout de code, voire, l'import de bibliothèques supplémentaires non officielles pour compenser ce manque de compatibilité et afficher ses graphiques.

Concernant l'interprétabilité globale, les algorithmes basés sur les forêts aléatoires (mais également SHAP) permettent d'identifier les variables influentes. Cependant, d'autres approches comme les permutations de variables peuvent être testées pour valider ces résultats.

6) Dashboard interactif

- Séparer en 2 onglets différents l'application front-end avec :
    - Les interprétations locales relatives aux clients sur le premier onglet.
    - Les interprétations globales, plutôt destinées aux chargés de communication, avec des fonctionnalités supplémentaires comme la matrice de confusion ou la capacité de changer le seuil de classification à la volée pour observer l'évolution des résultats en direct pour un client ou l'ensemble d'entre eux, par exemple.
- Ajouter un panneau latéral affichant les caractéristiques propres au client (âge, genre, adresse …) pour en faciliter l'accès.
- Après discussion avec les experts métier quelques améliorations sur la jauge de confiance pourraient aussi être apportées comme sa division en différentes zones de confiance, définies en amont, associées à des protocoles éventuellement différents pour gérer les dossiers clients.

7) Simulation du data drift

Discuter avec les experts métier et en apprendre davantage sur le coût monétaire d'une maintenance pour mieux adapter les seuils de drift de la simulation et définir une période de maintenance en adéquation.