Implémentation d'un modèle de scoring

Le contexte

- Une société financière, nommée "Prêt à dépenser", qui propose **des crédits à la consommation** pour des personnes ayant **peu ou pas du tout d'historique de prêt**.
- L'entreprise souhaite mettre en œuvre un outil de "scoring crédit" pour calculer la probabilité qu'un client rembourse son crédit, puis **classifie la demande en crédit accordé ou refusé**. Elle souhaite donc développer un algorithme de classification en s'appuyant sur des **sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).**
- De plus, les chargés de relation client ont fait remonter le fait que les clients sont de plus en plus demandeurs de **transparence vis-à-vis des décisions d'octroi de crédit**.

Le jeu de données

Celui-ci se compose de plusieurs fichiers csv, comportant des informations diverses sur la demande en cours et son demandeur (Application) et des informations sur de précédents emprunts effectués dans le même établissement ou un autre.

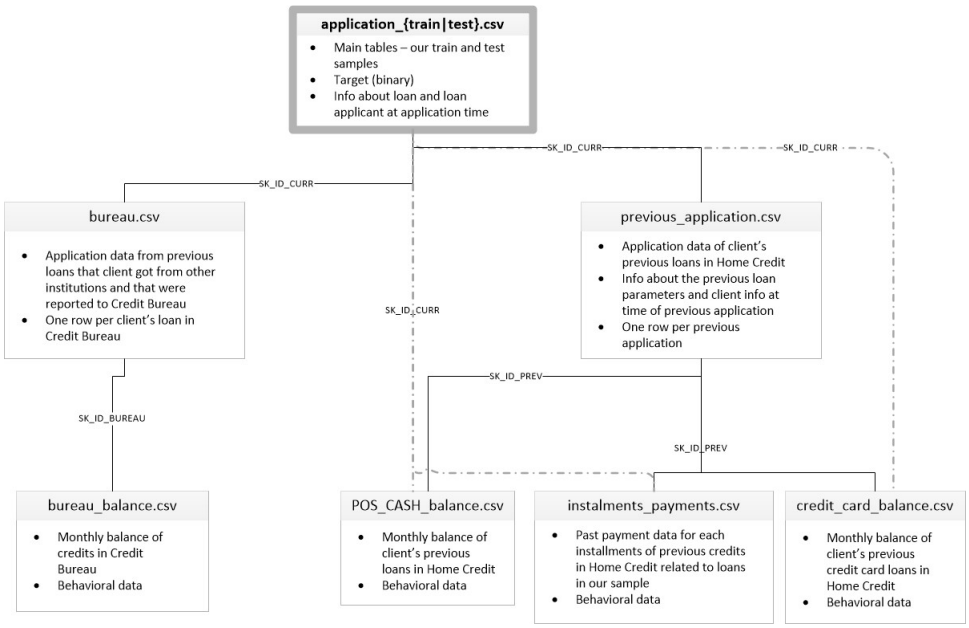

Figure 1: Composition du jeu de données

Ces données sont préalablement traitées afin d'en tirer un jeu de données unique où chaque ligne correspond à un client

On peut trouver une analyse très détaillée de ce défi Kaggle en trois parties sur Medium mais j'ai préféré ne le parcourir que très rapidement pour laisser la place à un travail plus personnel.

https://medium.com/analytics-vidhya/home-credit-default-risk-part-1-business-understanding-data-cleaning-and-eda-1203913e979c

https://medium.com/@dhruvnarayanan20/home-credit-default-risk-part-2-feature-engineering-and-modelling-i-be9385ad77fd

https://medium.com/@dhruvnarayanan20/home-credit-default-risk-part-3-modelling-ii-and-model-deployment-3b3f92e1926c

Préparation des données

J'ai repris en grande partie le feature engineering du kernel suivant https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features, que j'ai adapté légèrement en effectuant des opérations supplémentaires.

- Suppression des variables les moins bien renseignées
- Regroupement de certaines valeurs catégorielles pour éviter d'avoir trop de dimensions après l'encodage
- One-hot-encoding avec NAN comme catégorie

- Imputation par la médiane pour les variables numériques de la table **Application**

Après feature engineering et merge des dataframes :

- Imputation des valeurs vides par 0
- Sélection des 100 variables les plus « importantes » (via selectkbest – ANOVA -)

Méthodologie d'entraînement du modèle

Nous avons affaire ici à un problème de classification binaire sur un jeu de données déséquilibré, où la classe positive (défaut de paiement) qu'on cherche à prédire ne représente que 8% des données.

De plus, dans ce contexte métier (secteur bancaire) l'impact d'une erreur de classification diffère suivant sa nature.

En effet, accorder un prêt à quelqu'un qui se retrouve en défaut de paiement (faux négatif) n'a pas la même conséquence pour la banque que de refuser un prêt à un bon client (faux positif).

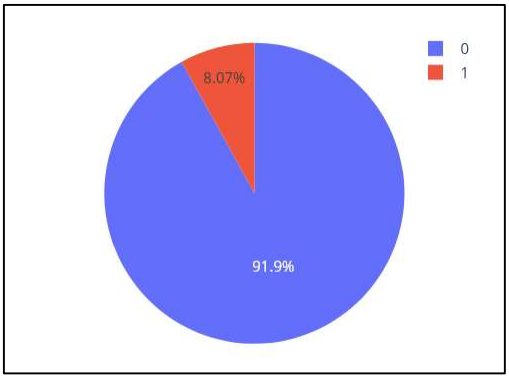

Figure 2: Répartition des cibles

Cela supposera donc de s'intéresser par la suite à plusieurs aspects :

- L'équilibrage du jeu de données
- Le coût des erreurs de classifications
- Le choix de la métrique d'évaluation

Gain métier et métrique d'évaluation

VRAIS NÉGATIFS +2	FAUX POSITIFS 0
FAUX NÉGATIFS -10	VRAIS POSITIFS 0

Figure 3 : Poids des prédictions

Pour cette étude, j'ai attribué un gain de 2 aux négatifs bien classés (un emprunteur rembourse avec des intérêts, il s'agit du gain de la banque) et j'ai attribué une perte de 10 aux positifs classés en négatifs (un emprunteur ne remboursant pas fait perdre la totalité du montant à la banque). J'ai fait le choix de considérer comme étant égal à 0 les bons payeurs refusés même si on pourrait prendre en compte aussi le manque à gagner en pénalisant ceux-ci.

Le gain de la banque peut donc se calculer ainsi :

$$\text{Gain} = 2 \times \text{nombre\_de\_négatifs\_détectés} - 10 \times \text{nombre\_de\_positifs\_non\_détectés}$$

Le gain total possible se définit sur l'ensemble de test de la manière suivante :

$$\text{Gain total possible} = 2 \times \text{nombre\_total\_de\_négatifs} - 10 \times \text{nombre\_total\_de\_positifs}$$

J'ai également défini un score qui me permettra d'évaluer les résultats des prédictions de mon modèle lors de l'apprentissage, et de l'utiliser lors de l'optimisation sous forme de scorer.

Gain score = Gain / Gain total possible

Ce score prend des valeurs entre **0 et 1** et correspond au ratio entre le gain calculé à partir de la matrice de confusion obtenu par prédiction sur un jeu de test, et le gain maximum pouvant être obtenu sur le jeu de test considéré. Plus il est élevé, meilleur est le modèle dans l'intérêt de la banque.

Comparaison de plusieurs modèles sur 50% des données

Le choix de ne comparer des classifieurs que sur une partie des données représente un gain de temps très important qui permet de choisir le modèle le plus pertinent à optimiser par la suite.

La stratégie retenue ici pour la comparaison est de calculer la moyenne de plusieurs scores obtenus par cross-validation avec cinq folds (StratifiedKfold) en conservant les valeurs par défaut des hyperparamètres. L'utilisation de folds (plis) consiste à diviser les données en autant d'ensembles de même taille que le nombre choisi, à apprendre sur 4 plis, et évaluer le résultat des prédictions sur le 5ème, en changeant à chaque itération de sorte que chaque pli a été utilisé pour la prédiction. Le stratified K-fold permet de conserver la répartition des différentes classes dans chaque pli.

J'ai comparé plusieurs modèles sur la moyenne des scores obtenus sur les 5 folds, ainsi que plusieurs stratégies de prise en compte du déséquilibre des classes (ou sa non prise en compte)

- Sans rééquilibrage des données : on conserve les données telles quelles et on n'utilise pas de paramètre particulier au niveau des modèles.

- Avec oversampling de la classe minoritaire en utilisant SMOTE (Synthetic Minority Oversampling Technique) : on synthétise des échantillons supplémentaires de la classe positive.

- En utilisant les paramètres class_weight ('balanced') ou scale_pos_weight (nombre de négatifs / nombre de positifs) des modèles. Cette approche permet au modèle d'apprendre automatiquement les poids de classe optimaux pendant l'entraînement, et elle a l'avantage plus facile à mettre en œuvre, et plus rapide, surtout lorsque que le volume des données est important.

Comparaison des durées d'entraînement des modèles

Le modèle 'DummyClassifier' fournit une baseline en faisant des prédictions ignorant les données d'entrée. Il peut prendre un paramètre « stratified » qui permet de se baser sur la probabilité des classes lors que la prédiction « au hasard ».

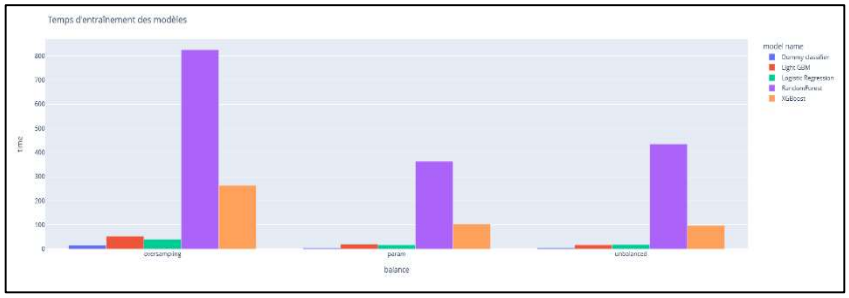

Figure 4: Durées d'entraînement

Le modèle LGBMClassifier l'emporte au temps, au coude à coude avec la régression logistique, en utilisant la version **paramétrée** gérant l'équilibre des classes. Il a de

surcroît l'avantage d'offrir plus de possibilités d'optimisation (un grand nombre d'hyperparamètres).

Comparaison des scores obtenus

Si on compare maintenant la moyenne des scores AUC, le modèle LGBMClassifier l'emporte de peu.

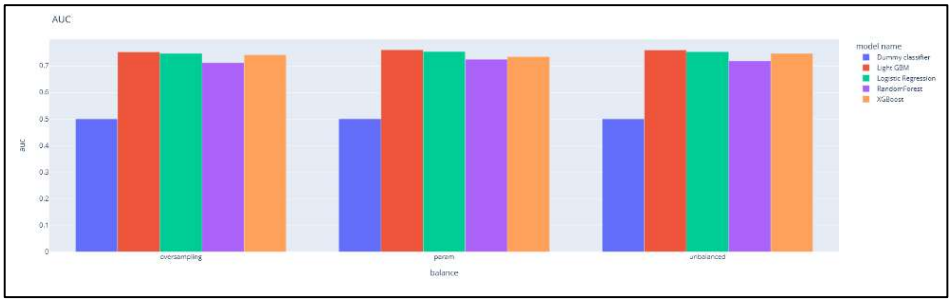

Figure 5 : Moyenne des scores AUC

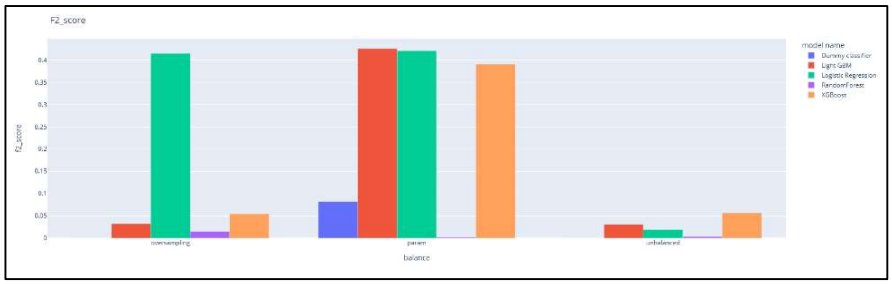

Figure 6 : Moyenne des f2_scores

De même pour le F2_score, correspondant au fbeta_score où beta est égal à deux.

On donne permet plus de poids au recall, c'est-à-dire à la proportion de vrais positifs détectés parmi tous les individus classés positifs.

En effet, un positif bien détecté nous évite une pénalité de -10.

Si on s'intéresse maintenant au gain, les différences entre les différents modèles sont moins marquées, et XGBoost semble être un peu plus performant.

Toutefois, la contrainte de temps me fera choisir Light GBM.

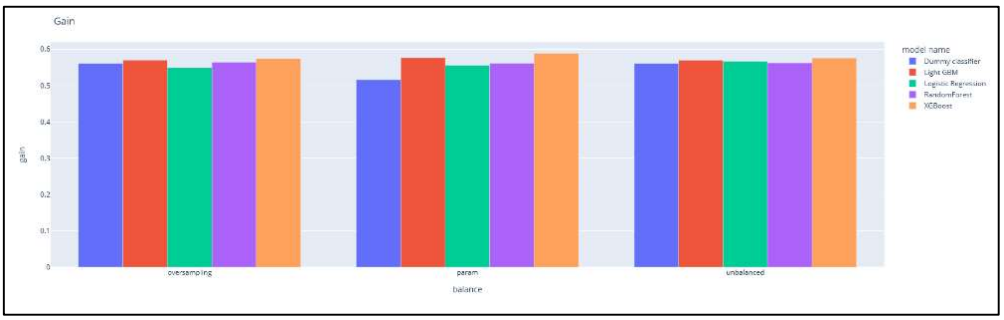

Figure 7 : Moyenne des gains

Optimisation du modèle

Recherche par grille et validation croisée

On divise les données en deux ensembles, un qui sera utilisé pour l'entraînement du modèle (train) et l'autre (20% des données) qui sera utilisée pour l'évaluation du modèle (validation).

L'optimisation des hyperparamètres est faite par une recherche par grille avec validation croisée (GridSearchCV). L'approche GridSearch utilise une grille de valeurs possibles pour chaque hyperparamètre. Elle effectue une validation croisée (5 plis, StratifiedKfold) sur chaque combinaison d'hyperparamètres de la grille et retourne le modèle qui a obtenu les meilleures performances en terme

du score choisi. D'autres approches sont possibles : RandomizedSearchCV, qui teste des combinaisons au hasard (jusqu'au nombre d'itérations choisi) ou encore BayesSearchCV où les valeurs des hyperparamètres sont choisies dans des espaces de recherche et non des listes de valeurs.

Avec le modèle le plus performant, on obtient un gain de 0.6 sur les données de validation (supérieur au score obtenu lors de la sélection du modèle sur la moitié des données).

```
Prédiction sur les données
  de validation – Scores
         obtenus

Confusion matrix:
 VN [[45673 10865] FP
 FN [ 2317  2648]] VP

auc : 0.7435
f2_score :0.3967
recall : 0.5333
gain : 0.6029
```

Maximisation du gain par le seuil de probabilité

Les scores obtenus se font par défaut avec une décision prise avec un seuil de probabilité de 0.5. On peut donc ajuster ce seuil de telle sorte que le gain soit maximisé, ceci en calculant les gains pour une série de seuils de probabilité.

On obtient ainsi un gain maximum de 0.623 en ne refusant un emprunt que si la probabilité de défaut de paiement dépasse 0.616.

Le graphe ci-dessous montre l'évolution du gain en fonction de la probabilité d'appartenance à la classe positive.

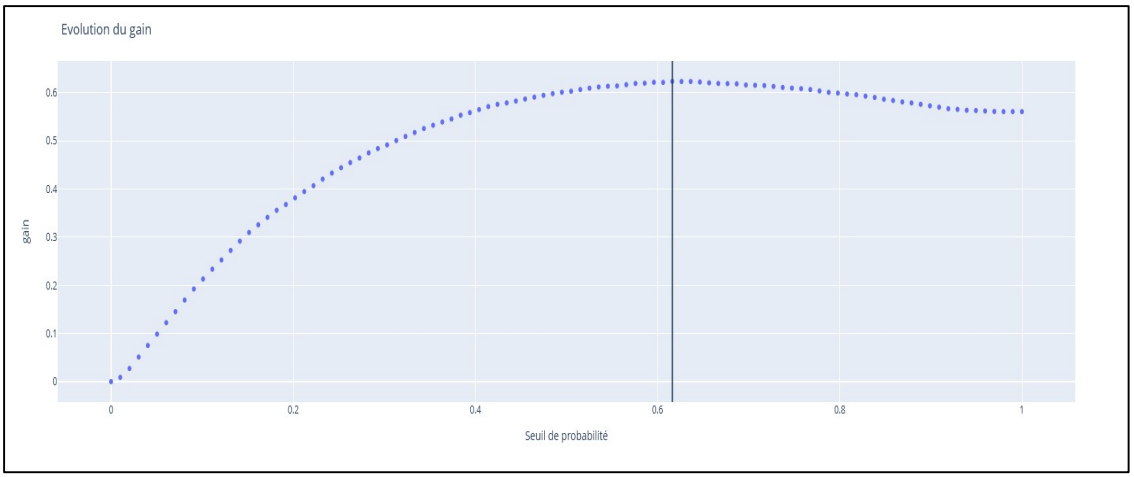

Figure 8 : Evolution du gain suivant le seuil de probabilité

Interprétabilité globale et locale du modèle

« *L'interprétabilité globale d'un modèle de machine learning désigne sa capacité à être compréhensible par des personnes non-expertes en termes de fonctionnement et de prise de décision. Un modèle de machine learning interprétable globalement est un modèle qui peut être expliqué de manière claire et concise, de sorte que les personnes qui ne sont pas familières avec les détails techniques du modèle puissent comprendre comment il fonctionne et pourquoi il prend les décisions qu'il prend.* »

Pour l'interprétabilité globale, on peut s'intéresser à l'attribut `_feature_importance` du modèle LGBMClassifier, qui attribue une valeur positive à chaque feature.

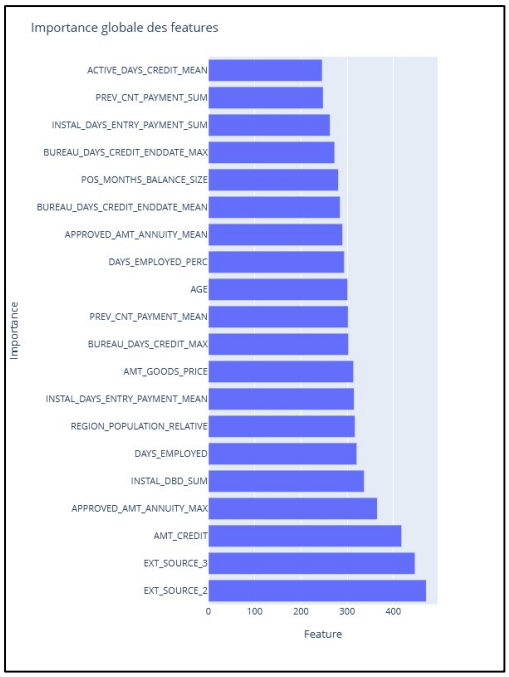

Figure 9 : Importance globale des features

La "feature importance" d'un LGBMClassifier est une mesure de l'influence de chaque caractéristique sur la prédiction d'un modèle de machine learning. Plus cette valeur est grande, plus la feature influe sur le résultat de la prédiction. Cependant elle ne fournit pas d'informations sur l'influence relative des caractéristiques pour des cas individuels. Pour l'explication de ces derniers, on s'intéresse à l'interprétabilité locale.

Pour expliquer les décisions prises par un modèle de machine learning pour des cas individuels, on peut considérer les valeurs SHAP (pour "SHapley Additive exPlanation"). Cette méthode attribue à chaque variable d'une entrée une valeur qui indique son influence sur la décision prise par le modèle pour ce cas en particulier.

Elles prennent des valeurs positives lorsqu'elles influencent le modèle vers la prédiction de la classe positive, et des valeurs négatives sinon.

Une représentation graphique permet de visualiser très facilement les influences respectives des features les plus importantes dans la décision.

Pour ce faire, j'ai utilisé la possibilité offerte par le `LGBMClassifier` de calculer les valeur SHAP (SHAP values) via le positionnement du paramètre `predict_contrib` lors de l'appel à la fonction `predict_proba` du modèle.

Cependant, le package `SHAP` permet de calculer ces valeurs via un `explainer`, dans le cas où le modèle ne permet pas d'obtenir directement ces valeurs.

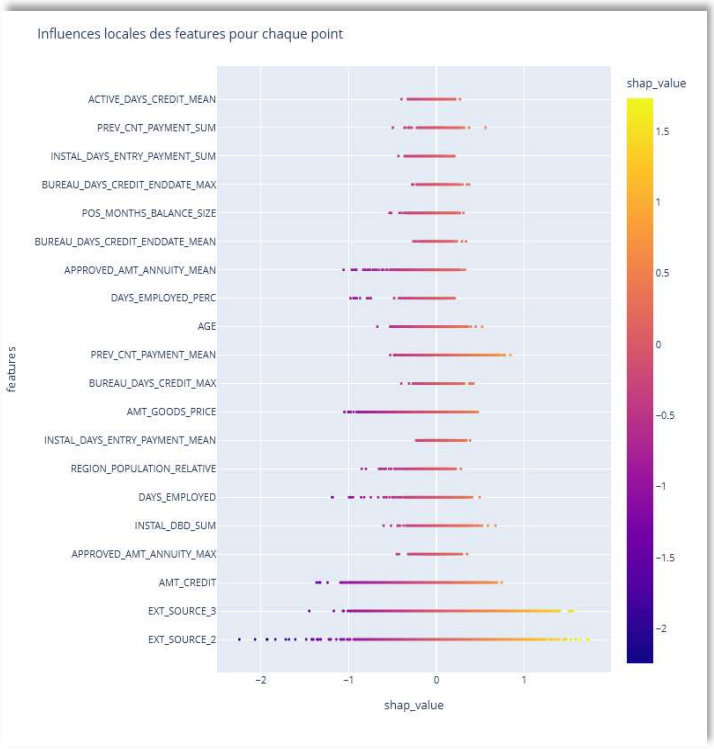

Figure 10 : Valeurs shap pour plusieurs entrées

Figure 11 : Valeurs shap pour un seul cas

Pour expliquer la décision pour une seule entrée, on utilisera un graphique de type waterfall, ordonné par les valeurs absolues des valeurs shap.

Limites et améliorations possibles

On peut constater que le résultat obtenu n'est pas très bon, puisqu'en utilisant ce modèle et le seuil de probabilité optimal, la banque ne gagne que 60% du montant total qu'elle pourrait gagner, et ceci pour deux raisons :

- On ne détecte pas tous les positifs (pénalité de -10)
- On exclut trop de négatifs (manque à gagner de 2)

Les améliorations possibles sont multiples :

- Une meilleure connaissance du métier permettrait d'effectuer un pré-traitement des données plus subtil, tant en termes d'imputation des valeurs manquantes – qui peut être propre à chaque variable - que de feature engineering. Une analyse descriptive des données d'apprentissage plus approfondie peut donner des pistes d'amélioration.
- Le choix du modèle et son optimisation ainsi que l'analyse approfondie de l'importance des features peut permettre d'améliorer le pré-traitement des données et d'écarter les features non pertinentes.
- L'optimisation du modèle pourrait aussi être plus fine en considérant une grille plus large pour les valeurs des hyperparamètres (cela nécessite du temps)
- D'une façon générale, la recherche du meilleur modèle devrait être faite par de multiples itérations sur les différentes étapes présentées dans ce document.