Présentation des modèles

DummyClassifier

Le modèle DummyClassifier est une méthode simple de classification souvent utilisée comme baseline pour évaluer la performance des modèles de classification plus complexes. Ce modèle attribue les prédictions de manière aléatoire en se basant sur la distribution des classes dans les données d'entraînement. Il est principalement utilisé pour comparer et évaluer d'autres modèles plus sophistiqués. En pratique, DummyClassifier peut prédire les classes de différentes manières : par la classe majoritaire, par une distribution uniforme, par stratification (en respectant la distribution des classes), ou encore de manière constante en choisissant une classe prédéfinie.

5 hyperparamètres ont été considérés :

1) most_frequent : Prédit la target la plus fréquente dans les données d'entraînement.
2) prior : Fait des prédictions en respectant la distribution de notre target.
3) uniform : Chaque classe (0 ou 1) a une probabilité égale d'être prédite.
4) stratified : Prédit un échantillon de manière probabiliste (probabilité des classes).
5) constant : La classe à prédire est fournie par l'utilisateur.

Logistic Regression (Régression logistique)

La régression logistique est un modèle de régression utilisé pour la classification binaire, bien qu'elle puisse être étendue à des problèmes de classification multiclasse. Contrairement à son nom, elle est utilisée principalement pour la classification plutôt que pour la régression. Ce modèle est basé sur la fonction logistique, qui modélise la probabilité qu'une observation appartienne à une classe particulière en fonction des variables explicatives. En entraînement, la modèle ajuste les poids des variables explicatives pour maximiser la probabilité d'observer les vraies classifications dans les données.

1) Cette probabilité possède la formule suivante :

$$p = \frac{e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_k x_k}}{1+e^{\beta_0+\beta_1 x_1+\beta_2 x_2+\ldots+\beta_k x_k}}$$

Figure 1: Probabilité de la Régression logistique

Avec x = Valeurs des features de l'échantillon.

Avec β = Coefficients à déterminer.

2) Ces coefficients β s'estiment en maximisant la vraisemblance.
3) L'hyperparamètre considéré est la régularisation L2 (Ridge).

LightGBM (Light Gradient Boosting Machine)

LGBM, est une bibliothèque populaire de boosting de gradient est utilisée principalement pour la classification et la régression dans les ensembles de données volumineux. LightGBM se distingue par sa capacité à entraîner des modèles rapidement tout en conservant une haute précision. Son approche repose sur le gradient boosting, une technique où les modèles de prédiction sont construits de manière itérative, chaque nouveau modèle corrigeant les erreurs des modèles précédents. LightGBM optimise également la croissance des arbres de décision de manière à maximiser l'efficacité du calcul pendant l'entraînement.

3 hyperparamètres ont été considérés :

1) **learning_rate :** Contrôle la contribution de chaque arbre.
2) **num_leaves :** Nombre maximum de feuilles.
3) **n_estimators :** Nombre d'arbres à entraîner.

Méthodologie d'entraînement du modèle

1. Imputation des données d'entraînement (ou non)

Pour le DummyClassifier et la régression logistique, il est important de prétraiter nos données en effectuant une imputation. Dans ce cas, nous avons choisi la stratégie d'imputation "mean" (moyenne). Cela signifie que les valeurs manquantes dans nos données seront remplacées par la moyenne des valeurs disponibles pour chaque variable. Cela assure que les modèles peuvent être correctement entraînés et testés sur des ensembles de données complets et cohérents.

Pour LightGBM, il est possible de travailler avec des données brutes contenant des valeurs manquantes (NaN). Cette approche offre l'avantage d'entraîner le modèle sur les données dans leur état initial, sans prétraitement préalable pour imputer les valeurs manquantes. Cela peut être bénéfique car le modèle peut potentiellement exploiter les informations présentes dans ces données incomplètes pour améliorer les prédictions.

2. Standardisation des données d'entraînement (StandardScaler)

Le StandardScaler transforme les caractéristiques de telle sorte à ce que leur moyenne soit nulle et que leur écart type soit égal à 1.

Standardiser les données avant d'entraîner nos modèles donne plusieurs avantages :

1) **Convergence plus rapide :** Pour des algorithmes basés sur la descente de gradient (comme la régression logistique), elle peut aider à accélérer la convergence.
2) **Évite la dominance des caractéristiques :** La mise à l'échelle garantit que toutes les caractéristiques ont le même poids initial.
3) **Importance des caractéristiques :** Si les caractéristiques sont mises à l'échelle, leurs poids peuvent être comparés plus facilement et directement.

3. Validation croisée (StratifiedKFold) – Nombre de folds = 5

Dans les problèmes de classification où il existe un déséquilibre important entre les classes cibles (comme c'est le cas dans ce projet), il est crucial que chaque fold lors de la validation croisée soit représentatif de l'ensemble complet des données. Pour garantir cela, les données ont été réorganisées de manière à ce que chaque fold conserve des proportions similaires d'échantillons par rapport à l'ensemble du dataset. Cette méthode est appelée stratification, et c'est pourquoi la méthode "StratifiedKFold" de scikit-learn a été choisie et utilisée.

4. Intégration de l'hyperparamètre seuil (threshold)

Les modèles de classification donnent une probabilité de prédiction par défaut avec un seuil de 0,5 pour décider de l'accord d'un crédit. Ajuster ce seuil permet d'optimiser le nombre de faux positifs et de faux négatifs, selon les besoins spécifiques du problème, en influençant la sensibilité ou la spécificité des prédictions.

5. Enregistrement des résultats avec MLflow

MLflow est une plateforme qui gère le cycle de nos modèles. Cela permet de suivre et de comparer les différents essais et modèles, facilitant la reproductibilité et le déploiement. Dans ce projet, MLflow enregistre les hyperparamètres, les sorties ainsi que les modèles sous forme de format pickle.

Le traitement du déséquilibre des classes

La target (0 = Aucun problème de paiement | 1 = Soucis de paiement) est très déséquilibrée.

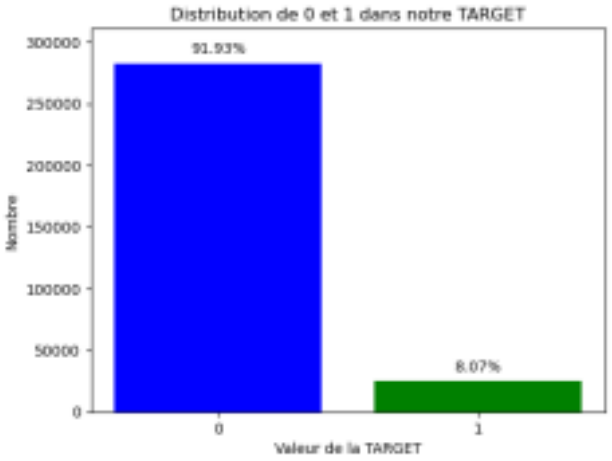


Figure 2: Déséquilibre de la distribution TARGET

Un tel déséquilibre soulève de nouvelles problématiques :

1) **Performance trompeuse :** Un modèle qui prédit toujours la classe majoritaire obtiendra une précision (accuracy) élevée malgré le fait qu'il ne soit pas vraiment efficace.

2) **Biais du modèle :** Car il est davantage "récompensé" pour avoir correctement prédit cette classe étant donnée sa prédominance.

Pour résoudre ces problématiques, nous pouvons utiliser la méthode "class_weights" incluse dans les modèles de Scikit-learn.

```
nb_0 = (df_classification_imputed['TARGET'] == 0).sum()
nb_1 = (df_classification_imputed['TARGET'] == 1).sum()
class_weights = {0: 1, 1: nb_0 / nb_1}
```

Figure 3: Code Python pour la méthode "class_weights"

Ce paramètre permet de donner un poids différent à différentes classes. En faisant cela, il est possible d'augmenter le coût des erreurs de classification sur la classe minoritaire, ce qui rend le modèle plus attentif à cette classe lors de l'apprentissage.

Ici, la classe 1 (minoritaire) a un poids qui est égale au ratio du nombre d'échantillons ayant la classe 0 par rapport à la classe 1.

Cette méthode offre une manière simple d'aborder le problème des classes déséquilibrées sans avoir besoin de techniques de sur-échantillonnage ou de sous-échantillonnage.

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

La fonction coût métier

L'objectif premier d'une banque lorsqu'elle octroie des crédits à ses clients est d'optimiser ses bénéfices. Cependant, deux scénarios peuvent entraîner des pertes financières :

- **Faux positifs (FP) :** Il s'agit de cas où la banque estime qu'un client ne sera pas capable de rembourser son prêt, alors qu'en réalité, il le pourrait.
- **Faux négatifs (FN) :** Ce sont les situations où la banque estime que le client remboursera le crédit, mais ce dernier finit par défaillir.

Il est intéressant de noter qu'un faux négatif est nettement plus coûteux pour la banque qu'un faux positif. Il peut être supposé, à titre illustratif, que le coût associé à un FN soit dix fois supérieur à celui d'un FP.

Ainsi, l'enjeu est de minimiser une fonction dite de "coût métier" formulée ci-dessous :

Coût métier = Somme(10*FN+FP)

L'algorithme d'optimisation

En apprentissage automatique, optimiser les hyperparamètres est essentiel pour maximiser les performances des modèles. L'une des méthodes avancées pour cette optimisation est l'arbre de Parzen (Tree-structured Parzen Estimator, TPE).

Le TPE utilise une approche bayésienne pour évaluer la probabilité que des combinaisons

spécifiques d'hyperparamètres conduisent à de bonnes performances, en se basant sur les essais précédents. Pour guider cette optimisation, une fonction "objective" est définie, que l'on cherche à minimiser ou maximiser.

À chaque essai, le TPE apprend et ajuste ses recommandations, en se concentrant davantage sur les zones de l'espace des hyperparamètres où les performances sont susceptibles d'être meilleures. Le package Optuna a été utilisé pour appliquer cette méthode d'optimisation de manière efficace.

La métrique d'évaluation

Comme mentionné dans la section 2 sur l'algorithme d'optimisation, l'arbre de Parzen vise à optimiser une fonction "objective" pour déterminer le meilleur ensemble d'hyperparamètres. Dans ce contexte, cette fonction objective représente le coût métier que nous cherchons à minimiser ou à maximiser.

Synthèse des résultats

Pour le LightGBM et la Régression Logistique, 50 essais à Optuna ont été accordés, lui permettant ainsi d'identifier de manière optimale les bons jeux d'hyperparamètres.

	Strategy	C	Learning Rate	Num Leaves	Threshold	AUC	Accuracy	Business Score
LightGBM	nan	nan	0.017	55	0.48	0.787	0.736	149513
LogisticRegression	nan	71.438	nan	nan	0.52	0.772	0.729	156247
DummyClassifier	most_frequent	nan	nan	nan	nan	0.5	0.919	248250

Figure 4: Résultats des 3 modèles

Il est souhaitable d'optimiser en priorité le coût métier (Business Score) mais il est toujours intéressant de regarder d'autres métriques pour évaluer de manière plus globale les performances de nos modèles :

- **Accuracy :** C'est le ratio du nombre total de prédictions correctes sur le nombre total de prédictions. Cette valeur peut être trompeuse surtout si les classes sont fortement déséquilibrées (comme c'est le cas ici).
- **AUC (Area Under the Curve) :** Elle mesure l'aire sous la courbe ROC (Receiver Operating Characteristic). La courbe ROC trace le taux de vrais positifs par rapport au taux de faux positifs à différents seuils de classification. Une AUC de 1 indique une classification parfaite, tandis qu'une AUC de 0,5 indique une performance équivalente à une prédiction aléatoire.

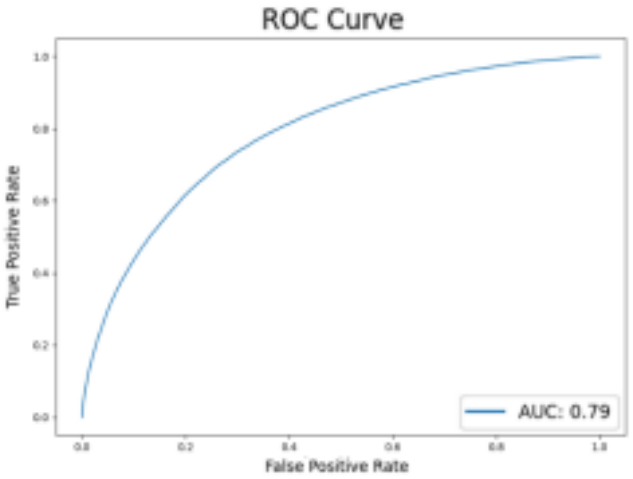


Figure 5: Courbe ROC du modèle LightGBM

La figure 4 montre qu'en effet, utiliser un seuil de 0,5 n'est pas forcément idéal car pour la Régression Logistique, utiliser un seuil de 0,52 optimise le Business score, le raisonnement est le même pour le LightGBM (0,48).

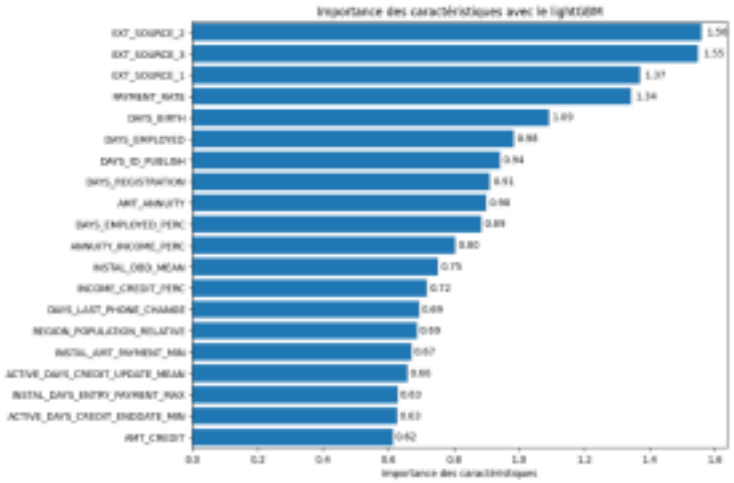
En conclusion, compte tenu de tous ces résultats, le modèle à utiliser en production est le suivant :

LightGBM (Num Leaves = 55 | Learning rate = 0,017 | Threshold = 0,48)

L’interprétabilité globale et locale du modèle

Interprétabilité globale

L'interprétabilité globale d'un modèle se réfère à notre capacité à comprendre comment le modèle prend ses décisions pour l'ensemble d’un jeu de données. Pour répondre à ces questions, l’importance des features a été utilisée. Cette métrique est intéressante pour détecter du data leakage (fuite de données). Le data leakage se produit lorsque des informations provenant de la variable cible s'infiltrent dans les caractéristiques utilisées pour former le modèle, ce dernier pourrait par conséquent afficher une performance artificiellement élevée.



Pour le LightGBM, il est constaté qu’aucune Feature est anormalement élevée relativement aux autres. Cela permet de conclure qu’il n’y a aucune fuite de données.

Figure 6: Feature importance du LightGBM

Interprétabilité locale

Contrairement à l'interprétabilité globale, qui cherche à comprendre comment un modèle fonctionne sur l'ensemble de ses données, l'interprétabilité locale cherche à comprendre pourquoi un modèle a fait une prédiction particulière pour un seul échantillon.

Par exemple, la méthode SHAP (SHapley Additive exPlanations) sera utilisée pour expliquer la probabilité donnée par le modèle. Cette méthode attribue une valeur a chaque fonctionnalité. Si cette valeur est positive, alors elle a tendance à augmenter la probabilité finale, et inversement.

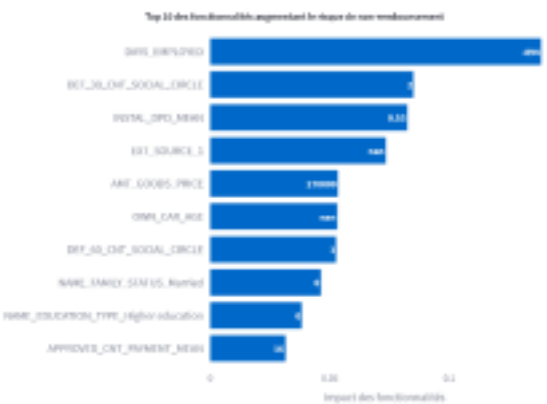


Figure 7: Exemple d'utilisation de SHAP

Les limites et les améliorations possibles

Utilisation d'un kernel Kaggle pour la création du dataset final

Pour accélérer ce processus, l'utilisation d'un kernel Kaggle peut être choisie, disponible à l'adresse suivante : (https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script)

Ce choix est intéressant car ce kernel en particulier donne des scores tout à fait satisfaisants (Public Score = 0.79070). Cependant, il serait toujours judicieux de vouloir améliorer ce kernel en ajoutant nos propres approches concernant l'analyse exploratoire, la préparation des données et le Feature engineering.

Le manque d'expertise

Bien que l'expertise en science des données soit axée sur la manipulation, l'analyse et la modélisation des données, les critères d'octroi de crédit présentent des spécificités complexes propres au domaine bancaire. Comprendre chaque feature était un défi majeur.

Par conséquent, la collaboration avec un expert du domaine bancaire aurait apporté une valeur ajoutée significative. Cet expert aurait pu clarifier la signification et la pertinence de chaque variable, ainsi qu'assurer l'adéquation des méthodes de traitement des données.

L’analyse du Data Drift

Le Data Drift fait référence à un changement dans les distributions de données au fil du temps par rapport aux données initiales utilisées pour former un modèle. Ce phénomène peut influencer les performances du modèle, car si les caractéristiques des données d'entrée évoluent significativement par rapport aux données d'entraînement, les prédictions du modèle peuvent devenir imprécises ou moins pertinentes.

L'hypothèse adoptée ici est que le jeu de données "application_train" représente les données utilisées pour la modélisation initiale, tandis que le jeu de données "application_test" représente de nouveaux clients ou des données observées une fois que le modèle est en production.

Pour évaluer le Data Drift, deux méthodes sont choisies :

- Pour les colonnes numériques, le test statistique de Kolmogorov-Smirnov sera utilisé pour comparer les distributions entre "application_train" et "application_test".

- Pour les colonnes catégorielles, l'indicateur PSI (Population Stability Index) sera utilisé pour évaluer les changements dans la distribution des catégories entre les deux ensembles de données.

Un seuil de 0,2 a été défini pour chaque test statistique, permettant de détecter des divergences significatives entre les distributions de données, ce qui pourrait indiquer un potentiel Data Drift nécessitant une réévaluation ou une mise à jour du modèle.

- Si la p-value du test K-S est inférieure à 0,2, cela suggère que les deux échantillons ne proviennent pas de la même distribution à un niveau de confiance de 80%.
- Si le PSI est supérieur à 0,2, cela suggère aussi une indication de Data Drift.

Le package evidently permet d’effectuer cette analyse de Data Drift et de créer une page html permettant l’analyse de ces résultats. Il est considéré que si plus de la moitié des fonctionnalités possèdent du Data Drift, alors le Data Drift sur l’ensemble des données est détecté.

Figure 8: Résultats de l'analyse du Data Drift

Il est noté que 58.8% des fonctionnalités possèdent du Data Drift. En conclusion, le Data Drift a eu lieu entre application_train et application_test.

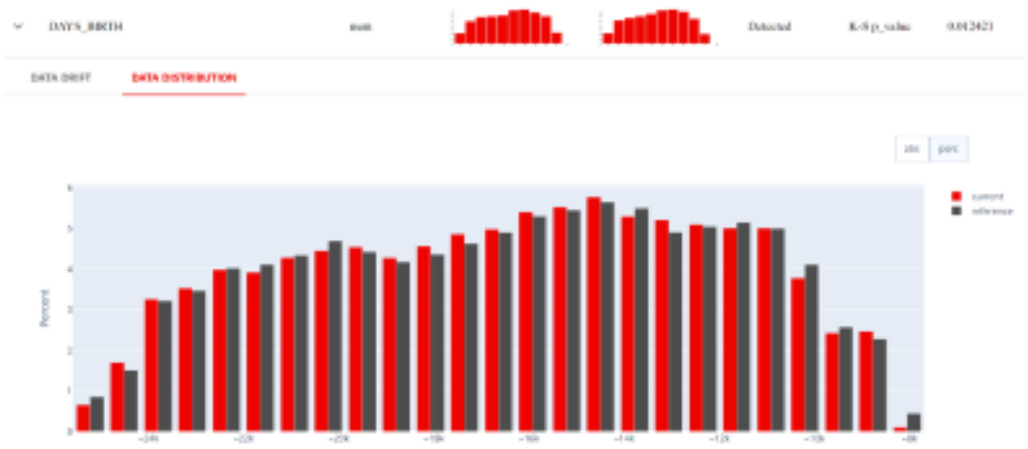


Figure 9: Exemple de Data Drift pour la feature DAYS_BIRTH