

Note méthodologique décrivant :

- 1) La méthodologie d'entraînement du modèle (2 pages maximum)
- 2) Le traitement du déséquilibre des classes (1 page maximum)
- 3) La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation (1 page maximum)
- 4) Un tableau de synthèse des résultats (1 page maximum)
- 5) L'interprétabilité globale et locale du modèle (1 page maximum)
- 6) Les limites et les améliorations possibles (1 page maximum)
- 7) L'analyse du Data Drift (1 page maximum)

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

Nous avons entraîné un classificateur de type LGBMClassifier sur un jeu de données déséquilibré, pour prédire si un client est susceptible de rembourser ou non son prêt auprès d'une société financière.

Nous avons procédé en plusieurs étapes.

1.1) La première étape consiste à préparer les données pour l'entraînement du modèle. Pour cela, nous avons divisé l'ensemble des données en un dataset d'entraînement et un dataset de test. Le dataset d'entraînement contient toutes les données ayant une valeur de la cible (la variable à prédire) tandis que le dataset de test contient toutes les données pour lesquelles nous voulons prédire la cible.

Ensuite, nous avons prétraité les données d'entraînement en utilisant la fonction preprocessing :

```
df_train = df.loc[df['TARGET'].isna() == False]
df_pred = df.loc[df['TARGET'].isna()]

df_train.drop(columns = ['SK_ID_CURR'], inplace= True)

from scipy.stats import zscore
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

def preprocessing(df) :
    limitPer = len(df) * .80
    df.dropna(thresh=limitPer, axis=1, inplace=True)
    df.dropna(inplace=True)
    print(df.shape)
    numeric_cols = df.select_dtypes(include=['float64','int64']).columns
    mms = MinMaxScaler()
    mms = mms.fit(df[numeric_cols])
    df[numeric_cols] = mms.transform(df[numeric_cols])
    df[numeric_cols]=df[numeric_cols][df[numeric_cols].apply(zscore) < 4]
    df.dropna(inplace=True)
    df.reset_index(drop=True, inplace=True)
    return df, mms
```

Cette fonction effectue les traitements suivants :

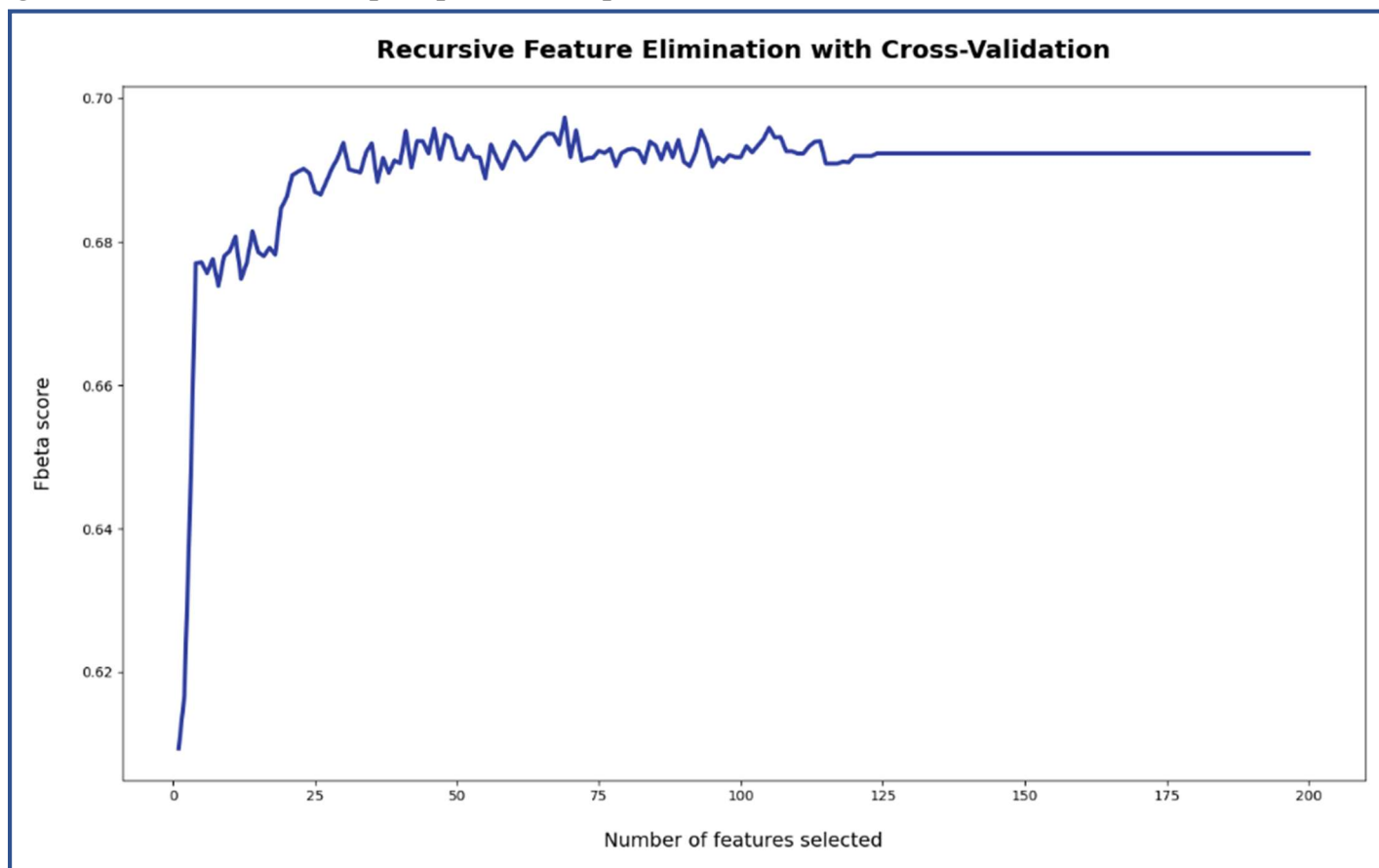
- Élimination des colonnes avec plus de 20% de données manquantes ;
- Normalisation des variables numériques (MinMaxScaler) ;
- Suppression des valeurs aberrantes en utilisant la méthode zscore avec une limite de 4 ;
- Élimination des lignes avec des valeurs manquantes.

1.2) Nous avons effectué un sous-échantillonnage du dataset d'entraînement pour résoudre le problème de déséquilibre entre les classes (cf. **partie 2) Le traitement du déséquilibre des classes**)

1.3) Nous avons utilisé la fonction RFECV pour sélectionner les features les plus pertinentes. Nous avons utilisé une cross-validation de 3, qui divise les données en 3 parties pour effectuer l'entraînement et la validation.

1.3.1) La fonction RFECV signifie Recursive Feature Elimination with Cross-Validation. Elle est une méthode d'élimination de features pour la sélection de features dans l'apprentissage automatique. Elle utilise une approche récursive pour supprimer les caractéristiques les moins importantes d'un ensemble de données tout en entraînant un modèle sur les caractéristiques restantes.

1.3.2) A partir ≈ 30 features pas d'augmentation significative de $F\beta$. En diminuant le nombre de features on simplifie notre problème et on augmente l'interprétabilité. Nous décidons ainsi de garder les 30 features les plus pertinentes pour notre modèle.



1.4) Nous avons entraîné un modèle LGBMClassifier sur les données transformées avec les paramètres obtenus l'aide d'une GridSearchCV pour trouver les paramètres optimaux pour le modèle.

1.5) Nous avons utilisé une fonction de coût métier personnalisée pour l'évaluation du modèle (cf. **partie 3) La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation**)

2) Le traitement du déséquilibre des classes (1 page maximum)

Pour résoudre le problème de déséquilibre des classes dans notre jeu de données, nous avons appliqué une technique de sous-échantillonnage aléatoire. Cette méthode consiste à éliminer aléatoirement des instances de la classe majoritaire jusqu'à ce que le nombre d'instances dans les deux classes soit équilibré.

Dans notre cas, la classe majoritaire correspond aux clients qui ont remboursé leur prêt, et elle est significativement plus représentée que la classe minoritaire des clients ayant fait défaut. Ce déséquilibre peut entraîner un biais dans le modèle entraîné. Ainsi, en plus du sous-échantillonnage, nous avons utilisé la fonction `fbeta_score` comme fonction coût métier pour tenir compte de l'asymétrie des coûts de faux positifs et de faux négatifs. Cela nous a permis de compenser le déséquilibre des classes et d'obtenir un modèle plus robuste.

```
# Under-sampling GS LGBMClassifier

X_train, X_test, y_train, y_test = train_test_split(df_train.loc[:, df_train.columns != 'TARGET'], df_train[['TARGET']])

X = pd.concat([X_train, y_train], axis=1)
class_count_0, class_count_1 = X['TARGET'].value_counts()
class_0 = X[X['TARGET'] == 0]
class_1 = X[X['TARGET'] == 1]

class_0_under = class_0.sample(class_count_1)
X = pd.concat([class_1, class_0_under], axis=0)
print('under_sampling :', X['TARGET'].value_counts())

X_train, y_train = X.loc[:, X.columns != 'TARGET'], X[['TARGET']]

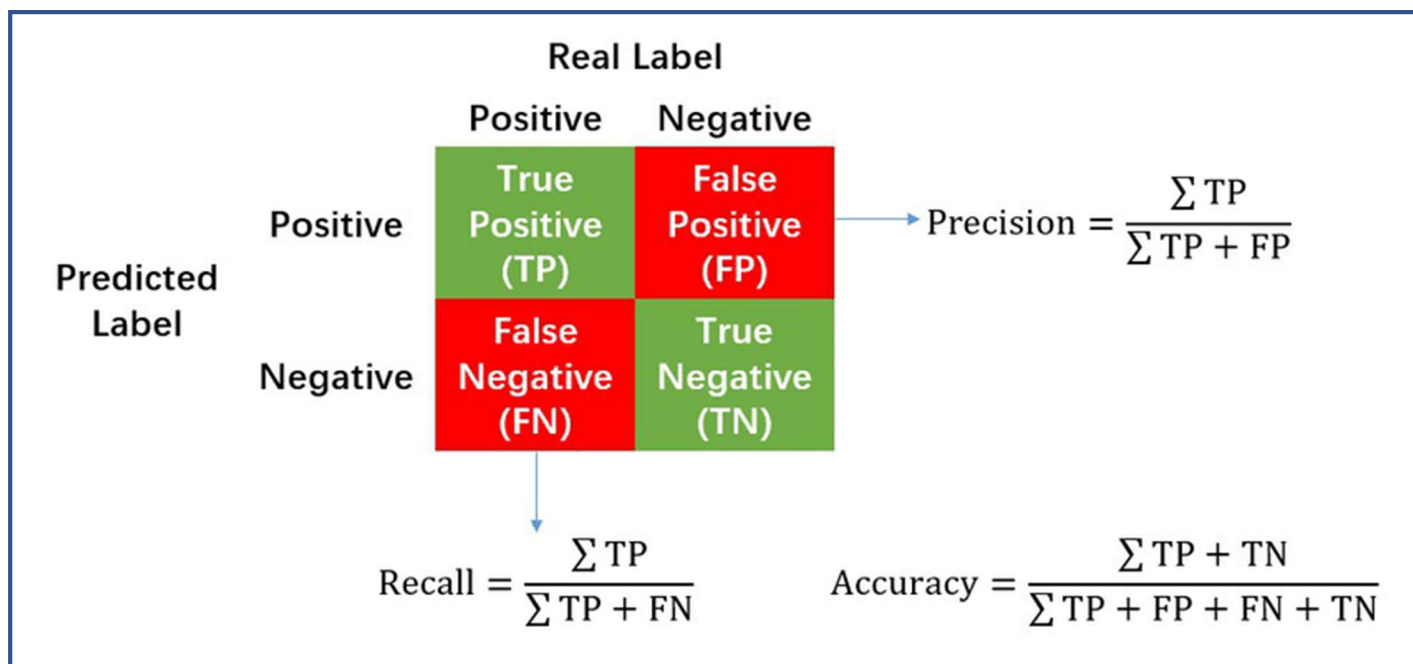
parameters = {"min_data_in_leaf" : np.arange(20,120,10), "learning_rate" : np.arange(0.01,0.10,0.01)}

model = LGBMClassifier()
selector = RFECV(model, step=1, cv=3, scoring=fb)
selector = selector.fit(X_train, y_train.values.ravel())
print(selector.transform(X_train).shape)
print('Optimal number of features: {}'.format(selector.n_features_))
clf = GridSearchCV(model, parameters, cv=3, scoring=fb)
clf.fit(selector.transform(X_train), y_train.values.ravel())
GSmodel = clf.best_estimator_
y_predict = GSmodel.predict(selector.transform(X_test))
print(metrics.fbeta_score(y_test, y_predict, beta = 3))
roc_auc = roc_auc_score(y_test, GSmodel.predict_proba(selector.transform(X_test))[:,1])
cf_matrix_roc_auc(GSmodel, y_test, y_predict, GSmodel.predict_proba(selector.transform(X_test))[:,1])
print(selector.get_params())

under_sampling : 0.0    12391
1.0    12391
Name: TARGET, dtype: int64
(24782, 30)
```

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

Il existe plusieurs métriques pour évaluer l'efficacité des différents modèles de machine learning. Cela dit, en raison de la spécificité de notre problème, nous ne pouvons pas simplement utiliser l'accuracy ou la précision. En effet, d'une part, nous avons un jeu de données déséquilibré et, d'autre part, une personne qui ne rembourse pas son prêt (faux négatif) coûte environ dix fois plus cher à la banque qu'une personne qui aurait pu obtenir un prêt mais que nous avons refusé (faux positif).



Pour tenir compte de cette particularité métier, nous allons utiliser la fonction F1 beta, dont la formule est présentée ci-dessous. La valeur de β est choisie de manière à ce que les faux négatifs aient environ 10 fois plus de poids que les faux positifs ($\beta^2=10 \Leftrightarrow \beta \simeq 3$).

$$F_{\beta\text{-score}} = \frac{TP}{TP + \frac{1}{1+\beta^2}(\beta^2 FN + FP)}$$

L'algorithme d'optimisation est le GridSearchCV, un algorithme de recherche par validation croisée de grille. Cet algorithme permet d'optimiser les paramètres des modèles d'apprentissage automatique. Il permet de déterminer la meilleure combinaison de paramètres pour maximiser les performances du modèle.

La métrique d'évaluation choisie est la F1 beta. Cette métrique permet de mesurer à la fois la précision et le rappel, qui sont deux indicateurs importants pour notre problème. La précision mesure la proportion de prédictions positives correctes par rapport à toutes les prédictions positives. Le rappel mesure la proportion de prédictions positives correctes par rapport à toutes les valeurs positives réelles. Le F1 beta est une moyenne pondérée de la précision et du rappel, avec un poids plus important pour le rappel. Dans ce cas, le poids du rappel est augmenté par un facteur de 3, ce qui signifie que les faux négatifs ont environ 10 fois plus de poids que les faux positifs.

4) Un tableau de synthèse des résultats (1 page maximum)

Après avoir testé les modèles ci-dessous (à partir du traitement des données d'entraînement + technique de sous-échantillonnage comme décrit dans la partie 1) La méthodologie d'entraînement du modèle), nous avons choisi le modèle LGBMClassifier() à partir des résultats F1 betascore :

Tests différents modèles

```
models=[LogisticRegression(),AdaBoostClassifier(),RandomForestClassifier(), GradientBoostingClassifier(),  
        DecisionTreeClassifier(),LGBMClassifier()]
```

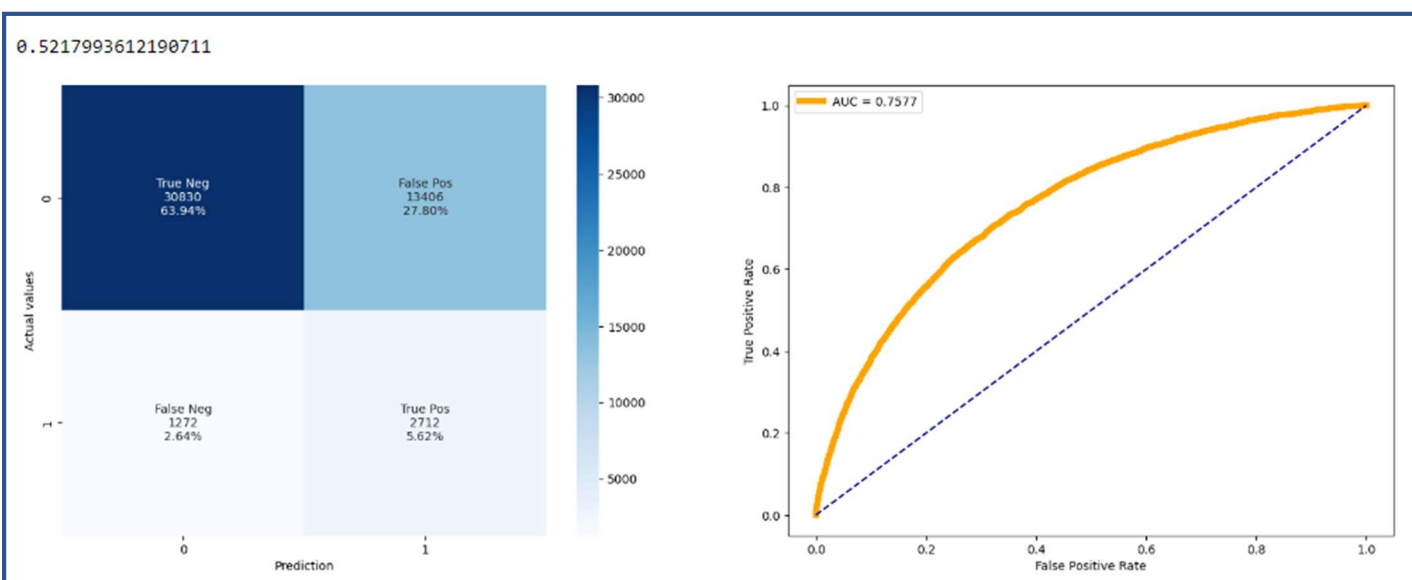
```
for model in models :  
    model.fit(X_train,np.array(y_train))  
    y_predict = model.predict(X_test)  
    print("F1 betascore model {0} : {1}".format(model,metrics.fbeta_score(y_test, y_predict, beta = 3)))
```

```
F1 betascore model LogisticRegression() : 0.5049806871315308  
F1 betascore model AdaBoostClassifier() : 0.5075403655438078  
F1 betascore model RandomForestClassifier() : 0.5035616775809332  
F1 betascore model GradientBoostingClassifier() : 0.5174176138686729  
F1 betascore model DecisionTreeClassifier() : 0.40619718309859165  
F1 betascore model LGBMClassifier() : 0.5206994348207545
```

Les résultats du modèle LGBMClassifier() montrent un score F1 betascore de 0,52, ce qui est un bon score compte tenu de la spécificité de notre problème. Les résultats de la matrice de confusion montrent que le modèle a une faible proportion de faux négatifs (2,64%) par rapport aux vrais positifs (5,62%). Cela signifie que notre modèle est assez précis pour prédire les personnes qui ne rembourseront pas leur prêt.

En revanche, le modèle a une proportion non négligeable de faux positifs (27,80%), cela signale le risque que le modèle prédise à tort que certaines personnes ne rembourseront pas leur prêt.

Enfin, l'aire sous la courbe ROC (AUC) est de 0,75, ce qui indique que notre modèle a une capacité assez bonne à distinguer entre les remboursements et les non-remboursements de prêts.

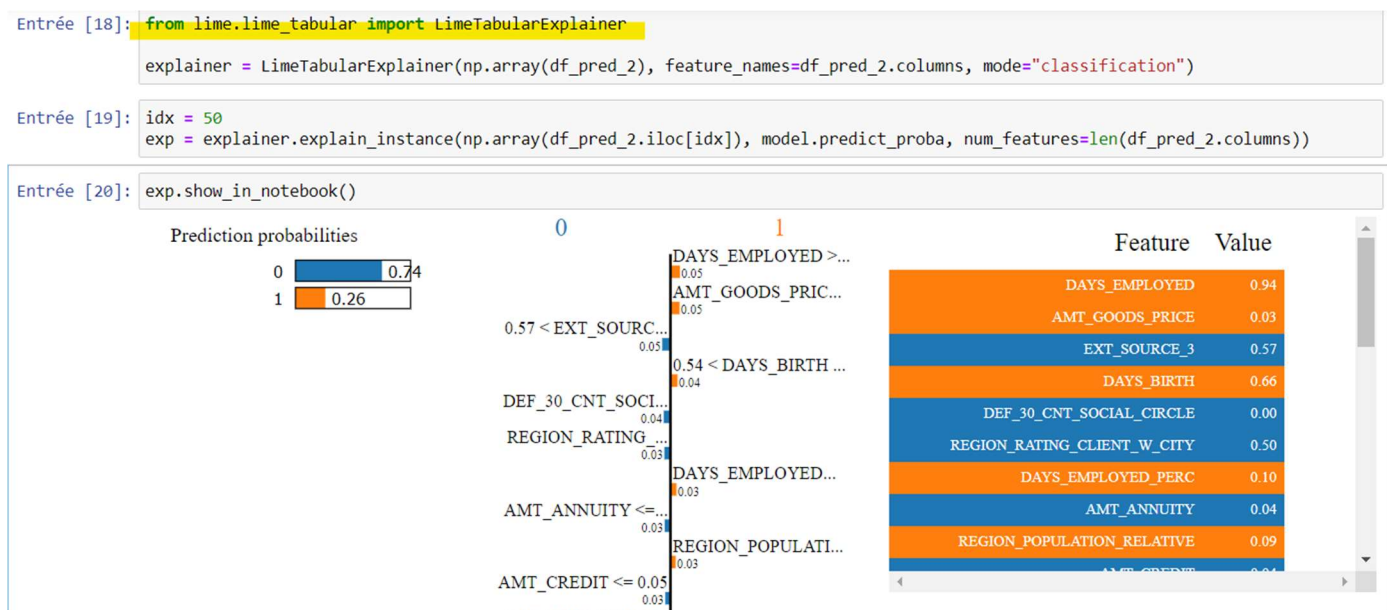


5) L'interprétabilité globale et locale du modèle (1 page maximum)

L'interprétabilité globale fait référence à la compréhension globale du modèle et de son fonctionnement, c'est-à-dire comment les caractéristiques d'entrée affectent la sortie du modèle. Elle permet de déterminer quelles variables sont importantes pour le modèle et comment elles sont utilisées pour prendre des décisions. Dans ce sens, nous avons utilisé la fonction RFECV (cf. partie 1) La méthodologie d'entraînement du modèle) de façon à garder que les features les plus pertinentes, simplifiant ainsi le problème initial et augmentant l'interprétabilité du modèle.

L'interprétabilité locale, quant à elle, se concentre sur la compréhension de la façon dont le modèle prend des décisions pour un exemple de données spécifique. Elle permet de déterminer comment chaque caractéristique d'un exemple de données particulier contribue à la décision prise par le modèle.

Pour l'interprétation locale du modèle, nous avons utilisé la méthode LIME (Local Interpretable Model-Agnostic Explanations). Cette méthode permet d'expliquer les prédictions du modèle pour une observation donnée en identifiant les caractéristiques les plus importantes qui ont influencé la prédiction.



6) Les limites et les améliorations possibles (1 page maximum)

Le modèle de classification présenté utilise une technique de sous-échantillonnage (under-sampling) en combinant les observations de la classe minoritaire avec un nombre équivalent d'observations de la classe majoritaire, suivi d'une Grid Search avec une validation croisée pour optimiser les hyperparamètres du modèle LGBMClassifier. Nous avons également sélectionné les features les plus pertinentes à partir de la fonction RFECV (comme décrit dans la partie 1) La méthodologie d'entraînement du modèle).

En somme, le modèle présenté a montré des résultats encourageants, mais il reste des améliorations possibles à explorer pour atteindre un équilibre optimal entre la précision et la réduction des faux positifs et des faux négatifs.

Les résultats du modèle ont montré un taux de faux négatifs (FN) de 2,64%, qui est considéré comme faible et répond à l'objectif de minimiser le risque de ne pas prédire correctement les personnes qui ne rembourseront pas leur prêt. Cependant, le taux de faux positifs (FP) est relativement élevé avec un pourcentage de 27,80%, ce qui signifie que le modèle prédit à tort que certaines personnes ne rembourseront pas leur prêt.

L'under-sampling a été utilisé dans cette étude en raison de sa rapidité et de sa rentabilité (technique moins coûteuse en termes de ressources). Nous avons testé les techniques SMOTE et over-sampling avec le modèle LGBMClassifier, mais les résultats étant proches ou inférieurs à ceux obtenus à partir du traitement under-sampling, nous avons choisi ce dernier.

Cela dit, nous n'avons pas testé les traitements SMOTE et over-sampling avec le modèle LGBMClassifier en faisant la sélection des features les plus pertinents à partir de la fonction RFECV. Cela peut être une bonne piste à explorer lors des traitements à l'avenir.

L'utilisation de modèles plus avancées telles que les réseaux de neurones profonds ou les modèles ensemblistes peut également améliorer les résultats. Il pourrait être pertinent de tester ces modèles avec des techniques de sur-échantillonnage (over-sampling / SMOTE) pour améliorer encore plus la performance de la classification.


7) L'analyse du Data Drift (1 page maximum)

Cette analyse de data drift s'inscrit dans le cadre du développement d'un modèle de scoring visant à prédire la probabilité de défaut de paiement d'un client. Pour l'entraînement, nous avons utilisé le dataset "application_train" et le modèle sera appliqué en production sur le dataset "application_test". L'objectif de cette analyse est d'évaluer la stabilité des données entre ces deux datasets afin de déterminer si le modèle peut être déployé en production sans risque de sur-apprentissage ou de sous-apprentissage. Pour cela, nous avons utilisé la librairie Evidently.

À partir des datasets "application_train" et "application_test", nous avons construit les datasets "reference" et "current" que nous avons nettoyés à l'aide de la fonction "preprocessing" (comme décrit dans la partie 1) La méthodologie d'entraînement du modèle). Nous avons ensuite sélectionné aléatoirement 5 000 échantillons pour chaque dataset, et conservé les 30 features les plus importants selon la fonction RFECV (également décrite dans la partie 1) La méthodologie d'entraînement du modèle).

Le rapport Evidently a indiqué que le data drift n'était pas détecté, avec une limite de détection fixée à 0,5. Cela signifie qu'une différence supérieure à cette limite entre les données actuelles et les données de référence serait considérée comme un drift. Cependant, un drift a été détecté pour 14 des 30 colonnes, soit environ 46,667% des colonnes. Il peut être pertinent de surveiller ces colonnes afin de détecter toute évolution future qui pourrait indiquer un data drift.

Nous tenons à souligner que les quatre features les plus importantes parmi les 30 que nous avons sélectionnées à partir de la fonction RFECV présentent des résultats encourageants d'après le rapport data drift :

Column	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score ↑
> DAYS_BIRTH	num			Not Detected	Wasserstein distance (normed)	0.040593
> EXT_SOURCE_3	num			Not Detected	Wasserstein distance (normed)	0.049243
> EXT_SOURCE_2	num			Not Detected	Wasserstein distance (normed)	0.061874
> DAYS_EMPLOYED	num			Not Detected	Wasserstein distance (normed)	0.089221