P7_Nettoyage.ipynb

Étapes :

1. Jointure des différentes tables, feature engineering par les opérations d'agrégation distinctes pour les features numériques et catégoriques.
2. Éliminer les features colinéaire filtrés par les coefficients de corrélation de Pearson. (df.corr())
3. Éliminer les features qui contiennent plus 75% de manquants
4. Utiliser un ML modèle pour faire une première sélection de features : enlever tous les features de zéro importance, garder seulement les top 95% features selon l'attribut 'feature_importance'.

NOTE :

1. Tous les relevés mensuels comme bureau_balance.csv devrait être d'abord groupé par l'ID Bureau, ensuite regroupé par l'ID Client.
   Par exemple, '**client_bureau_balance_MONTHS_BALANCE_count_mean**' signifie que l'on calcul d'abord le count de variable 'MONTHS_BALANCE ' groupé SK_ID_BUREAU, puis la moyenne groupé SK_ID_CURR. Le nom du feature original est écrit en majuscule. A gauche du nom de feature sont les tables jointes. A droit du nom de feature sont les opérations d'agrégation adoptées.
2. Tous les tableaux clientèle comme application.csv devrait être groupé par le client ID.

P7_Modelisation.ipynb

1. Prétraitement de données :
   1) Séparation de Train/Test :
      X_fit, y_fit : jeu de données servant à l'entrainement de modèles présélectionnés (XGB) et l'optimisation des hyperparamètres.
      X_eval, y_eval : jeu de données servant à l'évaluation finale du modèle

   2) Initial Sous-échantillonnage (undersampling) : Pour faciliter l'apprentissage du modèle, nous avons procédé à l'équilibrage des classes par sous-échantillonnage de la classe majoritaire.
      X_balance, y_balance : jeu de donnée équilibré (subset de X_fit, y_fit) servant à l'entrainement de modèles présélectionnés (XGB, LR, SVM) et l'optimisation des hyperparamètres.

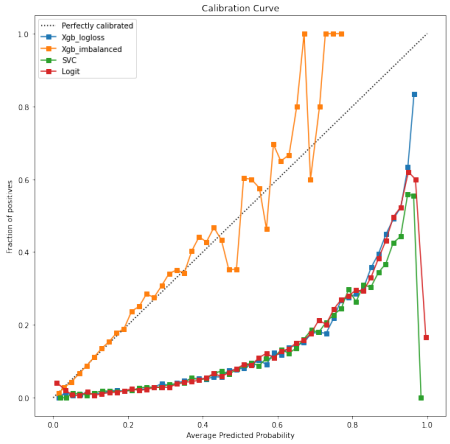
2. Modélisation :
   1) Entrainement de 4 modèles :
      - Entrainé par le jeu de données original déséquilibré(X/y_fit) :
        XGBoost
      - Entrainé par le jeu de données sous-échantillonné équilibré(X/y_balance) :
        XGBoost, SVC, Logistique Régression
   2) Cross Validation pour l'optimisation de hyperparamètres(logloss)

3. Métrique d'évaluation pour la sélection du modèle :
   1) **logloss**: $L_{\log}(y,p) = -(\mathrm{ylog}(p) + (1-y)\log(1-p))$

Pourquoi j'utilise le logloss mais pas 'accuracy', 'précision' ou 'recall'. Parce que l'objectif est de donner un score à chaque client, une probabilité de défaut à chaque client, mais pas simplement une étiquette (1 ou 0) défaut ou non défaut. Donc on va plutôt évaluer la fidélité de probabilité prédit. Pourtant, la métrique 'accuracy', 'précision' ou 'recall', ils sont plutôt pour mesurer la fidélité d'étiquette prédite. Le logloss s'occupe plutôt de la probabilité.

Résultat : XGBoost (0.571) < SVC (0.585) < LR (0.581)

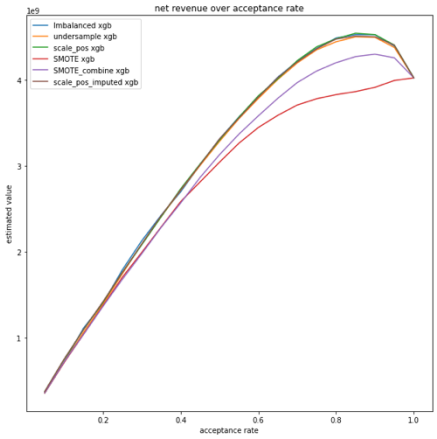
2) **Courbe de calibration** : vérifier si les probabilités en sortie sont bien calibrées. Cela signifie que la proportion d'individus ayant une même probabilité est égale à cette probabilité. L'algorithme :
   - Découper l'axe [0, 1] en plusieurs intervalles
   - Sélectionner les individus dont la probabilité de class positive appartient à chaque intervalle
   - Calculer la proportion d'individu appartenant à la classe positive (crédit défaut)



   - Plus proche de la diagonale, meilleur est la calibration. XGBoost donne un résultat un peu meilleur que les SVC et LR, surtout pour les individus ayant une haute probabilité de défaut.
3) Conclusion : Basé sur la comparaison de logloss et de courbe calibration, nous décidons de sélectionner le model XGBoost. Maintenant, nous allons optimiser le model sélectionné par l'équilibrage des classes.

4. Classification déséquilibrée :
Dans le cas de classes déséquilibrées, la classe majoritaire domine la classe minoritaire, ce qui fait que les classificateurs d'apprentissage automatique sont plus biaisés vers la classe majoritaire. Cela entraîne une mauvaise classification de classe minoritaire. Dans notre cas, le classificateur peut même prédire que toutes les données de test sont non-défaut crédit. Pour éviter cela, nous avons procédé à **l'équilibrage des classes** par 4 méthodes :
   1) Undersampling : sous-échantillonner la classe majoritaire (X_balance)
   2) Model Tuning: Tuning le XGBoost hyperparamètre scale_pos_weight
   3) SMOTE : sur-échantillonnage, qui vise à compléter le dataset original par des observations synthétiques des classes minoritaires.
   4) SMOTE combiné : sous-échantillonnage combiné avec sur-échantillonnage

5. Métrique d'évaluation pour l'équilibrage (**fonction coût**) :
   La positive classe est crédit défaut(y=1). Le négative classe est crédit non-défaut(y=0).
   Une prédiction False Positive (FP) entraînerait le rejet d'un client conforme, donc la perte de chiffre d'affaires et la perte de rémunération(intérêt) de prêt.
   Une prédiction False Négative (FN) entraînerait l'acception d'un client non-conforme, donc la perte du capital de prêt.
   Si nous acceptons 100% clients, le FN augmente et nous perdons le capital du prêt. Au contraire, si nous acceptons 0% clients, le FP augmente et nous perdons les chiffres d'affaires et l'intérêt du prêt. Pour régler ce type de trade-off entre FP et FN, nous avons défini une fonction à optimiser le taux d'acceptation (un threshold sur probabilité correspondant en même temps) pour maximiser la revenue nette :
   Revenue nette = Profit – Loss
   Profit = Montant total du prêt TN * profit_rate
   Loss = Montant total du prêt FN * LGD
   loss given default(LGD) est exprimé en pourcentage: perte du total du montant en cas de défaut d'un client.
   Ici nous supposons que LGD = 100%, profit_rate = 10%.
   C'est à dire on va perdre tout le capital en cas d'un défaut, on va gagner 10% du capital en cas de non-défaut.
   Le LGD dépends de la capacité de l'entreprise à gérer les crédits impayés. Ici, je suppose que le LGD égale à 100% pour pondérer la perte en cas de défaut afin de sélectionner un modèle plus prudent. Si vous préférez un portfolio plus gagnant cependant plus risqué aussi, vous pouvez diminuer le LGD à 60% ou moins.

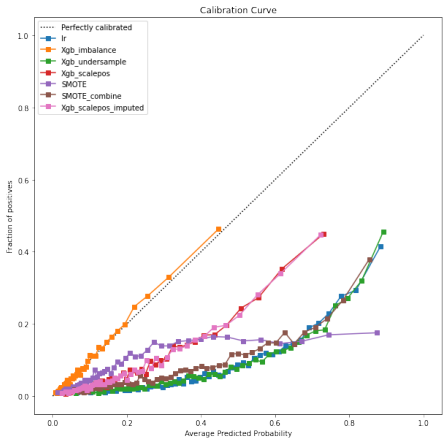


	acceptance_rate	threshold	bad_rate	estimated_value	n_accepted_loans
16	0.85	0.384	0.050	4.509662e+09	52294
17	0.90	0.450	0.057	4.502041e+09	55350
15	0.80	0.333	0.044	4.474492e+09	49223
18	0.95	0.551	0.065	4.403324e+09	58415
14	0.75	0.293	0.040	4.361820e+09	46127

Revenue nette

Nous voyons que :

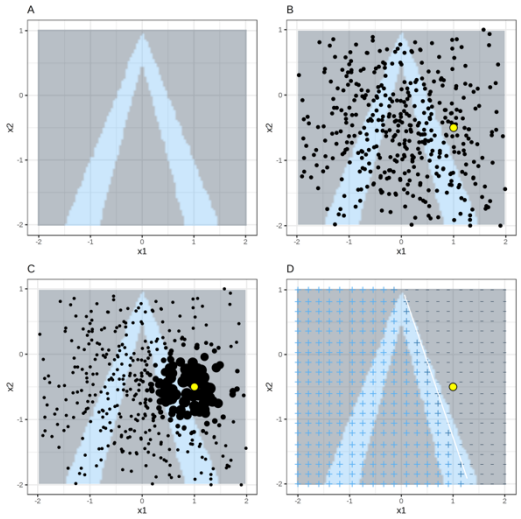
1) La revenue nette arrive au maximum lors d'un taux d'acceptation 85%, soit un seuil de probabilité environs 0.384. Cela signifie que nous devons refuser tous les clients recevant une probabilité de défaut > 0.384 pour maximiser la revenue.



1) Tuning le hyperparamètre scale_pos_weight (courbe red&rose) donne le meilleur résultat par rapport aux autres méthodes d'équilibrage
2) Le xgb entrainé par le dataset déséquilibré (courbe orange) donne aussi un bon résultat. Une hypothèse possible est que le training dataset déséquilibré contient beaucoup plus des individus que le dataset équilibré, ce qui rendre le xgb plus performant. Soyez prudent, on ne prend pas le modèle en dessus de la diagonale, ce qui signifie qu'il existe des défaut non-identifié.

6. Interprétabilité du modèle :
   Le modèle xgboost étant sélectionné sur la seule base de ses performances, il n'est moins interprétable par rapport au modèle linéaire. Nous avons seulement `feature_importances_` pour avoir une vue globale sur l'importance de features. Concernant à chaque client, chacune prédiction, nous allons utiliser la librairie `LIME` pour faire l'interprétation, surtout pour expliquer un rejet à un client.

Voici un exemple pour expliquer l'algorithme LIME pour data tabulaire :



A) Prédictions de la forêt aléatoire en fonction des caractéristiques x1 et x2. Classes prédites : 1 (sombre) ou 0 (clair).
B) Instance d'intérêt (point jaune) et données échantillonnées à partir d'une distribution normale (points noires).
C) Attribuer un poids plus élevé aux points proches de l'instance d'intérêt.
D) Les signes dans la grille (+, -) montrent les classifications du modèle appris localement à partir des échantillons pondérés. La ligne blanche marque la limite de décision (P(class=1) = 0,5). A partir de cette ligne, nous pouvons expliquer la prédiction à l'aide des coefficients de ligne.

7. Limites & Améliorations :
    1) Essayer plus de modèles : LGB (light gradient boosting), SGD (Stochastic Gradient Decent)
    2) Interprétabilité : Les coûts de calcul de SHAP sont trop élevés, c'est pourquoi nous utilisons le LIME. Si on trouve un serveur plus performant, nous pouvons essayer SHAP aussi.