Note Méthodologique

RONDEAU Eva

Projet 7 : Implémentez un modèle de scoring

OPENCLASSROOMS

Table des matières

1. Mission

Cette note méthodologique présente la démarche de modélisation de manière synthétique, dans le cadre du projet « Implémentez un modèle de scoring » du parcours Data Scientist d'OpenClassRooms.

En tant que Data Scientist au sein de l'entreprise "Prêt à dépenser", proposant des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt, l'idée était de mettre en œuvre un outil de scoring crédit pour calculer la probabilité qu'un client rembourse son crédit, et ainsi, de classifier la demande en crédit *accordé* ou *refusé*.
Puis, la création d'un dashboard interactif a permis aux clients un accès à leurs données personnelles.

Les données sont constituées de 307 511 identifiants clients avec 121 caractéristiques.

2. Méthodologie d'entrainement du modèle

a) Analyse exploratoire et nettoyage des données

Chacun des fichiers .csv a été exploré, les features contenant plus de 40% de valeurs manquantes ont été écartées du jeu de données. Certaines variables contenant des valeurs aberrantes ont également été traitées :

- Valeurs infinies remplacées par des NaN
- Variable « DAYS_EMPLOYED » dont 18% des clients > 0, remplacés par NaN
- Variables des valeurs aberrantes (valeur = 365 243), remplacées par NaN
- Autres valeurs aberrantes pour certaines variables, supprimées

Le traitement des valeurs manquantes s'est effectué de plusieurs façons différentes selon le type de variable.

Tab. 1. Traitement des valeurs manquantes selon différentes conditions

	Variable catégorielle	Variable numérique
> 10%	Catégorie « Unknown »	Médiane
< 10%	Valeur la plus fréquente	Valeurs la plus fréquente
> 10% + corrélées	-	IterativeImputer

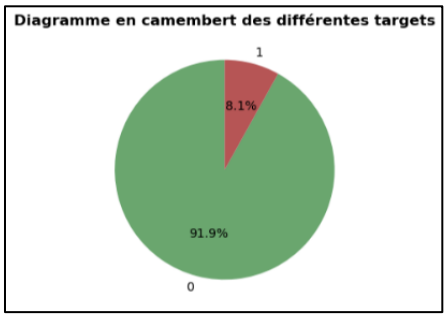

Figure 1 : Diagramme en camembert de la distribution de la target

Après exploration des données, il est observé que la variable cible est distribuée de façon très déséquilibrée au sein de nos données.

Le diagramme en camembert (figure 1) présente bien ce déséquilibre de la target.
Ainsi, 8% des clients ont des défauts de paiement (target 1) et 92% des clients n'ont pas de défauts de paiements.
Nous verrons par la suite comment de déséquilibre de classes est géré.

b) Feature engineering et fusion des donnés

Après le nettoyage des données, le feature engineering s'est déroulé en se basant sur le kernel ayant remporté la compétition Kaggle : https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features et en s'inspirant d'un autre kernel : LighGBM with Selected Features | Kaggle.
La plupart des variables sont créées en appliquant des fonctions statistiques telles que la valeur minimum, maximum, la moyenne, la somme et/ou la variance sur des tables groupées. D'autres variables simples, exprimées en pourcentages, ont été créées à partir de la table principale « application_clean.csv ».
Les variables catégorielles ont été encodées à l'aide d'une fonction « one_hot_encoder » avec get_dummies().

Après feature engineering et fusion des données selon l'identifiant de prêt client, notre dataframe est constitué de 305 900 clients et 567 features.

c) Sélection des variables importantes

Etant donné le grand nombre de variables, l'idée est de sélectionner les variables les plus importantes pour la prédiction du modèle, réduire la complexité et ainsi améliorer la performance du modèle. Pour cela, le kernel ayant remporté la compétition a permis d'effectuer cette sélection en utilisant un modèle LightGBM avec une validation croisée.

d) Modélisation et gestion du déséquilibre des classes

Une fois les 20 variables les plus importantes sélectionnées, les données ont été séparées en données de test (20%) et d'entrainement (80%).
Différents modèles de classification ont été comparés en utilisant différentes métriques permettant leur évaluation.

Les modèles de classification testés sont les suivants :

- Dummy Classifier (baseline)
- Logistic Regression
- Decision Tree Classifier
- LightGBM Classifier

Chaque modèle a été testé selon une méthode de déséquilibre des classes. Nous avons choisi de tester 3 méthodes différentes qui sont les suivantes :

- **SMOTE** (Synthetic Minority Over-Sampling TEchnique) : méthode de sur-échantillonnage (over-sampling) permettant de générer artificiellement des échantillons appartenant à la classe minoritaire (dans notre cas : target 1).
- **RandomUnderSampler** : méthode de sous-échantillonnage (under-sampling) de la classe majoritaire (dans notre cas : target 0) provenant de la bibliothèque imblearn, en prélevant au hasard des échantillons.
- **Class-weight** : paramètre accordant un poids plus important à la classe minoritaire.

Pour chaque modèle, un pipeline imblearn, qui est une bibliothèque permettant de traiter les problèmes de déséquilibres de classes, a permis de combiner plusieurs étapes de pré-traitement, à savoir, la standardisation des données, l'échantillonnage et l'entrainement d'un modèle de classification automatique afin d'améliorer la performance des modèles.

Une recherche des meilleurs hyperparamètres a été effectué par la méthode de recherche aléatoire **RandomizedSearchCV** qui est une technique d'optimisation des hyperparamètres moins coûteuse en temps de calcul que GridSearchCV.

3. Fonction coût métier, algorithme d'optimisation et métrique d'évaluation

a) Fonction coût métier

L'objectif pour la banque est de trouver les clients ayant des **défauts de paiements** (évènement positif). L'idée étant de **minimiser le nombre de faux négatifs** (clients prédits sans défauts de paiements mais ayant réellement des défauts de paiements) et **maximiser le nombre de vrais positifs** (clients prédits avec des défauts de paiements et ayant réellement des défauts de paiements).

Effectivement, la banque effectue un déficit si elle accorde un prêt à des clients n'étant pas en mesure de le rembourser. Néanmoins, elle risque la perte de clients dans le cas où elle refuse un prêt pour un client en mesure de le rembourser. Il serait certainement moins dramatique d'avoir de faux positifs supplémentaires pour réduire le nombre de faux négatifs. Mais, on veut tout de même éviter d'avoir un trop grand nombre de faux positifs.

L'idée est de créer un score pondérant les différents cas possibles (normalisé entre 0 et 1).

Tab. 2. Matrice de confusion

	Négatif (0)	**Positif (1)**
Négatif (0)	TN	FP
Positif (1)	FN	TP

Voici comment le score est calculé :

- cout_total = FN * cout_FN + FP * cout_FP + TP * cout_FP + TN * cout_TN
- gain_max = (TN + FP) * cout_TN + (TP + FN) * cout_TP
- gain_min = (TN + FP) * cout_FP + (TP + FN) * cout_FN

Score = (cout_total - gain_min) / (gain_max - gain_min)

Le **score normalisé** final calculé représente une performance globale tenant compte des gains et des pertes potentielles pour l'entreprise. Il va quantifier l'efficacité du modèle à accorder un prêt en tenant comptes des différents cas possibles et de leurs conséquences pour l'entreprise. Si sa valeur est proche de 0 : perte maximale pour l'entreprise, si sa valeur est proche de 1 : perte minimale pour l'entreprise.

Les valeurs de coûts sont attribuées de façon arbitraire :

- cout_FN = -20
- cout_FP = -2
- cout_TP = 0
- cout_TN = 10

b) Métrique(s) d'évaluation

Les différentes métriques utilisées pour l'évaluation du modèle sont les suivantes :

- **Accuracy score** : ratio entre nombre de prédictions correctes et le nombre total de prédictions
- **Précision** : ratio entre vrais positifs et tous les positifs. Une précision élevée indique un faible taux de faux positifs.
- **Recall** (= sensibilité) : taux de vrais positifs, ratio entre vrais positifs et vrais positifs + faux négatifs. Un recall élevé indique un faible taux de faux négatifs.
- **f1_score** : mesure prenant en compte la précision et le recall. Moyenne harmonique de la précision et du recall. Un score proche de 1 représente une performance parfaite du modèle (bonne harmonique entre recall et précision).
- **AUC** : aire sous la courbe ROC. Elle évalue la capacité du modèle à discriminer entre les classes positives et négatives. Plus la valeur AUC est élevée, plus la courbe ROC s'approche du coin gauche du graphique.
- **TP** : vrais positifs
- **FN** : faux négatifs
- **Score normalisé**

c) Algorithme d'optimisation

Le choix du modèle optimal s'est basé sur plusieurs critères (figure 2). Dans un premier temps, le score AUC est celui qui a aidé au choix du modèle. Mais dans le cas où deux modèles présentent le même score AUC, étant donné que l'objectif est de minimiser le nombre de faux négatifs (FN) et maximiser les vrais positifs (TP), ces deux métriques sont également prises en compte dans la sélection du modèle optimal.

Modèle	AUC	Accuracy score	Precision score	Recall score	F1_score	TP	FN	Score	Temps
Logistic Regression (SMOTE + RUS)	0.728436	0.682004	0.153272	0.647677	0.247883	3206	1744	0.68	69.263529
LightGBM (RUS)	0.752286	0.694067	0.163184	0.673737	0.262733	3335	1615	0.69	128.906979
LightGBM (CW)	0.752567	0.706669	0.167077	0.658788	0.266552	3261	1689	0.70	252.506408
Logistic Regression (SMOTE)	0.728461	0.682086	0.153541	0.649091	0.248338	3213	1737	0.68	47.889681

Figure 2 : Regroupement des meilleurs scores sélectionnés pour chaque méthode de gestion du déséquilibre des classes

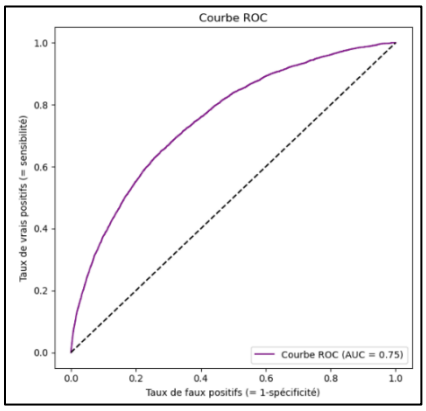

Figure 3 : Courbe ROC pour le modèle optimal LightGBM avec gestion du déséquilibre par RandomUnderSampling

Dans notre cas, nous avons choisi comme modèle optimal, le modèle LightGBM avec la gestion du déséquilibre des classes par RandomUnderSampling. Sa courbe ROC est la courbe affichée sur la figure 3.

Le modèle optimal a été choisi parmi les résultats stockés dans le tableau suivant :

Tab. 3. Synthèse des résultats obtenus pour les métriques d'évaluation des modèles de classification selon leur technique de gestion du déséquilibre des classes. Les flèches rouges représentent le meilleur modèle pour chaque technique de gestion des déséquilibres selon le score AUC

	Modèle	AUC	Accuracy	Precision	Recall	F1_score	TP	FN	Score_norm	Temps
	Dummy Classifier (baseline)	0.5	0.499	0.079	0.490	0.137	2426	2524	0.50	29.98
→	Logistic Regression (SMOTE)	0.728	0.682	0.153	0.649	0.248	3213	1737	0.68	47.89
	Decision Tree Classifier (SMOTE)	0.676	0.641	0.131	0.608	0.215	3009	1941	0.64	203.53
	LightGBM (SMOTE)	0.717	0.911	0.288	0.066	0.108	328	4622	0.87	430.41
	Logistic Regression (class_weight)	0.730	0.681	0.154	0.652	0.249	3228	1722	0.68	21.65
	Decision Tree Classifier (class_weight)	0.691	0.635	0.136	0.660	0.226	3267	1683	0.64	133.93
→	LightGBM (class_weight)	0.753	0.707	0.167	0.659	0.267	3261	1689	0.70	252.51
	Logistic Regression (RUS)	0.730	0.682	0.154	0.649	0.249	3213	1737	0.68	12.17
	Decision Tree Classifier (RUS)	0.684	0.653	0.136	0.615	0.223	3046	1904	0.65	32.84
→	LightGBM (RUS)	0.752	0.694	0.163	0.673	0.263	3335	1615	0.69	128.91
→	Logistic Regression (RUS + SMOTE)	0.728	0.682	0.153	0.648	0.248	3206	1744	0.68	69.26
	Decision Tree Classifier (RUS + SMOTE)	0.662	0.671	0.132	0.549	0.213	2719	2231	0.66	471.49
	LightGBM (RUS + SMOTE)	0.718	0.91	0.28	0.066	0.107	327	4623	0.87	3453.87

4. Interprétabilité globale et locale du modèle

a) Interprétabilité globale

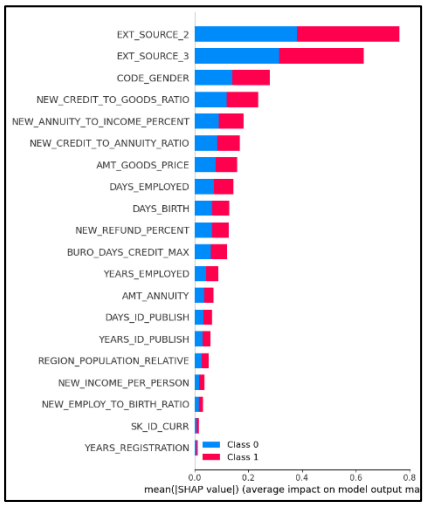

Figure 4 : Valeurs SHAP pour les features du modèle LightGBM (RUS)

Une fois le modèle optimal sélectionné, il est possible d'effectuer une analyse de la feature importance globale (figure 4) afin de mesurer l'importance de chaque caractéristique dans la prédiction du modèle. Pour cela, des valeurs SHAP (SHapley Additive exPlanations) sont calculées pour chaque caractéristique et chaque client.

Les barres bleues représentent la classe négative : caractéristique ayant une contribution négative à la prédiction de la classe 0.

Les barres rouges représentent la classe positive : contribution positive à la prédiction de la classe 1.

b) Interprétabilité locale

Pour le client spécifié, on observe le diagramme en cascade (figure 5) SHAP et la façon dont chaque caractéristique contribue à la prédiction du modèle pour ce client. On peut voir effectivement que pour la classe positive, les deux premières caractéristiques les plus importantes (EXT_SOURCE_2 et EXT_SOURCE_3) exercent une influence positive sur la prédiction de la classe.

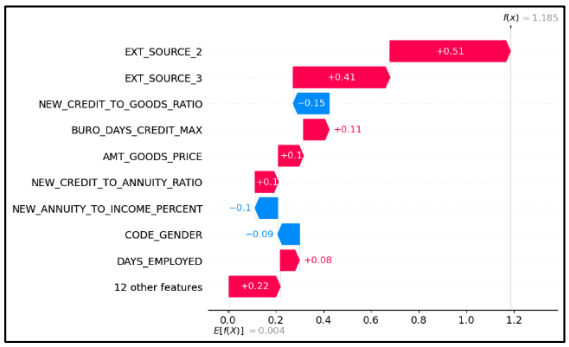

Figure 5 : Diagramme en cascade SHAP pour un client spécifié (pour la classe positive)

Des features exercent une influence négative sur la prédiction de la classe positive, c'est le cas pour (NEW_CREDIT_TO_GOODS_RATIO, NEW_ANNUITY_TO_INCOME_PERCENT, CODE_GENDER). Des valeurs élevées pour ces features diminuent la probabilité d'appartenir à la classe positive.

5. Limites et améliorations possibles

La première chose à noter, concernant les améliorations possibles, concerne la fonction coût métier pour laquelle les valeurs de coûts ont été choisies de façon arbitraire. L'idée serait, plutôt que de choisir arbitrairement les différentes valeurs selon le cas, de convenir avec plus de précision et selon un protocole des valeurs adaptées à chaque combinaison de classe prédite/classe réelle. Notre modèle ne peut donc qu'être amélioré.

En plus de cela, la création de features s'est essentiellement basée sur un projet. Si le temps l'avait permis, il aurait probablement été possible de créer d'autres features intéressantes pour plus de précision.

L'analyse exploratoire aurait pu être plus pointilleuse en effectuant une analyse au cas par cas pour le traitement des valeurs manquantes et des valeurs aberrantes.

Enfin, les données clients sont hébergées sur github, mais dans un cas concret, afin de garantir la protection des données et de respecter la conformité du RGPD, la protection des données serait à prendre en considération en cryptant les données et en limitant l'accès aux personnes autorisées.

6. Analyse du Data Drift

Le data drift ou dérive de données consiste en un décalage entre les données d'entrainement d'un modèle et les nouvelles données.

Dans notre cas, la librairie Evidently a permis de détecter la dérive de données en partant de l'hypothèse que le dataset d'entrainement « application_train.csv » représente les données de modélisation et le dataset de test « application_test.csv » représente les nouveaux clients une fois le modèle en production.

Les résultats ont été reportés dans le tableau HTML (figure 6) d'analyse qui montre une dérive de données détectée pour 10 features sur 120 (8,3%).

Dataset Drift

Dataset Drift is NOT detected. Dataset drift detection threshold is 0.5

120	10	0.0833
Columns	Drifted Columns	Share of Drifted Columns

Data Drift Summary

Drift is detected for 8.333% of columns (10 out of 120).

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
AMT_REQ_CREDIT_BUREAU_QRT	num			Detected	Wasserstein distance (normed)	0.500636
AMT_REQ_CREDIT_BUREAU_MON	num			Detected	Wasserstein distance (normed)	0.286321
AMT_GOODS_PRICE	num			Detected	Wasserstein distance (normed)	0.237215
AMT_CREDIT	num			Detected	Wasserstein distance (normed)	0.235083
NAME_CONTRACT_TYPE	cat			Detected	Jensen-Shannon distance	0.139496
AMT_REQ_CREDIT_BUREAU_WEEK	num			Detected	Wasserstein distance (normed)	0.138749
AMT_ANNUITY	num			Detected	Wasserstein distance (normed)	0.133096
DAYS_LAST_PHONE_CHANGE	num			Detected	Wasserstein distance (normed)	0.129984
FLAG_EMAIL	num			Detected	Jensen-Shannon distance	0.121881
AMT_INCOME_TOTAL	num			Detected	Wasserstein distance (normed)	0.100298

Figure 6 : Tableau HTML de la dérive de données