1. **Présentation des modèles**

* 1. **DummyClassifier**

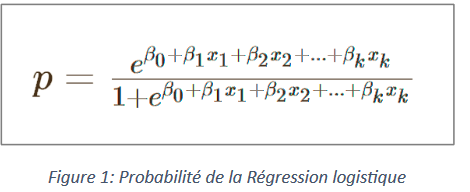
Il s’agit d’un classifieur dit "naïf" dans le sens où il ne tient pas compte des données d'entrée pour faire des prédictions. Il est souvent utilisé comme point de référence pour comparer les performances d'un modèle réel par rapport à un modèle qui prédit de manière simple.

**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.
   1. **Logistic Regression (Régression logistique)**

Il s’agit d’un classifieur linéaire utilisant une fonction logistique (ou fonction sigmoïde) pour serrer les prédictions dans un intervalle compris entre 0 et 1. Si cette probabilité est supérieure à un certain seuil, l'échantillon est classé dans la classe positive, sinon il est classé dans la classe négative.

1. **Cette probabilité possède la formule suivante :**



Avec x = Valeurs des features de l’échantillon.

Avec β = Coefficients à déterminer.

1. Ces coefficients β s’estiment en maximisant la vraisemblance.
2. L’hyperparamètre considéré est la régularisation L2 (Ridge).
   1. **LightGBM (Light Gradient Boosting Machine)**

Il s’agit d’un classifieur non-linéaire utilisant des arbres de décision. Il se sert d’une approche basée sur les feuilles plutôt que sur la profondeur pour faire croître ses arbres. Cette technique lui permet d’être plus rapide et de consommer moins de mémoire comparé à d’autres modèles utilisant des arbres de décision tels que XGBoost par exemple.

**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.

1. **Méthodologie d'entraînement du modèle**

L’analyse exploratoire, la préparation des données et le feature engineering nécessaires à l’élaboration du modèle de scoring sont principalement issus des kernels suivants :

* [*https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction*](https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction)
* [*https://www.kaggle.com/code/codename007/home-credit-complete-eda-feature-importance*](https://www.kaggle.com/code/codename007/home-credit-complete-eda-feature-importance)
* [*https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script*](https://www.kaggle.com/code/jsaguiar/lightgbm-with-simple-features/script)

Ils ont été adaptés et enrichis notamment sur la partie de préparation des données avec le retraitement des valeurs manquantes ou encore le regroupement de modalités.

Quatre méthodes de sélection de variables ont été utilisées :

* Suppréssion des variables à plus de 50% de valeurs manquantes
* Suppression des variables à haute corrélation 'High Correlation Filter'
* Suppréssion des variables à variance faible 'Low Variance Filtering'
* SelectKBest

**2.1 Sélection des données d’entraînement et de test**

En Machine Learning il ne faut jamais valider un modèle sur les données qui ont servi à son entraînement. Le modèle doit être testé sur des données qu'il n'a jamais vues. On aura ainsi une idée de sa performance future. Le dataset sera mélangé de façon aléatoire avant d'être divisé en deux parties :

* Un train set dont les données sont utilisées pour entrainer le modèle (80% des données)
* Un test set réservé uniquement à l'évaluation du modèle (20% des données)

La séparation du dataset en données d’entraînement et de test va permettre de détecter de l’overfitting (modèle trop complexe qui apprend parfaitement les données d’entraînement mais n’arrive pas à généraliser) ou de l’underfitting (modèle trop simple ou mal choisi).

**2.2 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.

Un faux positif n'a pas le même coût qu'un faux négatif. Ce dernier est beaucoup plus coûteux pour la banque. Nous avons supposé que cela coûtait 10 fois plus cher qu’un faux négatif.

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



Le seuil de probabilité a été défini de manière à détecter efficacement les clients défaillants. Nous souhaitons donc maximiser le nombre de vrais positifs (client défaillant prédit défaillant) mais sans trop augmenter le nombre de faux positifs. Le seuil de 0.52 permet d’avoir le score métier le plus faible

Les métriques d’évaluation prises en compte dans ce projet sont les suivantes :

* **AUC-ROC \_score :** Il mesure la capacité d'un modèle à classer correctement les exemples positifs par rapport aux exemples négatifs, quelle que soit la valeur du seuil de classification. L'AUC-ROC varie entre 0 et 1, où une valeur de 1 représente une performance parfaite et une valeur de 0,5 représente une performance aléatoire.
* **Accuracy\_score :** C’est un score d’exactitude calculé en divisant le nombre d’exemples correctement classés par le nombre total d’exemples. Une valeur de 1 indique que tous les exemples ont été classés correctement et une valeur de 0 indique une classification complètement incorrecte.
* **Recall\_score :** ou sensibilité. Il mesure la capacité d'un modèle à identifier correctement les exemples positifs (client à risque) parmi tous les exemples réellement positifs présents dans l'ensemble de données. Une valeur de 1 indique un rappel parfait, c'est-à-dire que tous les clients à risque ont été correctement identifiés, et une valeur de 0 indique que aucun client à risque n'a été correctement identifié.
* **Precision\_score :** Il mesure la capacité d'un modèle à classifier correctement les exemples positifs parmi tous les exemples prédits comme positifs. Une valeur de 1 indique une précision parfaite, c'est-à-dire que tous les clients prédits à risque sont corrects, et une valeur de 0 indique que aucun client prédit à risque n'est correct.
* **F1 score :** moyenne harmonique de la précision et du rappel, permet de focaliser sur la classe positive minoritaire

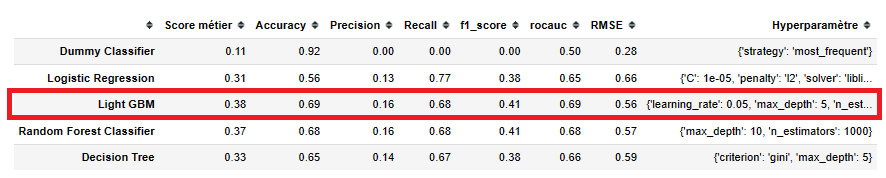
**2.3 Optimisation des modèles**

L’optimisation des hyperparamètres pour chaque modèle a été réalisée à l’aide de

GridSearchCV. Les hyperparamètres sélectionnés seront ceux qui permettent d’obtenir le meilleur score métier (max costum\_score) en utilisant la validation croisée avec 10 plis.

Ensuite, la valeur seuil moyenne obtenue à partir de GridSearchCV pour le meilleur modèle a également été récupérée. L’application de ce seuil sur la méthode predict\_proba permettra de classer tout nouveau client en tant que client fiable ou client à risque, et ainsi de décider d’accorder ou non le crédit.

**2. 4 Synthèse des résultats**

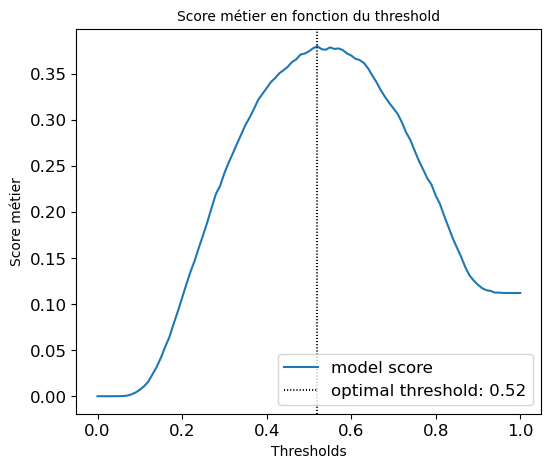


La comparaison des performances des modèles, comme illustré dans la figure ci-dessous, met en évidence les avantages de l’algorithme LightGBM. En effet, ce dernier a permis d’obtenir les meilleurs scores parmi tous les modèles évalués, tout en maintenant un temps de traitement raisonnable

**2.5 Intégration de l’hyperparamètre seuil (threshold)**

Les modèles de classification donnent en sortie une probabilité permettant de conclure si oui ou non, l’accord d’un crédit à un client peut s’effectuer. En général, la valeur par défaut du seuil est de 0,50

Cependant, il peut arriver que cette valeur puisse ne pas être optimale. En décidant d’ajuster ce seuil, il est possible d’optimiser le nombre de faux positifs et de faux négatifs.



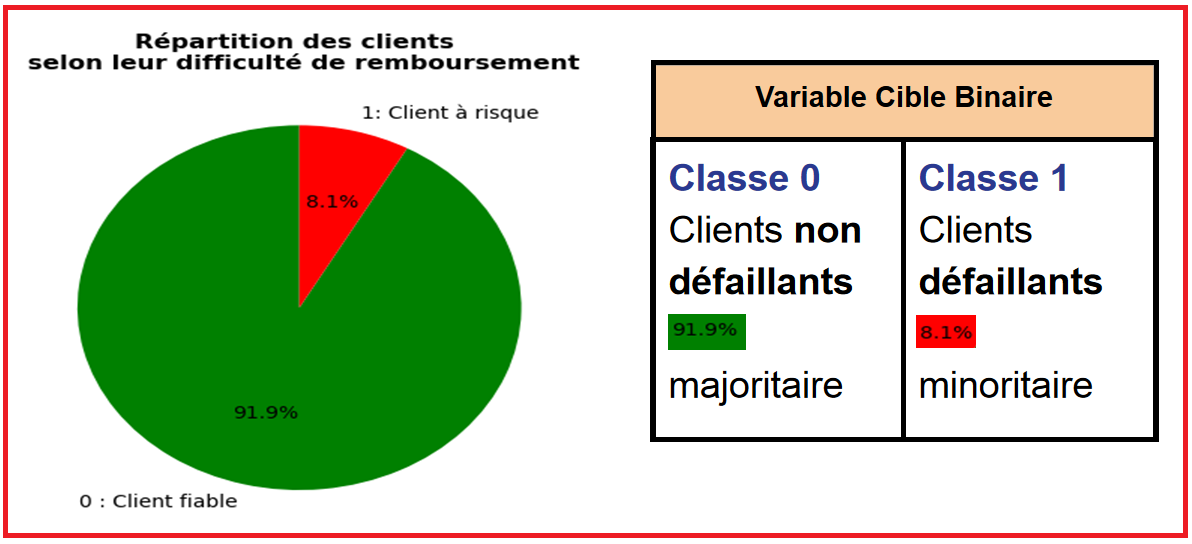
**2.5 Enregistrement des résultats avec MLflow**

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.

**2.6 Le traitement du déséquilibre des classes**

Lors de l'analyse exploratoire, nous avons remarqué que les données étaient très déséquilibrées entre les clients à risque et les clients fiables.

****

* **Classe 0 : 91.9 % des individus seront capables de rembourser ses crédits**:

**crédit accordé**

* **Classe 1 : 8.1 % des individus ne seront pas capables de rembourser ses crédits : crédit non accordé**

La plupart des modèles de Machine Learning vont ignorer la classe minoritaire et donc avoir des performances médiocres dans cette classe alors qu'en général c'est la performance de la classeminoritaire qui est la plus importante. Nous avons testé deux méthodes :

# 2.6.1 Rééquilibrage avec class\_weight :

# La méthode des “poids de classe” est une technique pour gérer les jeux de données déséquilibrés lors de la création d’un modèle. Elle attribue des poids différents aux différentes classes, en donnant plus de poids aux classes moins représentées. Cela influence le modèle pendant son entraînement, en rendant les erreurs de classification des classes minoritaires plus coûteuses que celles des classes majoritaires.

# 2.6.2 Rééquilibrage avec Sur-échantillonnage : SMOTE

Le sur-échantillonnage, ou “oversampling”, est une technique pour équilibrer les classes dans un jeu de données déséquilibré. Il crée des échantillons supplémentaires pour la classe minoritaire.

SMOTE, qui signifie “Synthetic Minority Oversampling Technique”, est une méthode spécifique de sur-échantillonnage. Au lieu de simplement dupliquer les échantillons minoritaires, SMOTE crée de nouveaux échantillons qui sont similaires mais pas identiques aux échantillons existants. Cela permet d’augmenter la diversité des échantillons minoritaires.

# 2.6.3 Rééquilibrage avec Sous-échantillonnage : RandomUnderSampler

Le sous-échantillonnage, ou "undersampling", est une méthode pour équilibrer les classes dans un jeu de données déséquilibré. Il réduit le nombre d'échantillons de la classe majoritaire.

# 2.6.4 Combinaison d'oversampling et undersampling : SMOTETomek

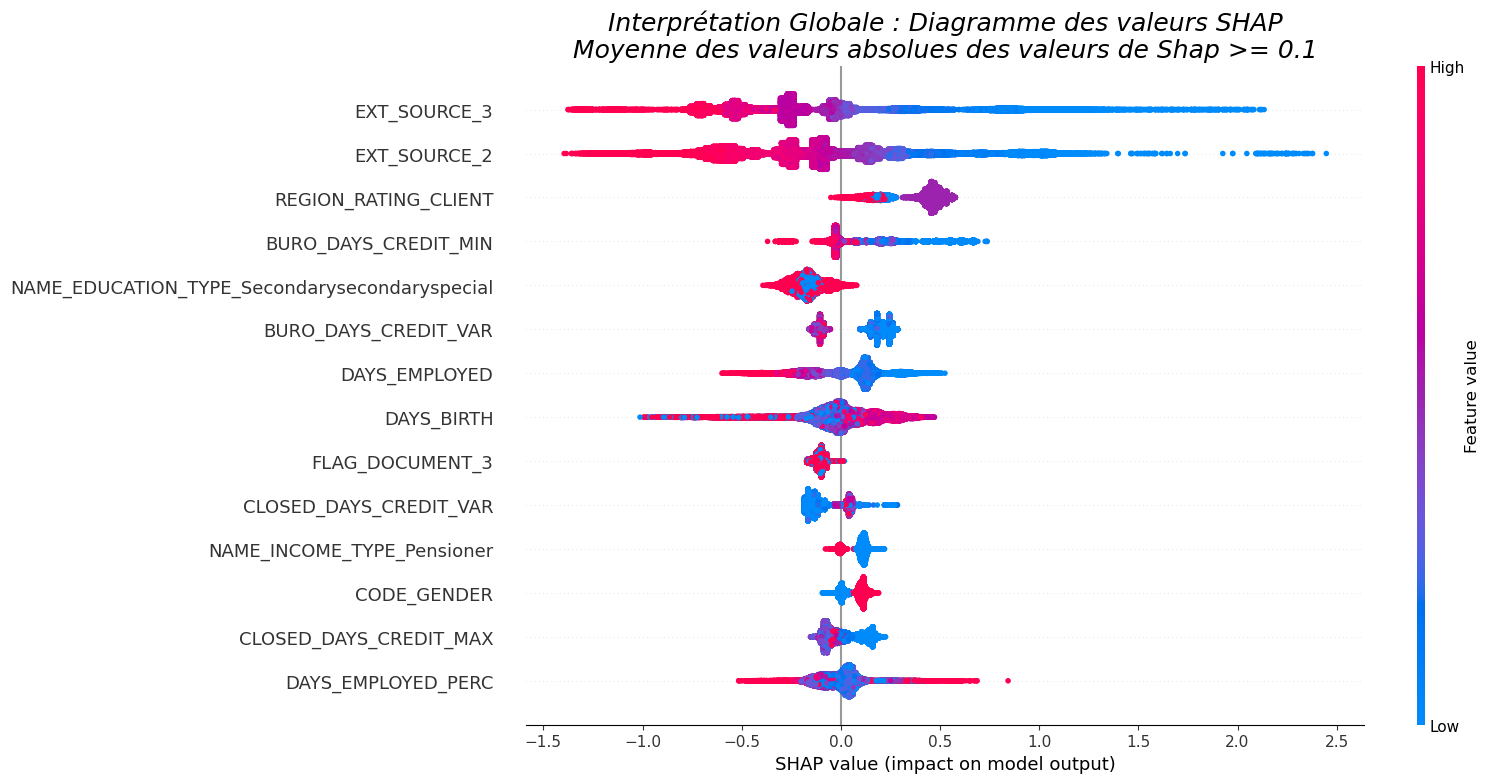
SMOTETomek est une technique combinant l'oversampling (SMOTE) et l'undersampling (Tomek Links). Elle vise à résoudre le déséquilibre de classes en augmentant la classe minoritaire par synthèse d'échantillons (SMOTE) tout en réduisant la classe majoritaire en supprimant les exemples qui se trouvent à la frontière entre les classes (Tomek Links).

* Nous avons opté pour la méthode de sous-échantillonnage avec RandomUnderSampler pour équilibrer nos données.
* Les méthodes class\_weight et RandomUnderSampler ont montré les meilleurs résultats en termes de score métier.
* Cependant, tous les modèles ne prennent pas en charge l'argument class\_weight.
* De plus, RandomUnderSampler est plus rapide que les autres méthodes. Nous avons donc choisi RandomUnderSampler pour pouvoir tester un plus grand nombre de modèles.

**4 L’interprétabilité globale et locale du modèle**

**4.1 Interprétabilité globale**

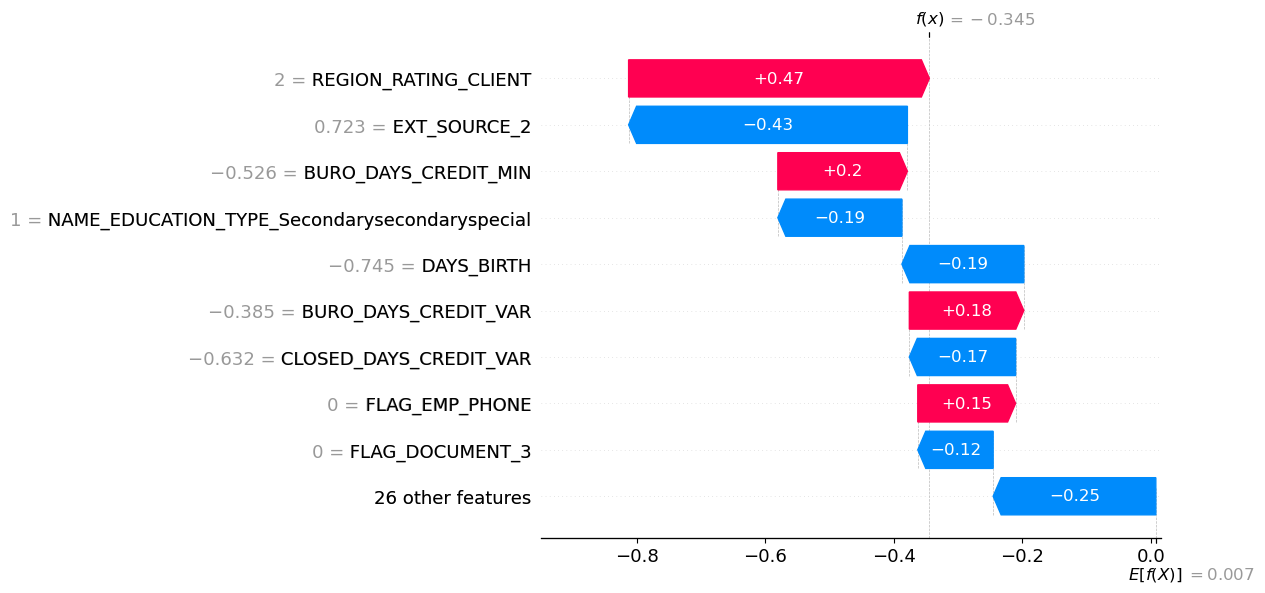
L'interprétabilité globale d'un modèle se réfère à notre capacité à comprendre comment le modèle prend ses décisions pour l'ensemble d’un jeu de données. Pour répondre à ces questions, l’importance des features a été utilisée. Cette métrique est intéressante pour détecter du data leakage (fuite de données). 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.



**4.2 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



**5- Analyse du Data Drift**

La librairie evidently a été utilisée pour détecter la présence de data drift en production dans le futur. L’hypothèse prise est que le dataset application\_train représente les données pour la modélisation et le dataset application\_test, les données de nouveaux clients une fois le modèle en production. Le rapport evidently sur le data drift compare les distributions de chaque variable dans les deux datasets. Le test statistique (ou la métrique appropriée) est automatiquement choisi en fonction du type des données et de leur volume et retourne les p-values ou distances et trace les distributions. Les tests statistiques ou palliers peuvent être ajustés manuellement.

