

Traitement des données :

Le jeu de données d'entraînement (application_train.csv) fourni comporte deux classes identifiées par les labels 0 et 1 comportant respectivement 282 686 et 24 825 individus. La population de ces classes étant déséquilibrée, il n'est pas possible d'entraîner un modèle de machine learning sur ces données en raison du fort risque de sur-apprentissage. Cela signifie que le modèle aura principalement "appris" à reconnaître la classe majoritaire et n'aura donc que très peu appris de la classe minoritaire. Ce modèle ne sera donc pas généralisable sur les données réelles.

Afin de corriger ce déséquilibre, nous allons mettre en place deux approches différentes :

Classe minoritaire / Oversampling

Le jeu de données comportant des variables continues et des variables catégorielles, nous avons utilisé la méthode du SMOTE-NC pour générer les individus synthétiques. Le principe de cette méthode est de sélectionner un individu initial et l'un de ses k plus proches voisins (ici, $k=5$). L'individu créé par la méthode SMOTE-NC aura des caractéristiques situées "entre" ces points de référence, la distance étant déterminée par un coefficient sélectionné aléatoirement.

Nous avons généré 3443 individus, pour un total de 28268 individus dans la classe minoritaire (soit 10% de la classe majoritaire pour une augmentation d'environ 14% de la classe minoritaire), et autant dans la classe majoritaire. Le jeu de données final comporte donc 56536 individus.

Classe majoritaire / Undersampling :

L'étape suivante est de sélectionner les individus de la classe majoritaire. Cette étape est réalisée de façon aléatoire et nous permet d'obtenir le jeu de données final équilibré.

Le jeu de données initiales comportant des valeurs manquantes NaN, celles-ci ont été remplacées par des valeurs temporaires (-99999 pour les variables continues et "PLCHLDR" pour les variables catégorielles) puis transformées de nouveau en NaN une fois les classes rééquilibrées.

Entraînement du modèle

La première étape consiste à séparer notre jeu de données initiales en un jeu d'entraînement (X_{train}) et un jeu de test qui permettra d'évaluer la qualité du modèle (X_{test}). Ceci est réalisé à l'aide de la fonction `train_test_split` de scikit-learn.

Au départ, nous avons donc 56536 individus donc 80% (soit 45228 individus) seront aléatoirement sélectionnés pour constituer le jeu de données `X_train` et 20% (soit 11308 individus) constituerons `X_test`.

Nous avons ensuite sélectionné plusieurs types de modèles à tester afin de choisir celui qui sera le plus adapté au jeu de données:

- Modèles de régression :
 - Régression linéaire
 - Elastic net
- Modèles de classification :
 - `K_neighbors`
 - Régression logistique
 - SGD (Stochastic Gradient Descent)
 - Random Forest

Pour chacun de ces modèles, nous avons mis en place une grille d'hyperparamètres, c'est à dire les paramètres "internes" au modèles et qui permettent d'adapter son comportement aux données en entrée. Cette grille est composée de listes représentant les différentes valeurs possibles pour chaque hyperparamètre. Par exemple, pour un modèle `RandomForest`, nous avons utilisé les paramètres suivants:

n_estimators	max_features	min_samples_split
1	'sqrt'	50
100	'log2'	100
150	None	200

Ensuite, à l'aide de la fonction `RandomizedSearchCV` du package `scikit-learn`, nous effectuons une recherche aléatoire des hyperparamètres: Plutôt que de tester toutes les combinaisons possibles, des combinaisons aléatoires sont sélectionnées et testées, puis évaluées selon la métrique spécifiée (ici, le score métier).

Pour cet entraînement, nous utilisons une méthode de validation croisée en 'k-folds'. Cette méthode divise les données en K sous-ensembles de taille égale. La valeur de K a ici été définie à 5, chaque sous-ensemble comportera donc 1/5e des données (soit environ 9045 individus par 'fold' pour nos 45228 individus de départ dans le jeu de données `X_train`).

A chaque itération, un de ces sous-ensemble sera utilisé comme jeu de validation tandis que le modèle sera entraîné sur les 4 autres sous-ensembles. Une fois le

modèle entraîné et validé sur le 'fold' exclu, les performances du modèle sont mesurées.

A la fin de l'étape, les mesures finales comme la moyenne des scores de validation croisées sont collectées pour comparaison entre les modèles.

Optimisation et évaluation du modèle

Afin d'évaluer la performance du modèle en fonction de la problématique, nous avons mis en place une fonction de score adaptée se basant sur la matrice de confusion du modèle. La matrice de confusion permet de comparer les vraies étiquettes / classes des données à celles prédites, mettant ainsi en évidence les vrais positifs, les faux positifs, les vrais négatifs et les faux négatifs.

Dans notre cas, un faux négatif, un individu "mauvais client" prédit comme un bon client et à qui le crédit serait accordé (perte financière) est le scénario le plus dangereux. Les faux positifs (bon client prédit mauvais donc manque à gagner mais sans perte directe) sont également peu désirables.

A partir de la matrice de confusion, on établira donc un score qui attribuera un poids plus important aux faux négatifs, et défini comme suit :

$$\text{Score} = (10 \times \text{FN} + \text{FP}) / \text{total}$$

avec FN, le nombre de faux négatifs et FP le nombre de faux positifs. Ce score peut prendre des valeurs entre 0 (si tout les individus sont bien classés) et 10 dans le cas hypothétique où on n'aurait que des faux négatifs. Il sera utilisé pendant l'entraînement du modèle comme métrique à optimiser. C'est à dire que l'on cherchera à avoir le score le plus faible possible.

L'algorithme d'optimisation du modèle donc la sélection des hyper-paramètres optimaux avec la validation croisée, comme décrite dans la partie **entraînement du modèle**, ainsi que la minimisation du score métier défini ici.

En plus du score métier, nous évaluerons les modèles de classification entraînés avec la métrique appelée AUC (Area Under Curve). l'AUC est une mesure de performance courante mesurant la capacité globale du modèle à classer correctement les individus. Cette métrique varie entre 0.5 (classement aléatoire) et 1 (classement parfait).

Pour les modèles de régression, nous utiliserons la métrique RMSE (Root Mean Square Error) qui mesure l'écart / l'erreur moyenne entre les prédictions par rapport aux valeurs réelles. Plus cette métrique est faible, plus le modèle de régression est précis.

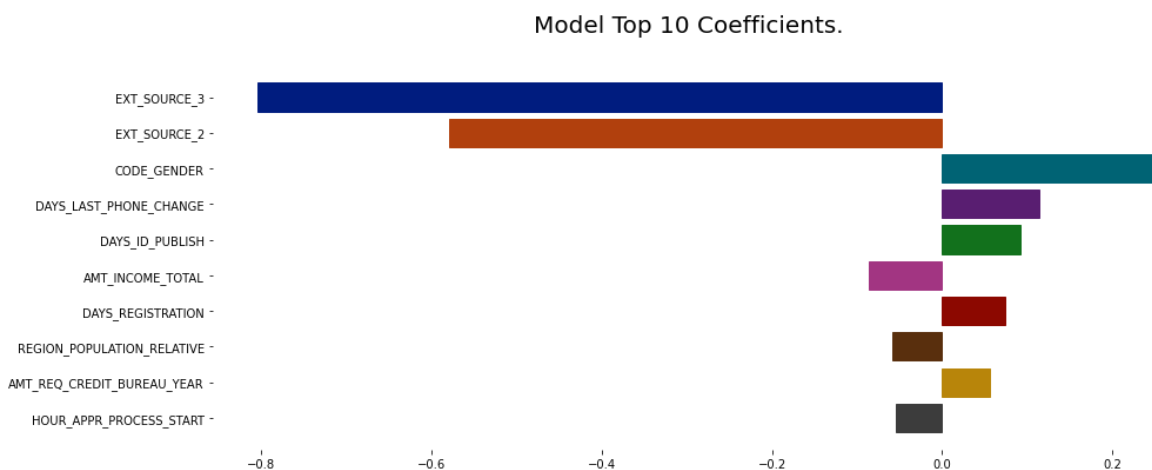
Résultats

	Durée d'entraînement	AUC	Score moyen	RMSE	Score métier
Régression - Régression linéaire	4.6s	-	-3.526	0.625	3.573
Régression - ElasticNet	7.3s	-	-4.725	0.693	4.751
Classification - Random Forest	7.9min	0.728	-1.561	-	1.575
Classification - Régression logistique	12.0s	0.686	-1.731	-	1.738
Classification - K-Neighbors	3.1min	0.699	-1.665	-	1.658
Classification - SGD	13.3s	0.67	-1.713	-	2.238

Les résultats montrent que les modèles de régression testés ont de performances assez mauvaises tant au niveau des scores métiers, plutôt élevés, que des RMSE indiquant une erreur importante, supérieure à ce qu'une prédiction aléatoire donnerait. Etant donné la nature du problème (Séparation des clients entre bons et mauvais clients), il semble normal que les modèles de classification soient plus efficaces.

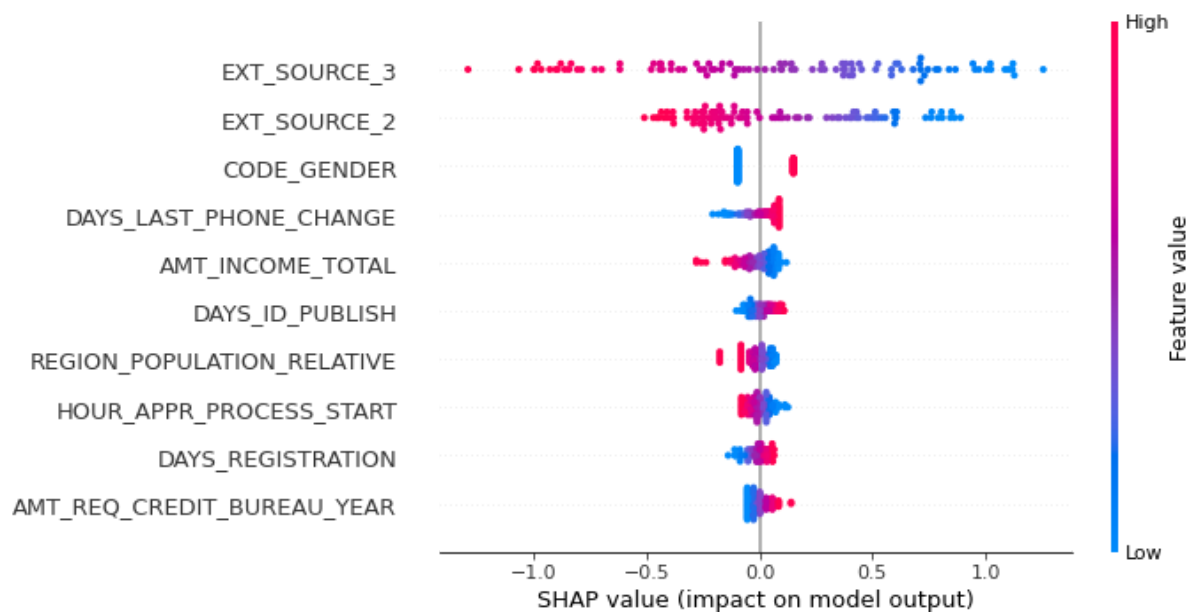
Parmi ceux-ci, le plus efficace est le Random Forest, avec un score métier inférieur à 1.6 et une AUC supérieure à 0.7. Ce modèle est efficace mais son entraînement est le plus long (près de 8 minutes). On préférera donc sélectionner un modèle parmi les trois restants : K-neighbors, régression logistique et SGD. Ces trois modèles ont de bonnes performances. On sélectionne le modèle de régression logistique en raison de sa rapidité supérieure au K-neighbor et son score métier meilleur que celui du SGD.

La figure suivante présente l'importance des différentes features pour le modèle. Les variables sont classées par ordre d'importance.



Nous avons ensuite utilisé le package Shap pour étudier l'impact des features sur les résultats de la prédiction. Les features sont classées en fonction de leur importance dans le modèle.





















La figure suivante montre, pour ces features, quel impact ont les valeurs de ces features sur les résultats du modèle. Ainsi, des valeurs élevées pour les features EXT_SOUCE_3 et EXT_SOURCE_2 ont un impact négatif important sur la prédiction (client moins susceptible d'être un mauvais client pour des valeurs élevées). Les variables AMT_INCOME_TOTAL, REGION_POPULATION_RELATIVE, HOUR_APPR_PROCESS_START ont également un impact négatif sur la prédiction pour des valeurs plus élevées, mais de façon plus modérée étant donné l'importance plus faible de ces variables. Les variables DAYS_LAST_PHONE_CHANGE, DAYS_ID_PUBLISH, DAYS_REGISTRATION et AMT_REQ_CREDIT_BUREAU_YEAR ont, à l'inverse, un impact positif sur les résultats du modèle pour les valeurs plus élevées (Client plus susceptible d'être un mauvais client).



Analyse du data drift

Le phénomène de data drifting correspond à un éloignement des données réelles par rapport aux données d'entraînement, ce qui créerait des imprécisions dans les prédictions du modèle. Par exemple, en cas de période de crise économique mondiale, l'influence des revenus d'une personne sur la prédiction pourrait varier par rapport à un modèle mis en place avant cette période donnée. Il faudra donc maintenir régulièrement le modèle et l'actualiser régulièrement.

L'analyse du data drifting sur un jeu de données antérieur aux données d'entraînement montre une déviation des nouvelles données comparées aux données de départ. Sur les 10 variables sélectionnées pour le modèle, 6 ont dévié (60% des variables) comme présenté sur la figure suivante.

Column	Type	Reference Distribution	Current Distribution	Data Drift
DAYS_LAST_PHONE_CHANGE	num			Detected
EXT_SOURCE_2	num			Detected
EXT_SOURCE_3	num			Detected
AMT_INCOME_TOTAL	num			Detected
DAYS_ID_PUBLISH	num			Detected
AMT_REQ_CREDIT_BUREAU_YEAR	num			Detected
REGION_POPULATION_RELATIVE	num			Not Detected
DAYS_REGISTRATION	num			Not Detected
CODE_GENDER	num			Not Detected
HOURL_APPR_PROCESS_START	num			Not Detected

On peut conclure de ces résultats qu'il faudra prêter attention aux performances du modèle au cours du temps et effectuer des maintenances périodiques pour actualiser le modèle face aux nouvelles données.

Limites et améliorations

Le modèle permet de déterminer avec une précision correcte la capacité de remboursement de clients (AUC ~ 0.7) et un score-métier minimisant le risque de faux négatifs (~ 1.7).

Bien que le modèle soit relativement performant, il est néanmoins nécessaire de surveiller l'évolution des prédictions au cours du temps: L'analyse effectuée sur un deuxième jeu de données montre un risque important de *data drifting*.

Concernant la méthode de scoring, on pourrait également prendre en compte le montant du prêt demander pour pondérer le risque impliqué par un faux positif : Un prêt d'un petit montant est à la fois plus simple à rembourser et moins risqué à accorder car la perte subie par l'entreprise serait plus faible.

En ce qui concerne la sélection des features finales, il serait également possible de sélectionner des features présentant une variance plus élevée et / ou de faire une pré-sélections des variables à l'aide d'une PCA.

Concernant l'optimisation du modèle, les hyperparamètres ayant été sélectionnés à partir d'une grille de paramètres fixés au préalable, il sera également possible d'ajouter une étape d'optimisation de ces hyperparamètres. Pour ce faire, une étape supplémentaire de validation croisée en faisant varier la valeur des hyperparamètres autour des valeurs sélectionnées sur la grille peut être considérée.

En l'état actuel, le dashboard représente les données des clients testés en regard des deux populations de clients ayant servis à entraîner le modèle (clients ayant remboursé le prêt et ceux en défaut de paiement. Sur une partie des clients, cela permet de bien visualiser dans quelle catégorie le client évalué se trouve (par exemple sur la variable `Ext_source_3` où les deux populations sont plutôt bien séparées), mais cela ne permet pas de comprendre clairement des cas plus "ambigus". Ce dashboard pourra être amélioré en ajoutant la visualisation de la feature importance locale de l'individu testé, et de détailler dans quelle direction les différentes variables ont orienté le résultat.