Projet: Implémentation d'un modèle de scoring

DS : C4M1N

Note méthodologique

Sommaire :

Rappel de la problématique métier :

Le but est de déterminer la probabilité de défaut de paiement d'un client compte tenu des données à disposition. Ceci se résume à un problème de classification binaire pour déterminer si un client est en défaut ou non de paiement.

Pour cela, un modèle est entraîné pour prédire la probabilité pour un client d'être en défaut ou non.

Une interface graphique client (Dashboard) intégrant le modèle est proposée pour pouvoir comprendre et interagir avec la prédiction affichée par ce modèle en fonction des données sur le client.

Une interface de programmation applicative (API) est déployée pour permettre de faire le lien entre l'interface graphique et le modèle entraîné.

L'interface graphique doit permettre de comprendre la prédiction effectuée par le modèle.

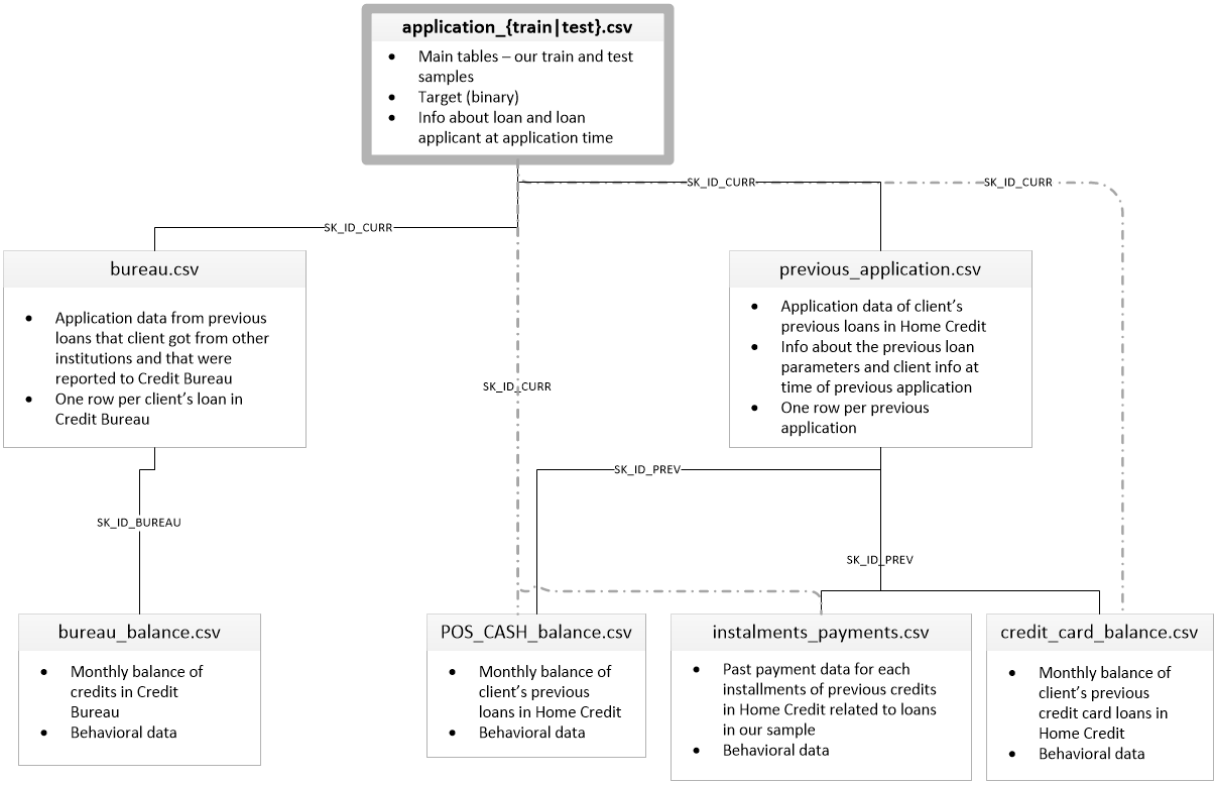
1) Méthodologie d'entraînement du modèle

Plusieurs modèles ont été entraînés pour ce projet.

Les données utilisées proviennent du site Kaggle à cette adresse : https://www.kaggle.com/c/home-credit-default-risk/data

Dans un souci d'efficacité, l'exploration et l'orientation prise sur la préparation des données ont été effectuées sur la base d'un kernel Kaggle existant : https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction#Introduction:-Home-Credit-Default-Risk-Competition

Le traitement et la préparation des données ont pris en compte l'ensemble des ressources disponibles en plus de celle du kernel existant. Soit sept tables de données ont été fusionnées pour extraire les informations utiles au projet. Voici le schéma des tables avec les relations existantes :



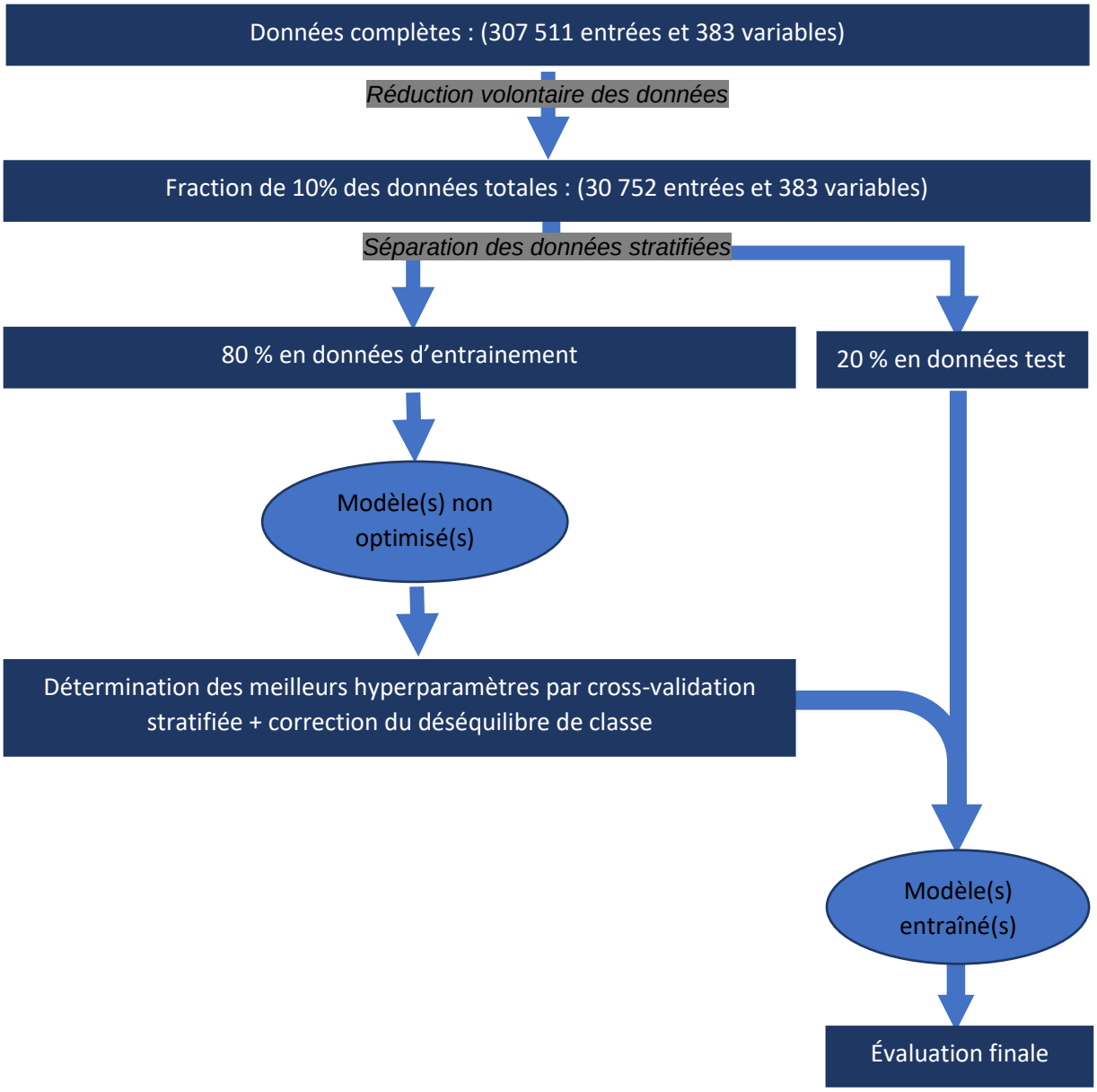
Après traitement et préparation des données, le jeu de données final obtenu pour ce projet est constitué d'un dataframe de 307 511 entrées avec 383 variables. Cette préparation a été effectuée avec un notebook jupyter disponible en annexe, ainsi que le dataset (jeu de données) « df_custom_global_optim.pkl.gz »

Pour les besoins d'entrainement des modèles, une fraction de 10% des données totales a été prise et sur cette fraction une séparation en 80% de données d'entrainement et 20% de données de tests a été réalisée.

Le choix de la fraction a été définie en fonction du temps et des capacités de calcul de l'ordinateur à ma disposition soit : i5 Core avec 8Go de RAM sans GPU.

Étant conscient de cette limitation pour l'entraînement des modèles, celle-ci constitue de ce fait un axe d'amélioration à prendre en compte pour optimiser les modèles proposés et ainsi que les résultats obtenus.

Pour, résumer schématiquement :



Les données initiales présentent un fort déséquilibre par rapport à la classe à prédire (client en défaut de paiement). Ce déséquilibre de l'ordre de 91,9% contre 8,1% pour la classe à prédire est à prendre en compte dans la détermination des hyperparamètres du modèle à entraîner. En effet, un modèle naïf prédirait systématiquement la classe majoritaire avec une « accuracy » assez élevée et ne permettrait pas de « détecter » les clients minoritaires en défaut de paiement que l'on souhaite identifier.

Pour résoudre ce problème de classification binaire à fort déséquilibre, plusieurs approches ont été testées lors de l'entraînement des modèles dont :

- L' « under-sampling » ou la réduction de la classe majoritaire ou profit de la classe minoritaire à prédire

- La génération de clients « synthétiques » créés à partir des profils de clients existants en défaut (SMOTE)
- Le paramétrage de l'hyperparamètre « class_weight » ou de « scale_pos_weight » pour la prise en compte de la proportion du déséquilibre de classe par le modèle
- Le paramétrage de « sample_weights » qui correspond à la prise en compte de la pondération du poids associé aux observations d'entraînement de telle manière à ne pas défavoriser les observations appartenant à la classe minoritaire pour le modèle.

Plusieurs modèles ont été utilisés avec ces approches pour établir les meilleures prédictions possibles.

La recherche des meilleurs résultats de prédiction a été effectuée avec les modèles suivants :

- RandomForest
- XGBoost
- CatBoost
- SVC

Remarque importante : la recherche a été limitée par la puissance de calcul disponible et la taille initiale des données d'entraînement, ici limitée sur 10 % du jeu complet pour éviter des temps de calculs ingérables.

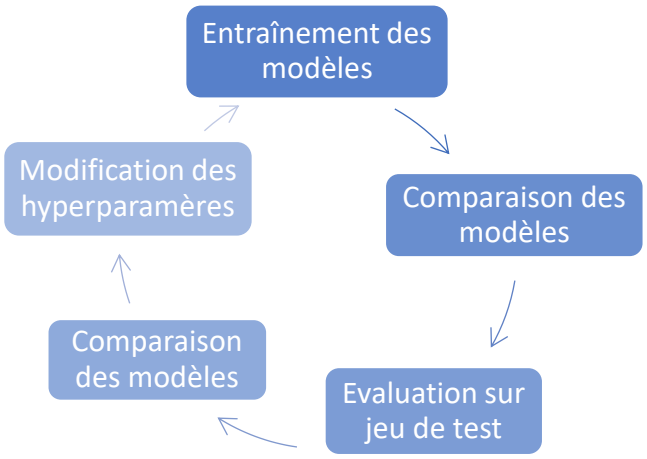
Le paramétrage de « class_weight » ou de « scale_pos_weight » ont été employés car donnant de meilleurs résultats par rapport aux autres méthodes.

L'« under-sampling », SMOTE et sample_weights ne donnant pas de résultats significativement meilleurs avec les données préparées, leur utilisation n'a pas été retenue.

L'entraînement d'un modèle de « stacking » a été également envisagé pour permettre d'obtenir des résultats supérieurs par rapport à chacun des modèles respectifs. La puissance de calcul disponible pour réaliser l'opération n'a pas permis de déterminer la meilleure combinaison de modèles parmi les 11 possibilités envisageables avec les quatre modèles entraînés.

L'emploi de méthodes comme « GridSearchCV » et « Hyperopt » ont été utilisées pour déterminer dans un temps raisonnable les meilleurs hyperparamètres des modèles parmi un espace d'hyperparamètres prédéfinis. Lors des entraînements, cet espace de recherche a été réduit progressivement en fonction des résultats obtenus avec les modèles.

Pour résumer schématiquement le processus d'entrainement des modèles :



2. La fonction coût, l'algorithme d'optimisation et la métrique d'évaluation

a. Fonction de coût

L'ensemble des modèles ont été entraînés avec la technique de validation croisée en se basant sur un espace étendu de combinaison d'hyperparamètres possibles.

Les fonctions de coût diffèrent suivant les modèles utilisés :

Nom du modèle entraîné	*Fonction de coût*
RandomForestClassifier	Entropie : maximisation du gain d'information lors de chaque création de nœud
XGBClassfier	Régression logistique binaire
SVC	Fonction de perte à marge maximale « hinge loss function » en anglais
CatBoostClassifier	Régression logistique binaire

b. Métrique d'évaluation :

Du fait, du déséquilibre par rapport à la classe à prédire : 92% en arrondissant de clients en règle contre 8% en défaut de paiement, il est nécessaire de choisir une métrique appropriée.

D'après la matrice de confusion utilisée dans la librairie Python Scikit-learn :

		Valeurs prédites	
Valeurs réelles		Classe Négative	Classe Positive
	Classe Négative	Vrais négatifs (Tn)	Faux positifs (Fp) *Erreur type 1*
	Classe Positive	Faux négatifs (Fn) *Erreur type 2*	Vrais positifs (Tp)

*(*utilisation de l'abréviation anglaise pour plus de facilité : Tn, Tp, Fn et Fp)*

La problématique métier, sans précision dans la part de la société « Prêt à dépenser », présente une double composante :

- D'un côté, il n'est pas souhaitable qu'un client « en règle » soit catégorisé à tort en « en défaut » (ce qui priverait la société d'un client potentiel et donc d'une absence possible de revenus)
- De l'autre, le fait qu'un client « en défaut » soit classé par erreur en « en règle » occasionnerait des frais de recouvrement ou de procédure à la société, soit une perte d'argent.

Bien que l'objectif soit d'estimer au mieux la probabilité de défaut de paiement d'un client, il faut tenir compte de cette « double » réalité d'un point de vue métier.

La recherche a donc été menée avec ce double objectif, c'est-à-dire celui de minimiser au maximum les Faux négatifs (erreur de type 2), tout en évitant de maximiser les Faux positifs (erreur de type 1).

Il est donc nécessaire de maximiser le Recall (Sensibilité) tout en ne dégradant pas la Précision de manière importante.

La F mesure ou F score qui est la moyenne harmonique entre Recall et Précision est intéressante comme métrique à envisager lors de l'entraînement des modèles. Mais il faut tenir compte du déséquilibre de classe qui pousse à modifier la F mesure en introduisant un coefficient régulateur en faveur du Recall.

Il faut attribuer une valeur à un paramètre appelé Beta pour une F mesure « corrigée » : la F Beta mesure :

- Tp = Vrais Positifs
- Tn = Vrais Négatifs

- Fp = Faux Positifs
- Fn = Faux Négatifs

$$Sensibilité\ (Recall) = \frac{Tp}{Tp + Fn}$$

$$Précision = \frac{Tp}{Tp + Fp}$$

$$F\ mesure\ standard\ (F1\ score) = 2 * \frac{Sensibilité * Précision}{Sensibilité + Précision}$$

Avec l'introduction de Beta, la F Beta mesure devient :

$$F\ Beta\ score = (1 + Beta^2) * \left(\frac{Sensibilité * Précision}{Beta^2 * Sensibilité + Précision} \right)$$

Pour l'ensemble des modèles testés, le F Beta score a été la métrique à maximiser.

Beta est le paramètre à déterminer pour obtenir la meilleure balance entre Précision et Sensibilité. Sa valeur a été déterminée en fonction des résultats d'entraînement et de la validation sur le jeu de test.

Nom du modèle	*Valeur de Beta*
RandomForestClassifier	2.0
XGBClassifier	2.2
CatBoostClassifier	2.0
SVC	1.8

Cependant en l'absence d'indications complémentaires de la société « Prêt à dépenser » sur les niveaux acceptables d'erreurs à optimiser (erreur de type 1 et de type 2), le choix s'est porté sur la meilleure balance possible entre Sensibilité et Précision pour chaque modèle. C'est-à-dire avec une valeur de Beta propre au modèle, ce qui permet d'avoir une métrique F Beta Score particulière à chaque modèle à maximiser.

c. Algorithme d'optimisation

Pour l'ensemble des modèles mis en œuvre, les meilleurs hyperparamètres ont été déterminés ainsi que la valeur de Beta permettant d'obtenir la meilleure balance entre Sensibilité et Précision.

En se basant uniquement sur les résultats, le modèle le plus performant est le Support Vector Classifier (SVC).

Cependant, si l'on considère le temps de calcul pour la prédiction d'un client ou de l'ensemble des clients proches à celui-ci, le temps d'estimation du modèle SVC est le plus long parmi l'ensemble des modèles testés.

Il faut donc tenir compte à la fois du temps d'exécution d'un modèle et également de sa capacité à prédire en production. Sinon l'expérience utilisateur via le dashboard risque d'être entachée à cause d'un temps de calcul trop long malgré une bonne performance.

Dans la version du dashboard présentée pour ce projet, le choix a été laissé à l'utilisateur expérimenté de sélectionner un modèle plutôt qu'un autre après consultation de ses performances.

Un modèle de « stacking » a été envisagé pour tirer le meilleur parti de la combinaison de plusieurs modèles. En revanche son évaluation n'a pas été possible par manque de puissance de calcul à ma disposition. Il aurait été intéressant de déterminer le niveau de gain de performances par rapport à un simple modèle et de savoir également si le modèle de « stacking » est plus lent ou rapide à la prédiction.

3. Interprétabilité du modèle

Dans un but de transparence et de compréhension pour l'utilisateur lambda concernant la lecture du dashboard, les quatre premières variables les plus importantes dans la prédiction d'un modèle sont affichées et modifiables dans l'interface.

Pour déterminer les variables par modèle, la librairie LIME de Python a été utilisée dans le notebook jupyter de préparation des données. Cette librairie est disponible à cette adresse : https://github.com/marcotcr/lime

LIME a été préféré à la librairie SHAP pour sa facilité de mis en œuvre et la visualisation simple des explications pour une simple prédiction sur l'un des clients du dataset.

4. Les limites et les améliorations possibles

Avec une puissance de calcul suffisante, l'entraînement sur la totalité du dataset moins un jeu de test entre 25 et 30%, aurait été intéressante pour tenter d'améliorer significativement les performances des modèles existants.

Mon estimation sur une configuration hardware appropriée serait : un processeur i7 / i9 Core ou double Xeon avec 64 ou 128 de RAM avec GPU puissant pour un dataset de 300 000 entrées avec plus de 350 variables.

Ensuite la valeur de Beta étant fixée pour chaque modèle par détermination de la meilleure balance Sensibilité / Précision : un complément d'informations avec la société « Prêt à dépenser » serait utile pour déterminer les niveaux acceptables ou non d'erreurs. Dans le but de définir une valeur fixe de Beta et d'entraîner l'ensemble des modèles avec la métrique parfaitement adéquate.

Autre point : les données initiales pourraient être réévaluées pour obtenir un dataset avec moins de variables au total et avec une logique propre au domaine d'activité. (feature engineering spécifique)

La traduction de certaines variables normalisées dans l'interface utilisateur en valeurs réelles métier et manipulables pourraient améliorer l'expérience utilisateur avec le dashboard.

Le paramétrage sur mesure de plusieurs variables simultanément à partir d'un profil client existant pourraient également être un axe d'amélioration. Avec la possibilité de partir d'un profil vierge ou conseillé « sans risque » à partir des variables catégorielles classiques importantes (e.g. : profession, niveau de revenu, etc.)