Note méthodologique

La méthodologie d'entraînement du modèle (2 pages maximum)

Pour développer un outil de scoring crédit capable de prédire la probabilité de remboursement d'un crédit par un client et de classifier les demandes en crédit accordé ou refusé, nous avons suivi une méthodologie structurée utilisant des données étiquetées provenant de huit jeux de données distincts. La variable cible est binaire : le crédit est remboursé (0) ou non (1).

Nous avons commencé par entraîner un modèle de référence, le DummyClassifier, pour établir une base de comparaison des performances des autres modèles. Ensuite, plusieurs modèles de classification ont été testés avec leurs paramètres par défaut, notamment : DummyClassifier, KNeighborsClassifier, LogisticRegression, GaussianNB, SVC, CatBoostClassifier, LGBMClassifier, DecisionTreeClassifier, RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier et XGBClassifier. Cette diversité de modèles permet d'explorer différentes approches de classification.

Chaque modèle a été entraîné sur les données d'apprentissage et évalué sur les données de test. Les performances des modèles ont été suivies et enregistrées avec MLFlow en utilisant plusieurs indicateurs de performance, notamment un score personnalisé basé sur une fonction de coût adaptée, le score AUC et l'accuracy. D'autres indicateurs comme le temps d'exécution, le score F1, la précision ont également été pris en compte.

Le modèle le plus performant, le CatBoostClassifier, a été sélectionné car permettant d'obtenir les meilleurs scores personnalisés, AUC et d'exactitude, tout en minimisant les proportions de faux positifs et faux négatifs.

Pour optimiser les hyperparamètres du CatBoostClassifier, nous avons utilisé GridsearchCV. Cette technique entraîne le modèle sur différentes combinaisons d'hyperparamètres et évalue ses performances sur les données de validation, sélectionnant ainsi la meilleure combinaison. Un pipeline de données intégrant SMOTE (Synthetic Minority Over-sampling Technique) et l'algorithme de classification a été mis en place. SMOTE a permis de générer de nouvelles instances de la classe minoritaire en interpolant entre les exemples existants, équilibrant ainsi le dataset pour un entraînement plus efficace (cf partie ci après). Nous avons également appliqué la technique de cross-validation avec GridsearchCV, divisant les données en k sous-ensembles (plis) pour entraîner le modèle sur k-1 plis et l'évaluer sur le pli restant, répétant cette procédure k fois pour une évaluation robuste.

En raison de la nécessité d'obtenir des résultats explicables pour les conseillers bancaires, nous avons analysé l'importance de chaque feature à l'aide de l'attribut feature_importances_ du CatBoostClassifier, sélectionnant ainsi les 10 features les plus importantes. Le jeu de données a été réduit à ces 10 features et le processus de sélection du meilleur modèle, d'optimisation des hyperparamètres et d'entraînement a été repris. À ce

stade, le GradientBoostingClassifier s'est avéré être le modèle le plus performant après réduction du jeu de données.

Pour fournir une explicabilité locale des prédictions du modèle, nous avons utilisé la technique SHAP (SHapley Additive exPlanations), qui explique la contribution de chaque feature à la prédiction du modèle pour une instance donnée.

Enfin, le modèle entraîné a été sérialisé et stocké au format pickle pour permettre une sauvegarde et un rechargement facile pour des prédictions futures.

Le traitement du déséquilibre des classes (1 page maximum)

Le déséquilibre des classes est un défi courant en machine learning, particulièrement dans des contextes comme le scoring crédit, où les cas de défaut de paiement sont souvent moins fréquents que les cas de paiement régulier. Pour gérer ce déséquilibre, deux approches principales peuvent être utilisées : l'undersampling et l'oversampling.

L'undersampling réduit la taille de la classe majoritaire pour équilibrer les proportions des classes. En retirant une partie des instances majoritaires, l'algorithme accorde plus d'importance aux individus minoritaires. Cependant, cette approche peut entraîner une perte d'information importante en éliminant des données potentiellement utiles.

L'oversampling augmente la taille de la classe minoritaire pour équilibrer les proportions des classes. Le nombre d'individus minoritaires est augmenté, ce qui permet de leur accorder plus d'importance lors de la modélisation. Différentes techniques existent pour l'oversampling, comme le clonage aléatoire des instances minoritaires ou l'utilisation de SMOTE.

Dans ce projet, nous avons choisi l'option d'oversampling, notamment via la technique SMOTE (Synthetic Minority Oversampling Technique). SMOTE génère des exemples synthétiques de la classe minoritaire en interpolant entre les instances existantes de cette classe, créant ainsi un ensemble de données plus équilibré sans simplement dupliquer les exemples existants.

Il est crucial de réaliser la division des données en ensembles d'entraînement et de test (TrainTestSplit) avant d'appliquer SMOTE. Cela permet d'éviter l'introduction de données synthétiques dans le jeu de test, ce qui pourrait fausser les résultats de validation (data leakage).

La définition de l’instance SMOTE inclut deux paramètres principaux :

- k_neighbors : Le nombre de plus proches voisins utilisés pour générer les exemples synthétiques.
- sampling_strategy : Le taux d'observations minoritaires à atteindre après le suréchantillonnage.

Nous faisons varier ces deux paramètres pour observer leur impact sur les performances du modèle baseline et choisir les paramètres les plus adaptés à nos données.

Nous effectuons également des tests pour évaluer les performances de divers modèles sur les données traitées avec et sans SMOTE. Cela nous permet de vérifier l'impact de SMOTE sur la qualité des prédictions du modèle.

L'utilisation de SMOTE pour le traitement du déséquilibre des classes dans notre modèle de scoring crédit s'avère être une approche efficace pour améliorer la performance du modèle. En augmentant le nombre d'observations minoritaires de manière synthétique, nous pouvons mieux équilibrer les classes et ainsi obtenir des prédictions plus fiables et pertinentes pour notre application métier. Cette approche garantit que notre modèle est capable de traiter efficacement les cas minoritaires, réduisant ainsi les risques de faux négatifs coûteux pour l'entreprise.

La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation (1 page maximum)

Dans notre projet de scoring crédit, il est crucial de définir une fonction coût métier reflétant les enjeux financiers de l'entreprise, d'utiliser un algorithme d'optimisation adéquat et de choisir une métrique d'évaluation pertinente pour mesurer la performance du modèle.

La fonction coût métier est conçue pour pénaliser particulièrement les faux négatifs (prêts accordés non remboursés) en raison de leur impact financier significatif. Elle est définie dans le package cost.py et attribue les poids suivants :

- Vrai Positif (TP) et Vrai Négatif (TN) : Poids de +1, car ils représentent des décisions correctes.
- Faux Positif (FP) : Poids de -1, représentant un manque à gagner potentiel.
- Faux Négatif (FN) : Poids de -10, représentant une perte en capital significative.

Le score global est obtenu en sommant les poids pondérés des TP, TN, FN et FP. Un score positif indique un gain financier, tandis qu'un score négatif indique une perte.

Pour optimiser notre modèle de scoring crédit, nous utilisons GridSearchCV en combinaison avec la technique de cross-validation. GridSearchCV permet de tester différentes combinaisons d'hyperparamètres et de sélectionner la meilleure en fonction de notre fonction coût métier personnalisée. Nous intégrons notre fonction coût métier dans GridSearchCV en utilisant la fonction make_scorer de scikit-learn.

Pour évaluer la performance de notre modèle, nous utilisons la fonction coût métier comme métrique d'évaluation principale. Cette approche garantit que le modèle optimisé répond aux exigences financières et opérationnelles de l'entreprise. En plus de notre fonction coût métier, nous suivons d'autres métriques standard pour avoir une vue d'ensemble complète de la performance du modèle, telles que :

- AUC Score
- Accuracy Score

Ces métriques complémentaires permettent de vérifier que le modèle offre de bonnes performances globales tout en étant optimisé pour notre objectif financier principal.

En utilisant une fonction coût métier personnalisée, un algorithme d'optimisation adapté et des métriques d'évaluation pertinentes, nous nous assurons que notre modèle de scoring

crédit est non seulement performant, mais aussi aligné avec les objectifs financiers de l'entreprise. Cela permet de maximiser les bénéfices tout en minimisant les risques associés aux prêts non remboursés.

L’interprétabilité globale et locale du modèle (1 page maximum)

Dans le cadre de notre projet de scoring crédit, il est essentiel de garantir l’interprétabilité du modèle par les conseillers bancaires pour assurer la transparence et la confiance des utilisateurs finaux. Pour ce faire, nous utilisons des techniques d’interprétabilité globale et locale, en particulier l’attribut feature_importances_ et SHAP (SHapley Additive exPlanations).

L’interprétabilité globale vise à comprendre le comportement général du modèle en identifiant les variables les plus influentes et en analysant leur impact global sur les prédictions. De nombreux algorithmes de machine learning, comme les arbres de décision et les forêts aléatoires, fournissent directement une mesure de l’importance des features via l’attribut feature_importances_. Cette mesure quantifie l’importance de chaque feature en termes de réduction de l’impureté ou de gain d’information qu’elle apporte au modèle. Les features avec des scores d'importance élevés sont considérées comme ayant un impact significatif sur les décisions du modèle.

Pour notre modèle de scoring crédit, nous avons utilisé cette technique pour évaluer les 150 features initiales et avons identifié les plus importantes. Par exemple, des features comme l’historique de crédit, le revenu et l'âge se révèlent souvent avoir un impact significatif sur les décisions du modèle.

L’interprétabilité locale vise à expliquer les prédictions du modèle pour des instances individuelles. SHAP est une méthode puissante pour cette tâche. Elle attribue à chaque feature une valeur Shapley, qui représente sa contribution à la prédiction pour une instance donnée. Basées sur des concepts de la théorie des jeux, ces valeurs fournissent une explication équitable et cohérente des prédictions.

Pour chaque demande de crédit, nous utilisons SHAP pour décomposer la prédiction du modèle en contributions individuelles de chaque feature. Par exemple, pour un client donné, SHAP peut montrer que des facteurs comme un revenu élevé et un faible ratio d’endettement ont contribué positivement à la prédiction de remboursement, tandis qu'un historique de crédit limité a eu un effet négatif. Ces explications sont cruciales pour les conseillers bancaires, car elles leur permettent de comprendre pourquoi une demande de crédit a été acceptée ou refusée et de communiquer ces raisons de manière claire et transparente aux clients.

Les visualisations SHAP, telles que les plots de valeurs SHAP et les summary plots, offrent des outils visuels intuitifs pour analyser les contributions des features. Les summary plots combinent l’importance globale et les distributions de valeurs SHAP, permettant de voir l’impact de chaque feature sur l'ensemble des prédictions du modèle. Les dependency plots montrent comment les contributions de deux features interagissent.

Nous avons intégré SHAP dans notre pipeline de machine learning pour générer automatiquement des explications pour chaque prédiction. Les valeurs SHAP sont calculées après l'entraînement du modèle et sont stockées avec les prédictions pour une analyse ultérieure. Cela permet aux utilisateurs finaux de consulter les explications des prédictions via une interface utilisateur conviviale.

L'utilisation des attributs feature_importances_ pour l'interprétabilité globale, combinée à SHAP pour l'interprétabilité locale, permet de démystifier le fonctionnement interne de notre modèle de scoring crédit. Cela renforce la confiance des utilisateurs en fournissant des explications claires et détaillées des décisions du modèle, tout en respectant les exigences réglementaires de transparence dans le secteur financier.

L'analyse du Data Drift (1 page maximum)

Dans le cadre de l'évaluation de la stabilité des modèles en production, une analyse du Data Drift a été réalisée pour détecter les changements potentiels dans la distribution des données entre le jeu de données utilisé pour la modélisation (application_train) et les nouvelles données provenant des clients en production (application_test). Nous avons utilisé la bibliothèque Evidently pour évaluer la cohérence des données et détecter les déviations.

Le dataset application_train représente les données utilisées pour entraîner le modèle.
Le dataset application_test représente les données de nouveaux clients soumises au modèle en production.

Nous avons comparé les données des jeux de données application_train et application_test pour détecter d'éventuels changements dans la distribution des principales caractéristiques (features). Cette analyse a permis d'évaluer la stabilité des données entre l'entraînement du modèle et sa mise en production. Les résultats ont été générés sous forme d'un rapport HTML détaillant les changements détectés pour chaque feature.

Nous avons également évalué le Target Drift en comparant les valeurs réelles avec les valeurs prédites par le modèle. Cette analyse visait à déterminer si les prédictions du modèle restaient fiables malgré d'éventuels changements dans les données de production. Les résultats ont également été présentés dans un rapport HTML.

Résultats
Analyse du Data Drift : Des déviations significatives ont été identifiées dans 9 des 121 colonnes étudiées entre les jeux de données application_train et application_test. Ces résultats soulignent l'importance de surveiller régulièrement la stabilité des données en production pour garantir la fiabilité continue du modèle.

Analyse du Target Drift : Aucun signe de Data Drift n'a été détecté, indiquant que les prédictions du modèle restent cohérentes avec les valeurs réelles malgré les changements potentiels dans les données de production.

L'utilisation de la bibliothèque Evidently s'est avérée efficace pour détecter les signes de Data Drift entre les jeux de données d'entraînement et de production. Cette analyse

régulière est cruciale pour maintenir la performance et la précision des modèles en production face aux évolutions des données réelles. En identifiant et en surveillant les déviations dans les données, nous pouvons garantir que les modèles continuent de fournir des prédictions fiables et précises, assurant ainsi la confiance des utilisateurs finaux et la stabilité opérationnelle.