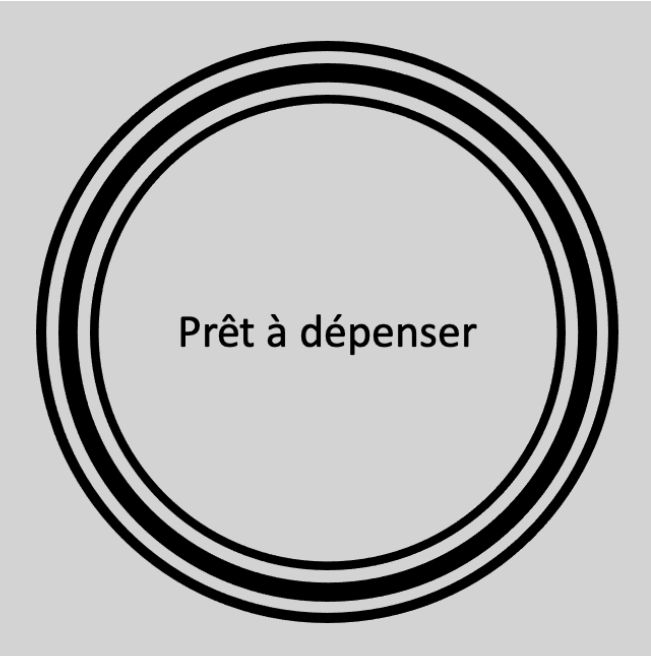
Implémentation d'un modèle de scoring

La société financière "Prêt à dépenser" propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt.



Afin de répondre aux demandes de transparence de ses clients vis-à-vis des décisions d'octroi de crédit, l'entreprise souhaite développer un dashboard interactif basé sur un modèle de scoring de la probabilité de défaut de paiement du client, réalisé à partir de sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.).

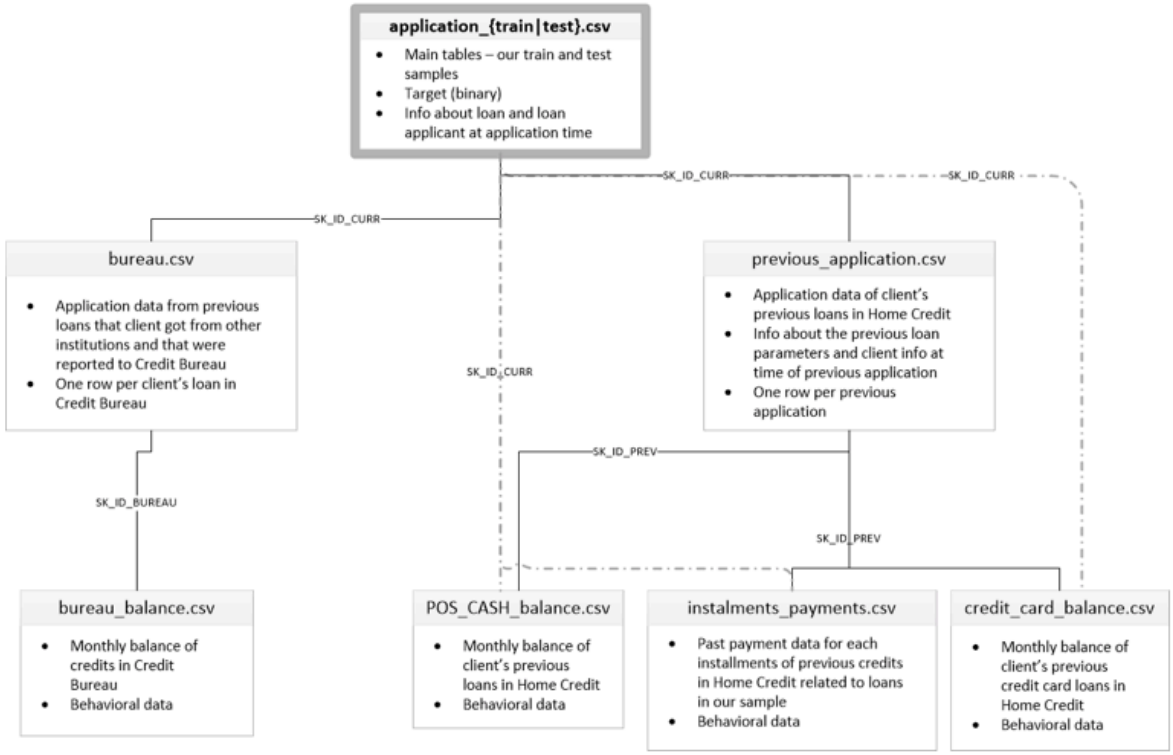
Ce document présente la démarche mise en œuvre pour développer ce modèle.

Présentation du jeu de données

Le jeu de données proposé pour l'élaboration du modèle est fourni par Home Credit et est disponible ici.

Les données se décomposent en 8 fichier(s) et proposent une variété d'informations anonymisées sur la situation personnelle et les activités bancaires des clients.

Les fichiers sont décrits et reliés comme suit :



La valeur à prédire, le risque de défaut du client, se situe dans la feature TARGET. Il prend la valeur 0 lorsque le client a remboursé son crédit, 1 s'il ne l'a pas fait. Il s'agira donc des 2 classes de notre problème.

Analyse exploratoire des données

Afin de se familiariser avec le jeu de données et de faire ressortir des caractéristiques pertinentes pour la modélisation, le jeu de données a été soumis à un processus d'analyse exploratoire en 3 parties :

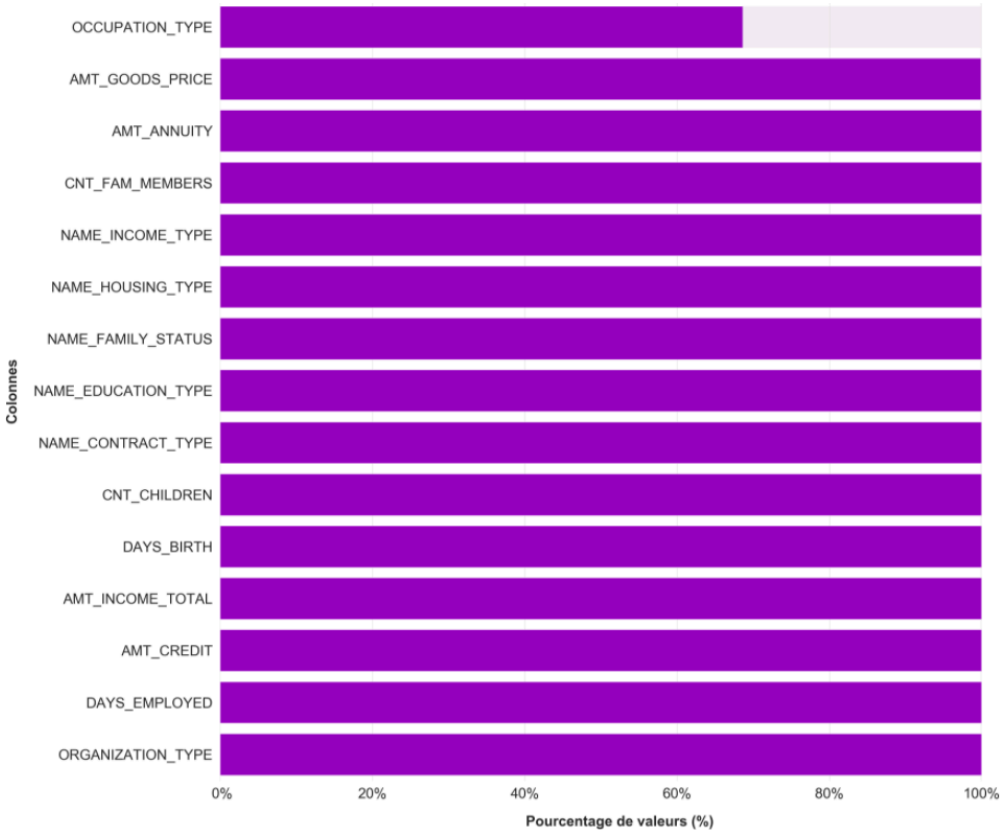
- Étude des valeurs manquantes
- Étude de la distribution des valeurs
- Étude des corrélations

Valeurs manquantes

33.61% des features contiennent plus de 50% de valeurs manquantes, soit plus d'un tiers des features.

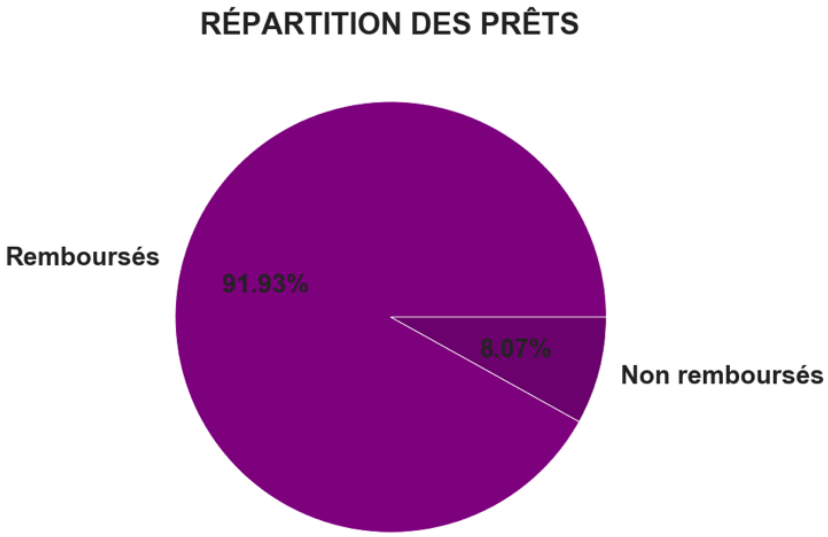
Cette information sera à prendre en compte lors de l'étape de modélisation. En effet, une étape d'imputation des valeurs pourra être nécessaire pour rendre le jeu de données exploitable par certains algorithmes.

Les colonnes qui d'un point de vue instinctif semblent les plus intéressantes semblent toutefois avoir un meilleur taux de remplissage :



Distribution des valeurs

Le risque de défaut
Dans notre jeu de données, une large majorité de prêts a été remboursée.



Il s'agit donc d'un problème présentant un déséquilibre de classes. Lors de l'étape de modélisation, il faudra donc réfléchir à la méthodologie à employer pour obtenir un modèle performant.

Les autres features
Un outlier ressort clairement du graphe présentant la distribution de la feature DAYS_EMPLOYED.

En effet, cette valeur représente le nombre de jours d'ancienneté du client au jour de la demande du prêt, en valeur négative.

Ici, on observe une valeur qui est non seulement positive, mais également d'une valeur absolue significative de 365000, soit 1000 ans d'ancienneté.

Si l'on observe cette valeur de plus près, deux points ressortent :

- cette valeur anormale concerne 18% des clients du jeu de données.
- le taux de défaut des clients concernés par cette anomalie est de 5.4% contre 8.66% pour les clients qui ne le sont pas.

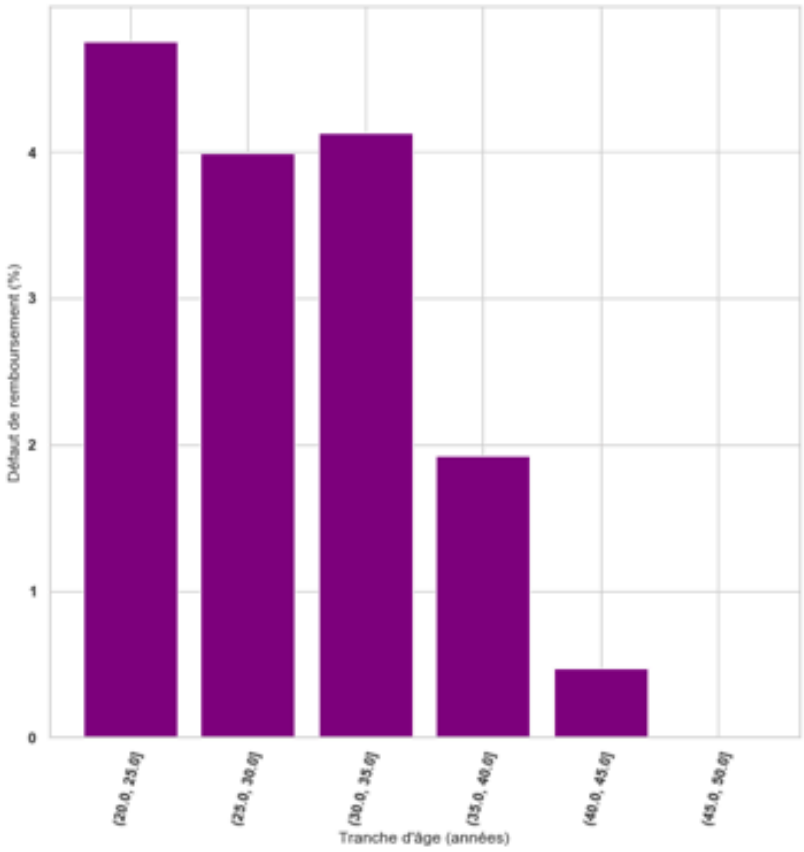
Cette anomalie pourrait donc avoir une signification et il sera intéressant de la retransmettre au modèle, tout en la traitant.

Pour l'étude des corrélations, cette valeur anormale sera simplement remplacée par NaN.

Étude des corrélations

Avec le risque de défaut
Le nombre de jours d'ancienneté et l'âge ont les corrélations positives les plus fortes avec le risque de défaut, tandis que les 3 indicateurs synthétiques présentent les corrélations négatives les plus fortes avec celui-ci.

- Focus ancienneté

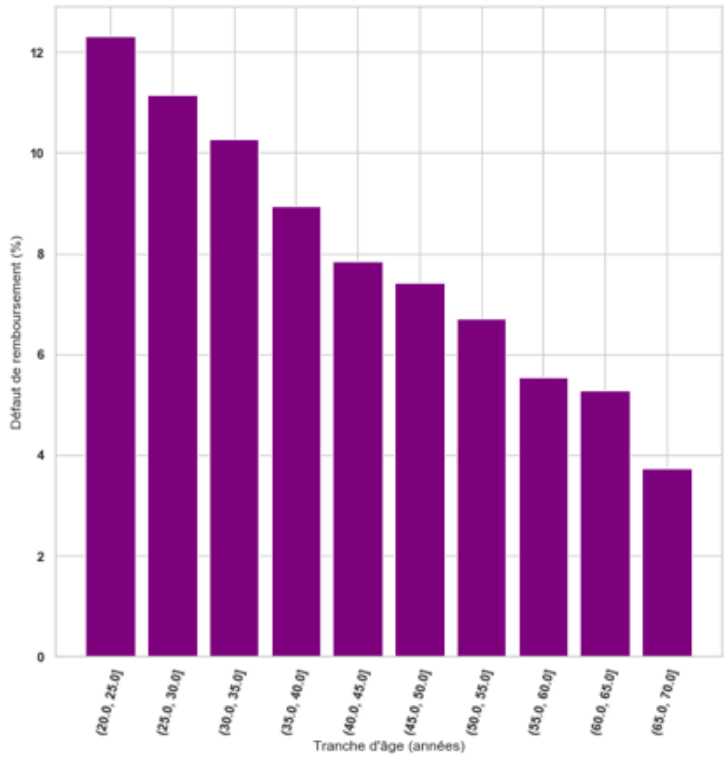


Le graphe ci-dessus montre :

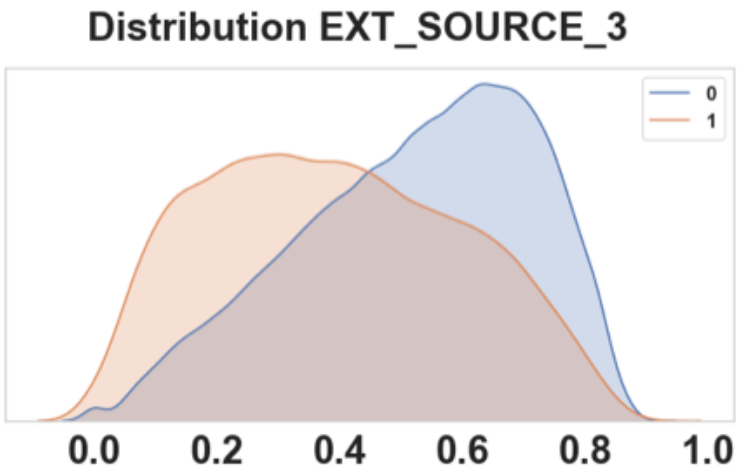
- qu'à partir de 35 ans d'ancienneté, le risque de défaut de paiement est divisé par plus de 2 par rapport aux employés ayant moins d'ancienneté.
- que le nombre de défauts de paiement semble décroître avec l'âge.

- Focus âge

L'histogramme par tranche d'âge confirme ces premières observations : le taux de défaut de remboursement est supérieur à 10% pour les tranches d'âges les plus basses, et inférieur à 5% pour les trois tranches d'âge les plus élevées.

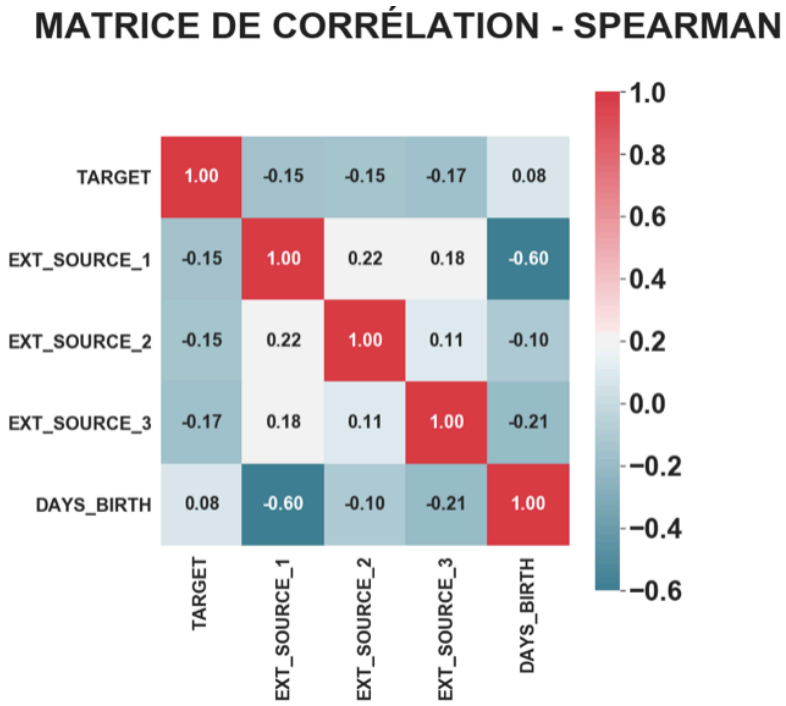


- Focus indicateurs synthétiques



La feature EXT_SOURCE_3 présente les variations les plus importantes dans sa distribution entre prêts remboursés / non-remboursés.

Entre les autres features



L'âge et l'indicateur synthétique EXT_SOURCE_1 semblent fortement corrélés.

Conclusion de l'EDA

L'analyse a pu mettre en lumière plusieurs informations importantes pour la modélisation à venir :

- Un tiers des features du jeu de données ont 50% de valeurs manquantes.
- Le nombre de jours d'emploi de certains clients présente la même valeur anormale
- des corrélations existent - bien que toutes globalement faibles - entre le risque de défaut et certaines features du jeu de données, en particulier l'âge, le nombre de jours d'ancienneté et les 3 indicateurs synthétiques.

Création de features & Assemblage des données

Le jeu de données comprend de nombreuses informations sous la forme de features qualitatives et quantitatives réparties dans 8 fichiers, dont la structure varie. En effet, suivant le fichier, une demande de crédit peut comporter une ou plusieurs observations.

Afin d'obtenir un jeu de données homogène et des features exploitables aussi riches que possible, de nouvelles features seront créées et des jointures effectuées. Le résultat sera ainsi un jeu de données unique, avec une ligne par client.

Création de nouvelles features

Bien que la démarche idéale consiste à demander à un expert métier quelles seraient les features qu'il serait pertinent de créer, certains ratios semblent intuitivement intéressants :

- apport initial / prix du bien
- montant annuel du crédit / revenu annuel du client
- ancienneté / âge

Une approche systématisée est ensuite appliquée pour obtenir différentes features statistiques (compte, moyenne, somme, minimum, maximum) à partir des features existantes.

On obtient ainsi un jeu de données unique, comportant une ligne par client.

Traitement des anomalies

L'analyse exploratoire a fait ressortir une anomalie identique pour 18% des clients : une ancienneté au travail de 365000 jours. Ces clients présentaient également un risque de défaut inférieur à la moyenne. Il semble donc que ces clients aient une particularité, qu'il pourrait être souhaitable de communiquer au modèle.

Pour cette raison, cette valeur est remplacée par NaN, mais une nouvelle feature est créée contenant la valeur "True" pour ces seuls clients.

Suppression des features non pertinentes pour la modélisation

Certaines features pourraient induire le modèle en erreur, comme l'ID du client. Cette feature est donc supprimée.

D'autres features pourraient, elles, contribuer à perpétuer des biais sexistes. Les colonnes ayant trait au sexe du client sont donc également supprimées.

Encodage des données

Afin d'être exploitable par différents algorithmes, les variables qualitatives doivent être transformées en variables quantitatives.

Pour ce faire, les variables qualitatives sont encodées suivant le principe suivant :
- label encoding pour les variables qualitatives comptant moins de 2 modalités
- one hot encoding pour les autres variables qualitatives

Conclusion

Le jeu de données final assemblé rassemble l'ensemble des informations contenues dans les 8 fichiers fournis, et comprend :

- 307511 observations
- 1774 features
- 1 observation / client

Méthodologie de rééquilibrage et choix de l'algorithme

L'analyse exploratoire a permis de mettre en lumière le déséquilibre de classes existant au sein du jeu de données. Il s'agit donc ici d'un problème de classification déséquilibré.

Ce type de problème présente des challenges uniques en termes de performances du modèle. Il conviendra ici de choisir une métrique pertinente pour juger réellement de la qualité du modèle. Afin de mettre précisément cette considération en lumière, prenons l'exemple de la métrique "accuracy". Dans le cas présent, 91% des observations ont une valeur de 0 pour le risque de défaut. Un modèle prédisant toujours "0", quelle que soit l'observation aurait donc une accuracy de 91%, pour un modèle qui serait en réalité inutilisable.

Afin d'assurer un modèle réellement performant, 2 approches seront utilisées :

- assurer un rééquilibrage des classes ou indiquer explicitement au modèle qu'il s'agit d'un problème déséquilibré
- créer une fonction de coût et une métrique permettant de tenir compte de la nature déséquilibrée du problème, ainsi que de la problématique métier de la banque, à savoir minimiser ses pertes.

Préparation des données

Sampling des données
Afin de minimiser les temps de calcul, un sampling de 100000 observations est réalisé.

Les algorithmes seront testés sur ce sample.

Séparation des données
Le jeu de données est ensuite séparé entre données d'entraînement et données de test, en veillant à maintenir les proportions des classes 0 et 1 dans nos nouveaux jeux de données.

Création fonction de coût et métrique

Dans le cadre du problème posé, les grandeurs suivantes peuvent être définies :

- TP = True Positive : la classe a été prédite comme "1", et la classe réelle est bien "1".
  => le modèle a prédit que le client ne rembourserait pas, et il a bien été en défaut.
- FP = False Positive : la classe a été prédite comme "1", mais la classe réelle est "0".
  => le modèle a prédit que le client ne rembourserait pas, mais il a bien remboursé son crédit.
- TN = True Negative : la classe a été prédite comme "0", et la classe réelle est bien "0".
  => le modèle a prédit que le client rembourserait, et il l'a bien fait.
- FN = False Negative : la classe a été prédite comme "0", mais la classe réelle est "1".
  => le modèle a prédit que le client rembourserait, mais il a été en défaut.

Il est important dans ce cadre de pénaliser les faux négatifs bien plus que les faux positifs.

En effet, dans le cas des faux positifs, la banque ne prête pas, et perd l'argent des intérêts que le client aurait remboursés.

Dans le cas des faux négatifs, la banque prête et perd la somme prêtée puisque le client ne rembourse pas.

Une fonction de coût et une score associé sont donc créés afin de refléter les points établis ci-dessus.

Le score ainsi créé est borné par les grandeurs suivantes :

- une valeur minimale à 0 : le modèle donne une prédiction à 0 (classe majoritaire) pour toutes les observations.
- une valeur maximale de 1 : le modèle donne une prédiction correcte pour toutes les observations.

Préprocessing

L'analyse exploratoire a permis de mettre en lumière :

- Un certain nombre de valeurs manquantes
- Le caractère déséquilibré du jeu de données : 91,68% des prêts ont été remboursés, contre seulement 8,33% de prêts non remboursés dans le jeu de données.

L'étape de préprocessing va donc comprendre 3 étapes :

- l'imputation des valeurs manquantes
- la mise à l'échelle des valeurs
- le rééquilibrage des classes

Imputation des valeurs manquantes

Certains algorithmes ne supportent pas les valeurs manquantes. Il convient donc de les remplacer.

Pour ce faire, l'ensemble des valeurs manquantes est remplacé par la valeur médiane de la feature via un SimpleImputer de la librairie Scikit-Learn.

Mise à l'échelle des valeurs

Afin de ne pas donner un poids supérieur à certaines features du fait de leur valeur absolue supérieure par rapport à d'autres, une mise à l'échelle est une étape incontournable. Elle est réalisée via un MinMaxScaler de la librairie Scikit-Learn qui ramène l'ensemble des valeurs entre 0 et 1.

Rééquilbrage des classes
3 approches sont possibles dans ce cas :

- Undersampling : supprimer des observations de la classe majoritaire afin de rééquilibrer le jeu de données.
- Oversampling : répéter des observations de la classe minoritaire afin de rééquilibrer le jeu de données.
- l'argument class_weight="balanced" : indiquer à l'algorithme le déséquilibre afin qu'il en tienne compte directement.

Ces 3 approches seront utilisées et les résultats obtenus comparés pour l'ensemble des algorithmes sélectionnés.

Les classes et librairies utilisées sont les suivantes :

- pour l'undersampling : RandomUnderSampler de la librairie Imbalanced-Learn
- pour l'oversampling : SMOTE de la librairie Imbalanced-Learn

Choix de l'algorithme

Explications de l'approche
Afin de modéliser au mieux le problème, les performances des trois algorithmes suivants vont être comparées :

- Régression Logistique via la classe LogisticRegression de la librairie Scikit-Learn
- RandomForestClassifier via la classe RandomForestClassifier de la librairie Scikit-Learn
- Light Gradient Boosting Machine via la classe LGBMClassifier de la librairie lightgbm

Leurs performances seront comparées à celles d'une baseline naïve, une instance de la classe DummyClassifier de la librairie Pandas, instantiée avec la stratégie "most_frequent", c'est-à-dire prédisant systématiquement la classe la plus fréquente.

Chacun de ces algorithmes est testé sur plusieurs jeux de données :

- le jeu de données sans rééquilibrage
- le jeu de données avec undersampling
- le jeu de données avec oversampling
- le jeu de données sans rééquilibrage, avec l'argument class_weight="balanced" fourni à l'algorithme.

Toutefois, l'algorithme DummyClassifier n'acceptant pas ce paramètre, le comportement a été simulé avec la stratégie "stratified" au lieu de "most_frequent", qui va générer des prédictions en respectant la distribution de classe du jeu de données d'entraînement.

Particularités de l'algorithme LGBMClassifier

- Ce classifieur peut travailler avec des valeurs manquantes et est insensibles aux variations d'échelle des features.

  2 variantes pour chaque jeu de données initial seront donc testées :

  - Le jeu de données avec imputation et mise à l'échelle des valeurs, sur lequel on appliquera les opérations d'under- et oversampling, pour obtenir 3 jeux de données. Ces 3 jeux seront utilisés par les algorithmes de régression logistique, de forêts aléatoires ainsi que par le dummy.
  - Le jeu de données sans aucune manipulation, sur lequel on appliquera uniquement l'undersampling, l'oversampling étant impossible pour un jeu de données comprenant des valeurs manquantes, pour obtenir 2 jeux de données. Ces 2 jeux de données seront utilisés uniquement par l'algorithme de Light Gradient Boosting Machine.

- Ce classifier peut recevoir une fonction de perte personnalisée, contrairement aux algorithmes issus de la librairie Scikit-Learn. La fonction de pertes donnée pour entraîner l'algorithme sera la fonction de coût personnalisée implémentée dans le cadre de cette modélisation (cf section Création fonction de coût et métrique)

Comparaison des résultats

ALGORITHME	STRATÉGIE DE RÉÉQUILIBRAGE	SCORE (ENTRAÎNEMENT)*	SCORE (TEST)
Dummy Classifier	None	0	0
	Undersampling	0.32	0
	Oversampling	0.32	0
	Balanced	-0.103	-0.105
Régression Logistique	None	0	0
	Undersampling	0.541	-0.393
	Oversampling	0.541	-0.197
	Balanced	-0.271	-0.246

Random Forest Classifier	None	0	0
	Undersampling	0.587	-0.072
	Oversampling	0.917	0.002
	Balanced	0	0
LGBM	None	0.023	0.025
	Undersampling	0.605	-0.095
	Oversampling	NA	NA
	Balanced	0.041	0.048

L'algorithme Light Gradient Boosting Machine avec l'argument "balanced" donne les meilleurs résultats. C'est donc l'algorithme et la stratégie de rééquilibrage que nous choisirons.

Pour terminer l'élaboration de notre modèle, deux étapes restent à effectuer :

- affiner la sélection des features : notre algorithme utilise actuellement plus de 1000 features, ce qui présente deux inconvénients :
  - des coûts en calcul élevés
  - certaines features ne contribuent peu ou pas au modèle, et peuvent même l'induire en erreur
- affiner les valeurs des hyperparamètres pour obtenir les meilleurs résultats possibles.

Sélection des features

Le but de cette étape est d'obtenir un modèle plus performant en réduisant le nombre de features.

Plusieurs approches seront testées et comparées :

- suppression des features colinéaires à plus de 90%
- suppression des features dont plus de 75% des données sont manquantes
- suppression des features ayant une importance nulle pour le modèle. L'opération est répétée jusqu'à ce qu'il n'y ait plus aucune feature dont l'importance soit nulle dans le jeu de données.
- suppression des features ayant moins de 95% d'importance pour le modèle.

Les performances seront comparées à une baseline, le score obtenu pour le jeu de données dans son intégralité.

Séparation des données

Les données sont séparées entre données d'entraînement et de test, en veillant à respecter la structure des classes via la fonction train_test_split de la librairie Scikit-Learn, avec l'argument "stratify" calqué sur la distribution de la feature à prédire.

Comparaison des performances des modèles suivant features

FEATURES			**SCORE**
Description		**Nb**	
Toutes les features		1774	0.108
Features de la ligne précédente – features corrélées	Coeff de corr de Pearson > 90%	0.106	0.106
Features de la ligne précédente – features ayant plus de 75% de features manquantes			0.106
Features de la ligne précédente – features ayant 0 importance pour le modèle		**131**	**0.109**
Features de la ligne précédente – features ayant moins de 90% importance pour le modèle			0.101

Les meilleurs résultats sont obtenus en supprimant :

- les features colinéaires
- les features présentant un pourcentage de valeurs manquantes supérieur à 75%
- les features ayant une importance nulle pour le modèle

C'est donc cette sélection de features que nous conserverons.

Optimisation des hyperparamètres : random search

Le but de cette dernière phase est de tenter d'optimiser le choix des valeurs des hyperparamètres afin d'obtenir de meilleurs performances.

La stratégie utilisée ici est celle d'une recherche randomisée en cross validation. Cette méthode a été prouvée être la plus efficace par rapport à GridSearchCV et la cross-validation seule.

Le modèle avec paramètres optimisé obtient un score de 0.131 sur le jeu de données test, soit une amelioration de 13% par rapport à la baseline consistant à prédire systématiquement que le client remboursera.

Modèle finalisé

Le modèle finalisé obtenu utilise :

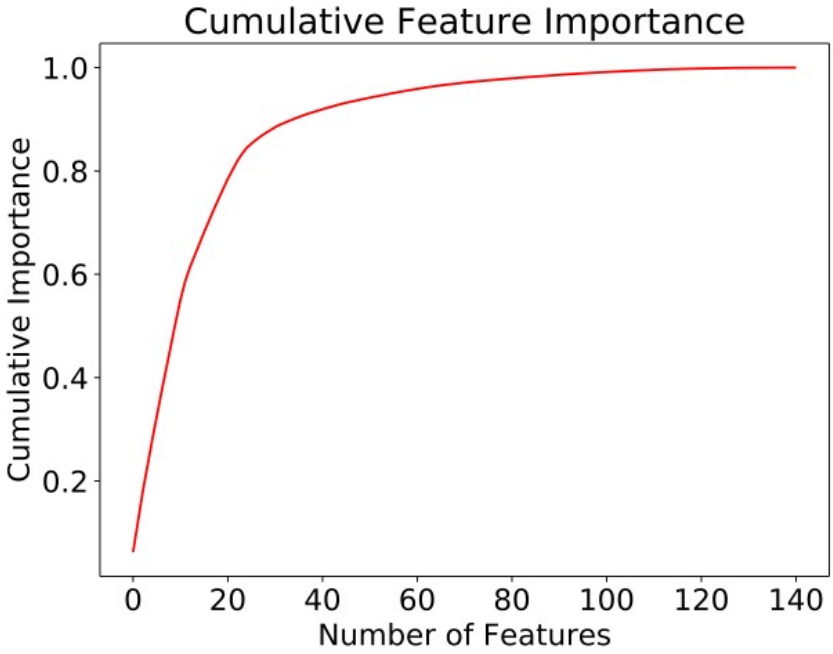
- 131 features (voir la liste complète des features en annexe)
- l'algorithme LGBMClassifier de la librairie lightgbm
- les hyperparamètres suivants :

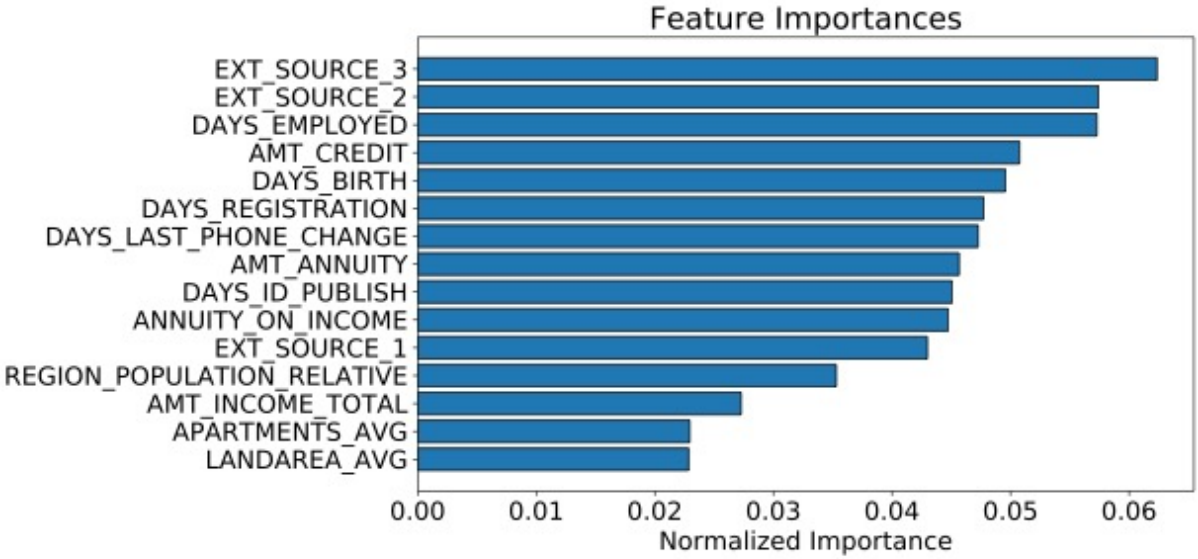
PARAMÈTRE	VALEUR OPTIMISÉE
n_estimators	10000
objective	Binary
class_weight	Balanced
boosting_type	dart
num_leaves	42
earning_rate	0.051284
subsample_for_bin	60000
min_child_samples	210
reg_alpha	0.285714
reg_lambda	0.081633
colsample_bytree	0.6
subsample	0.671717

Sur le jeu de données de test, le modèle donne un score de 0.13, soit une amélioration de 13% par rapport au meilleur score de la baseline consistant à prédire systématiquement la classe majoritaire.

Interprétabilité

Les features les plus importantes pour le modèle sont les suivantes, en cohérence avec les résultats de l'analyse exploratoire :





Conclusion

Nous disposions d'un jeu de données présentant un déséquilibre de classes, et demandant ainsi la mise en œuvre de stratégies spécifiques afin d'obtenir un modèle aussi performant que possible.

Notamment, une fonction de coût et métrique propres ont été créées afin de minimiser le risque pour la banque.

Après analyse des données, de nouvelles features ont été assemblées et plusieurs algorithmes et stratégies de rééquilibrage testés et comparés.

Le modèle final obtenu présente une amélioration de 13% par rapport à une baseline prédisant systématiquement le remboursement par le client. Parmi les features les plus importantes pour ce modèle, on retrouve l'ancienneté, le montant du crédit, l'âge du client, le montant de l'annuité, et le rapport annuité / revenu du client.

Le modèle pourrait potentiellement être amélioré en poursuivant l'optimisation des hyperparamètres, et/ou en faisant appel à un expert métier qui serait à même de mener l'élaboration de nouvelles features métier utiles.