



Prêt à dépenser

Implémentez un modèle de scoring

Projet 7 du parcours
« Data Scientist » d'OpenClassrooms

Mark Creasey

Sommaire

Implémentez un modèle de
scoring

- 01 La problématique
- 02 Les données
- 03 Modélisation
- 04 Dashboard
- 05 Conclusion

01 Présentation de la problématique

Mission – Implémenter un modèle de scoring

La société financière « Prêt à dépenser » propose:

- des **crédits à la consommation**
- pour les personnes ayant **peu ou pas du tout d'historique** de prêt

Basée sur les données financières

- Données internes
- Données externes



Critères de succès

- **Maximiser** le nombre de **prêts aux clients qui peuvent payer** (le profit)
- **Minimiser les pertes** en refusant les clients qui ne peuvent pas repayer
- **Transparence de la décision** sur l'octroi du crédit
- **Déploiement** d'un dashboard permettant de visualiser les informations clients pour
- **Permettre l'interprétation** de la décision faite par le modèle

02 Les données

Nettoyage, exploration

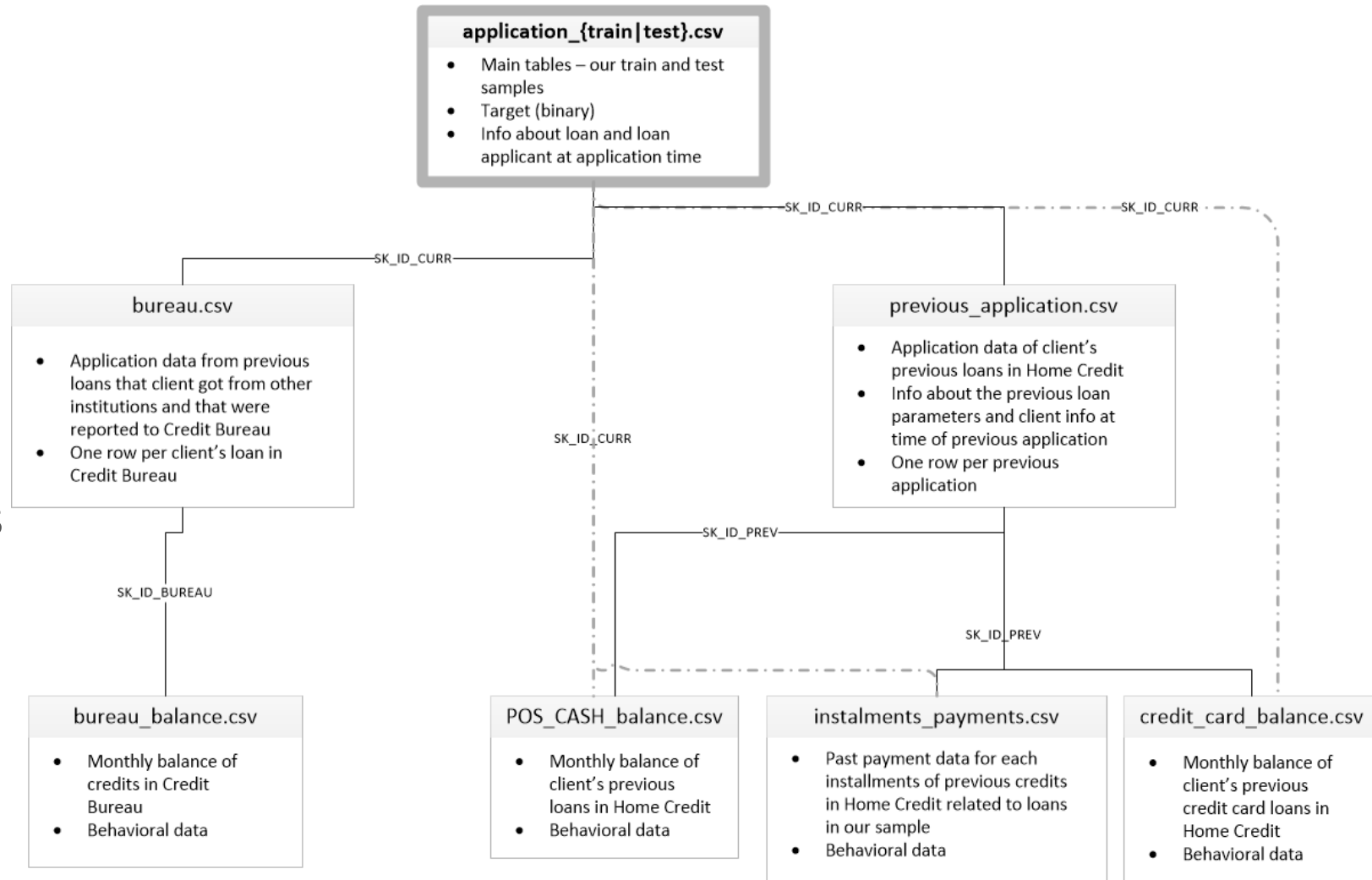
Données financières sous 7 tables

Feature engineering script:

- <https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features/script>
- Agrégations
- LabelEncode (catégories binaires)
- OneHotEncode (catégories)
- Traitement valeurs aberrantes

Sélection de top 100 features

- Ensemble de Filter, Embed et Wrapper méthodes
- Elimination de colonnes hautement colinéaires ($VIF > 5$)



Distribution de la variable cible très déséquilibrée

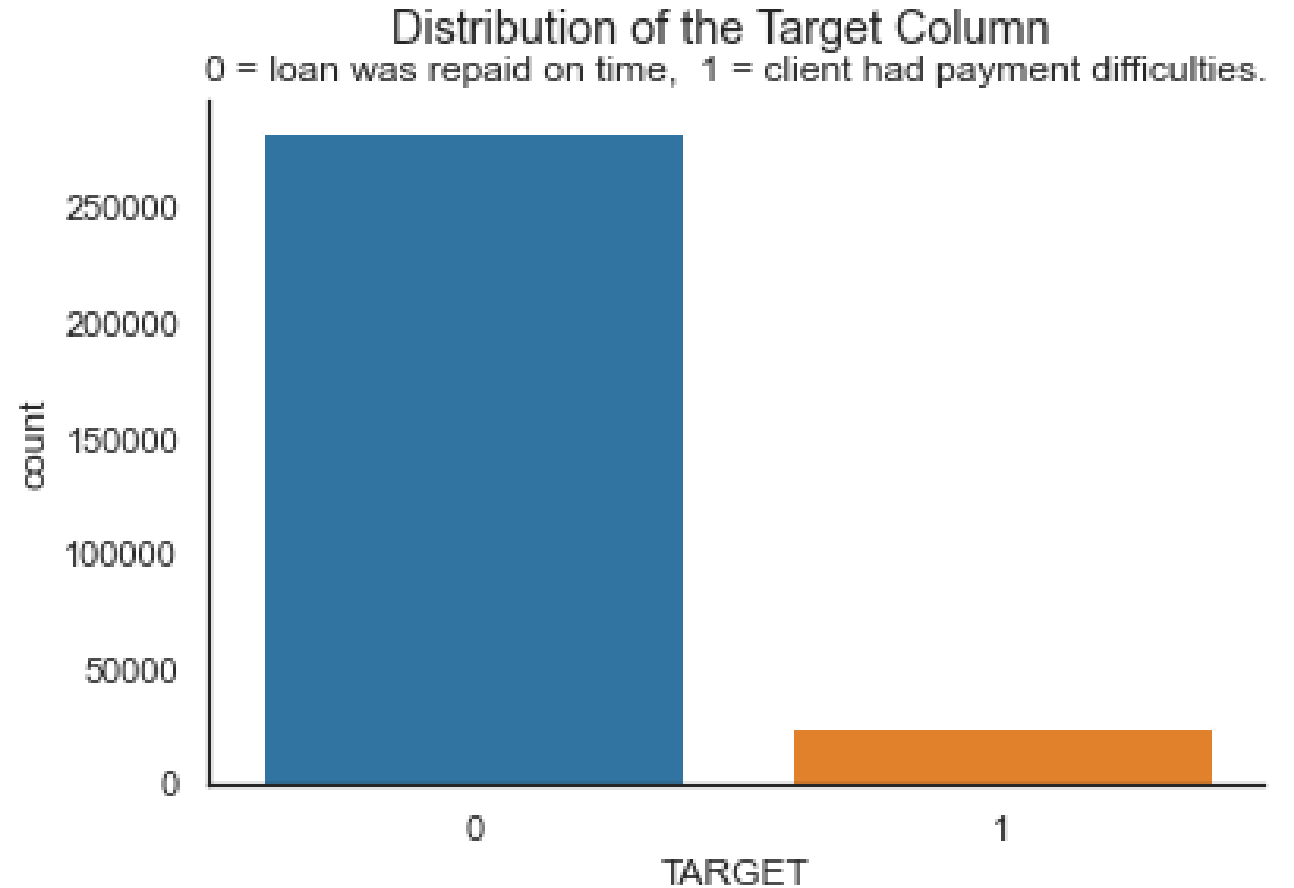
Prédiction que tous les clients sont bons

→ précision de 93%

→ on n'aura identifié aucun client défaillant.

- 0 – client non-défaillant
- 1 – client défaillant

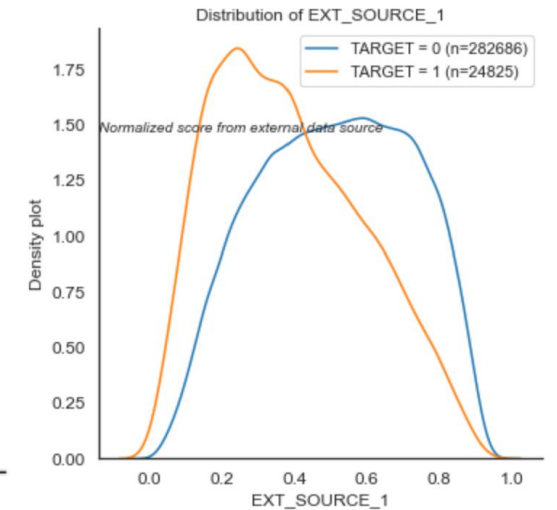
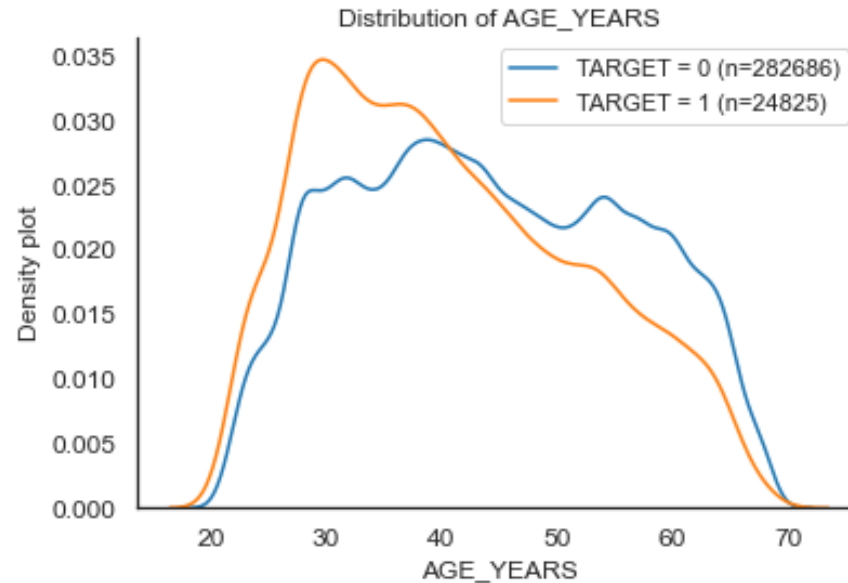
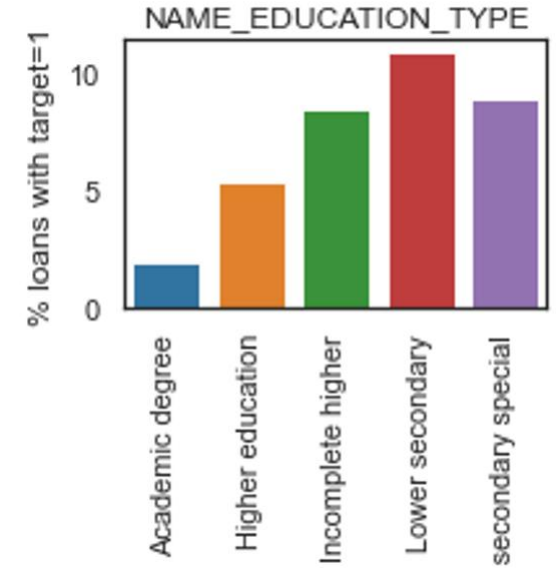
Certains modèles sont très sensibles à des classes déséquilibrées



Exploration des features importants

Facteurs de risque (entre autres)

- Les hommes
- < 40 ans
- Bas niveau d'éducation
- Un score < 0.5 dans des sources externes
 - EXT_SOURCE_1
 - EXT_SOURCE_2
 - EXT_SOURCE_3



03. Modélisation

Sampling : Options pour re-équilibrage des classes

Cost-sensitive

- Plusieurs modèles permettent de multiplier le score de la classe minoritaire par un poids (`class_weight = 'balanced'`)

Random under/over sample

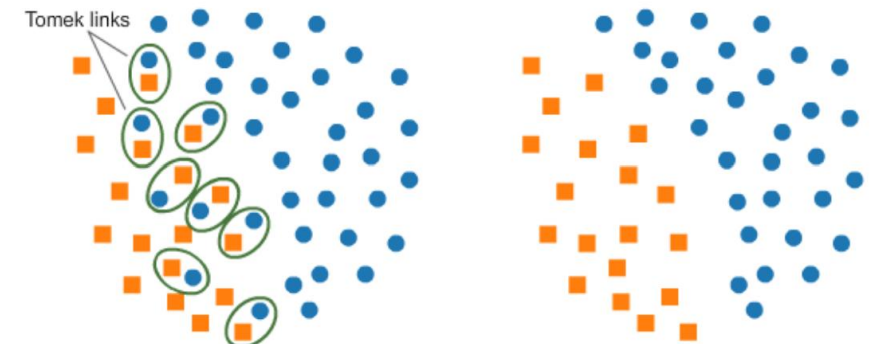
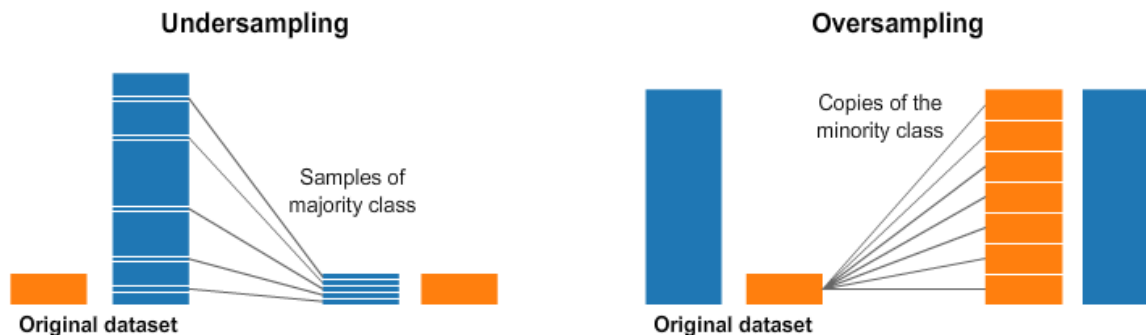
- Undersample: sélectionner aléatoirement le même nombre de clients dans la classe majoritaire qui sont dans la classe minoritaire
- Oversample: dupliquer aléatoirement les clients de la classe minoritaire

SMOTE (Synthetic Minority Oversampling Technique)

- synthétiser des nouveaux éléments pour la classe minoritaire, basés sur les valeurs des variables des k voisins les plus proches

SMOTE Tomek links

- Oversample la classe minoritaire avec SMOTE, puis supprimer les paires d'observations plus proches mais de classes différentes



Métriques d'évaluation

Précision

- Quelle portion des clients prédits comme défaillants sont de la vraie classe défaillante?

$$\frac{TP}{TP + FP}$$

Recall

- Quelle portion de la vraie classe est présente dans le cluster prédit ?

$$\frac{TP}{TP + FN}$$

F1 Score

- accuracy « équilibré » :

$$2 * \frac{Precision * Recall}{Precision + Recall}$$

F(beta) score

- Peser plus sur recall (beta > 1)

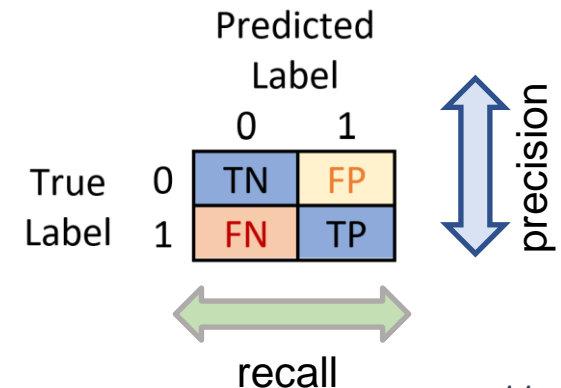
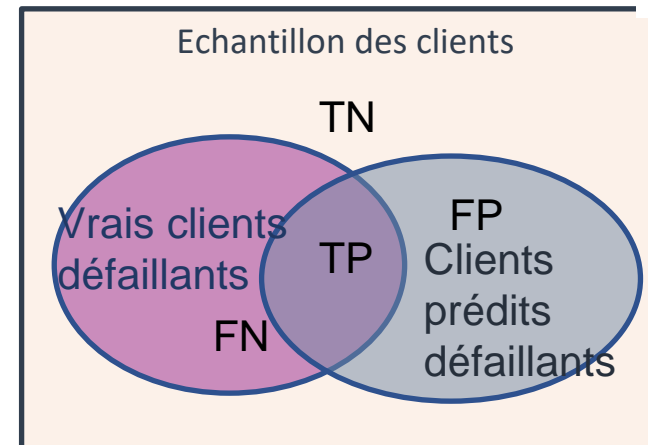
$$F_{\beta} = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

ROC AUC

- Meilleur compromis entre sensibilité (FPR) et spécificité (TPR)

- $FPR = \frac{FP}{FP + TN}$

- TPR = recall



La fonction coût métier

Un score représentant un bénéfice financier pour la banque:

Les gains (profit) :

- donner un prêt à un bon payeur (TN)
- refuser un prêt à un mauvais payeur (TP)

Les pertes (loss) :

- donner un prêt à un **mauvais payeur (FN)**
- refuser un prêt à un bon client (FP)

Custom Credit Score (entre 0 et 1)

$$= \frac{(\text{Profit} - \text{Loss})}{(\text{Max Profit} - \text{Max Loss})}$$

Les pondérations (à revoir avec les experts métier):

- **tn_profit** = Profit moyenne par prêt = 1
- **fp_loss** = perte moyenne par prêt non donné = 0.5
- **fn_loss** = perte moyenne par prêt défaillant = 10
- **tp_profit** = profit de refuser un mauvais payeur = 0.2

Coût métier

$$= (\text{TN} - 0.5 \cdot \text{FP} - 10 \cdot \text{FN} + 0.2 \cdot \text{TP})$$

normalisé entre 0 et 1

Pénaliser les FN

=

Favoriser le recall

Evaluation AUC - compromis entre précision et recall

LGBM (weight-balanced) - Summary.

best params={'clf__class_weight': 'balanced', 'clf__max_depth': 6, 'clf__min_child_samples': 50}

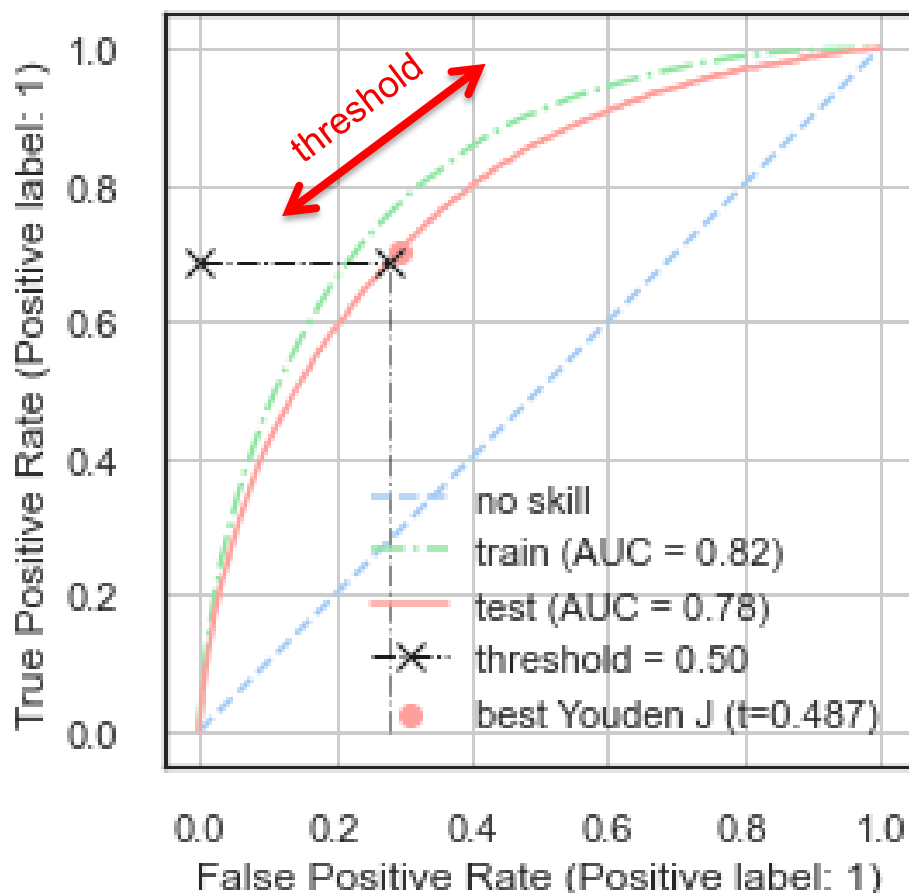
Confusion Matrix

True label	Predicted label	
	0	1
0	0.72	0.28
1	0.31	0.69

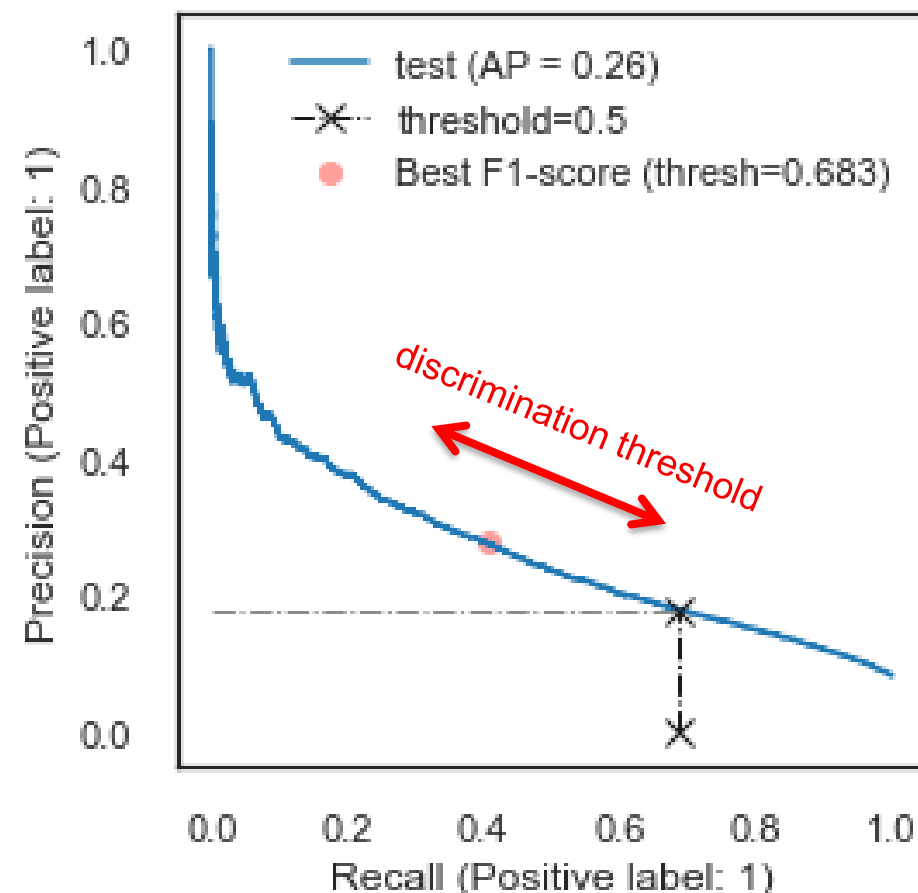
counts

True label	Predicted label	
	0	1
0	40891	15646
1	1561	3404

ROC Curve



Precision - Recall Curve



Les modèles

Chaque Modèle

composé d'une pipeline imblearn:

- Preprocess (preprocessor)
- Sampling (sampler)
- Feature Selection (passthrough)
- Classification (classifier)

Train-Test Split

- 80% données train, 20% test

Sampling

- Cost-sensitive (balanced weights)
- Random under/over sample
- SMOTE / TomekLinks

Les Classifieurs

- Dummy
- RidgeClassifier (linéaire, rapide)
- LogisticRegression
- RandomForestClassifier
- LightGBM Classifier

Stratified Gridsearch

Avec données train sur les hyperparamètres associés à chaque modèle

Cross-validation

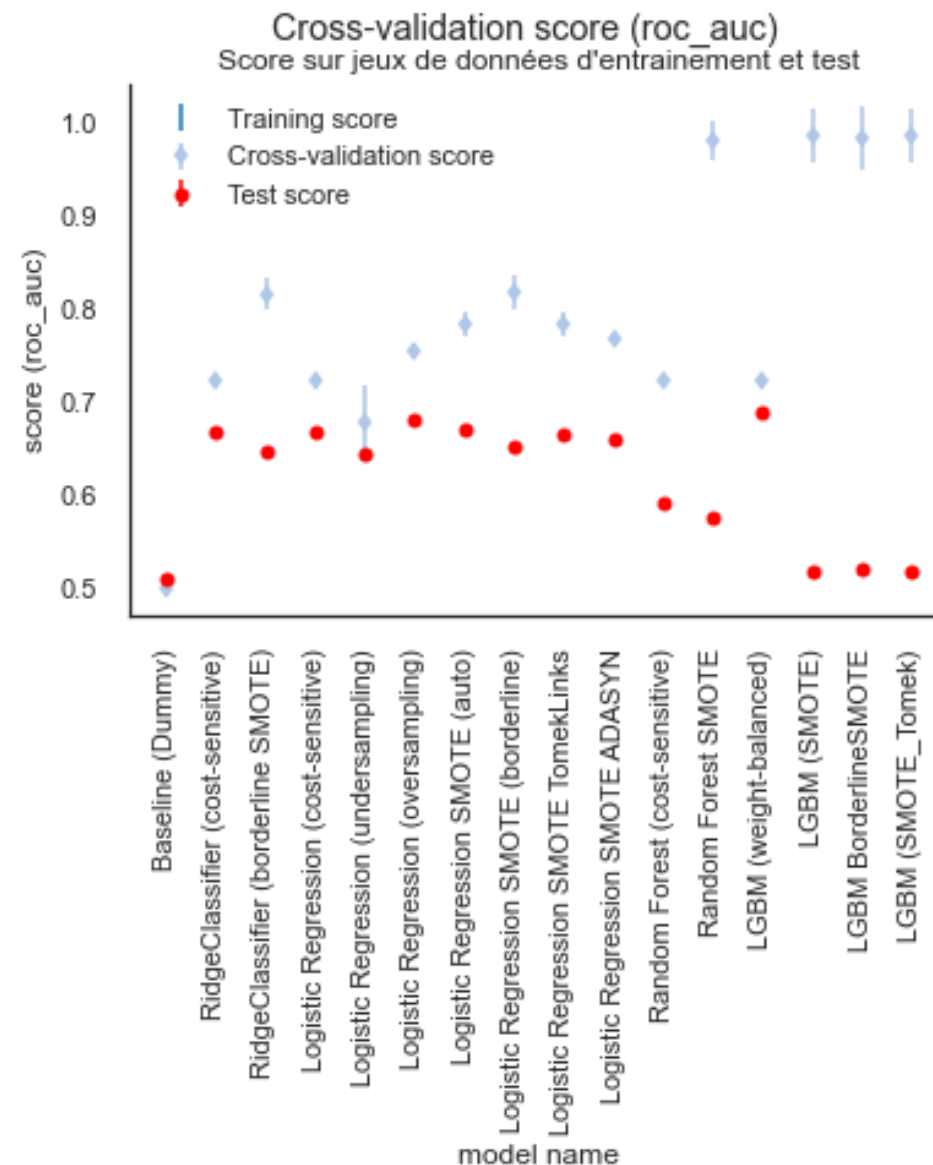
- sur des données test

Performance metrics

Choix du modèle

Meilleur ROC_AUC sur le jeu de test:

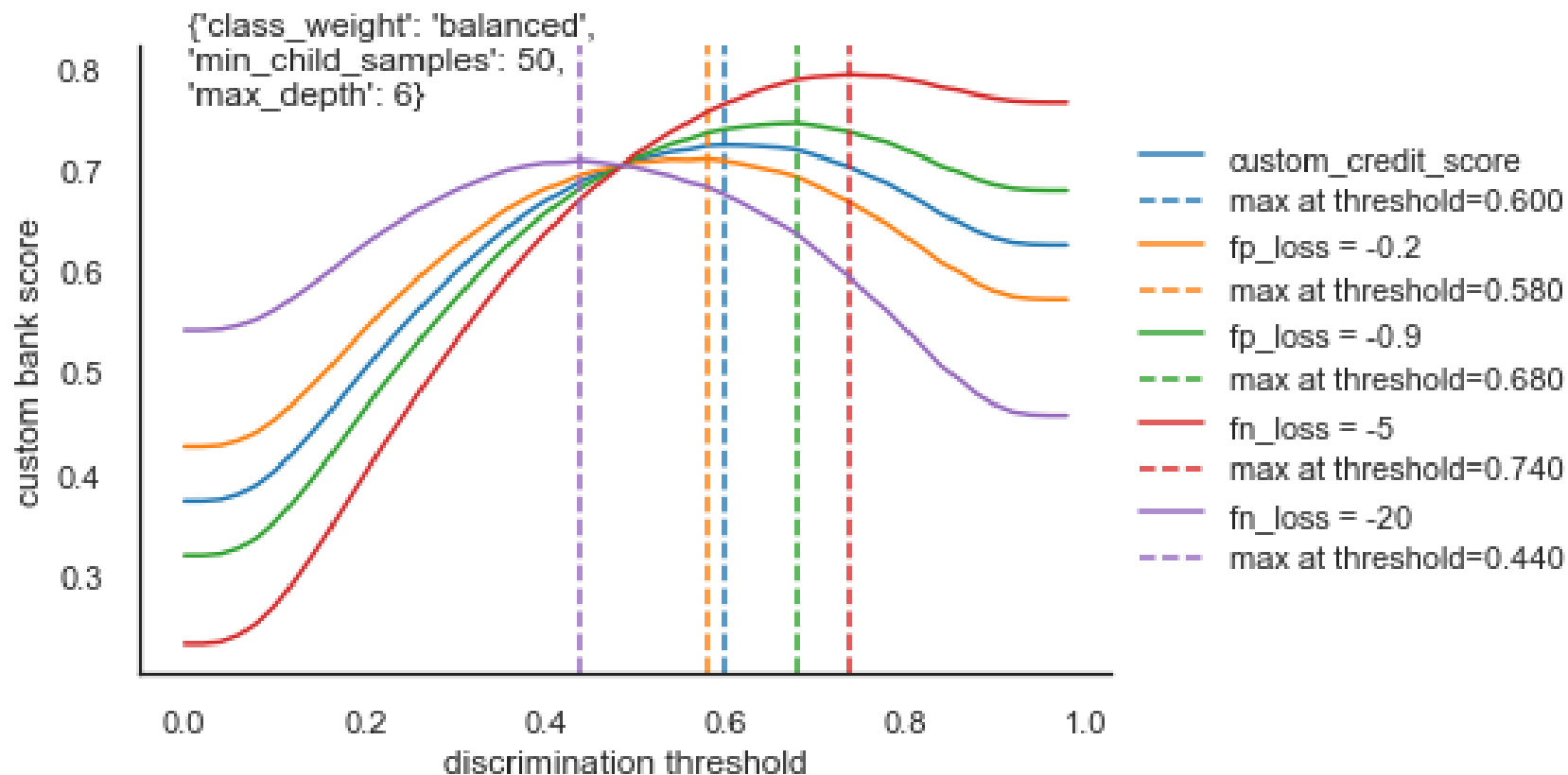
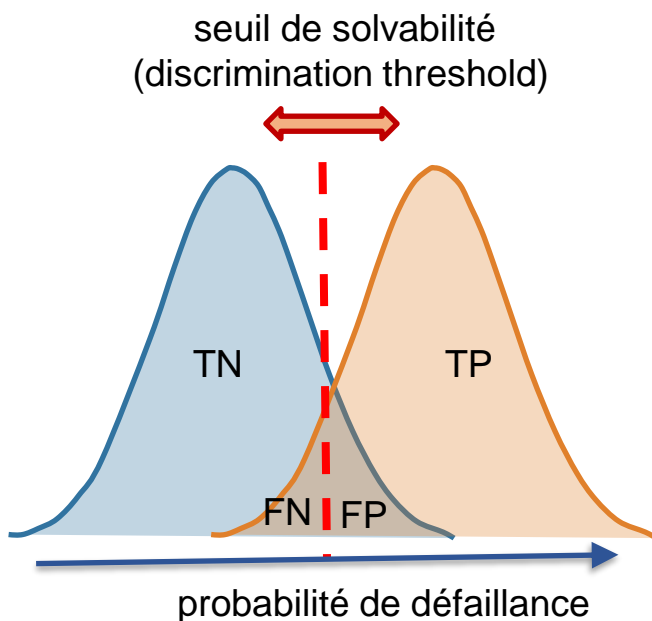
- LightGBM Classifier
 - class_weight = balanced
 - Min child_samples=50
 - max_depth = 6
- Autres méthodes de sampling:
 - overfitting sur jeu d'entraînement (Random Forest et LGBM)
 - SMOTE très lent à faire l'entraînement
 - Amélioration de ROC_AUC n'est pas significative



Optimisation du seuil (discrimination threshold)

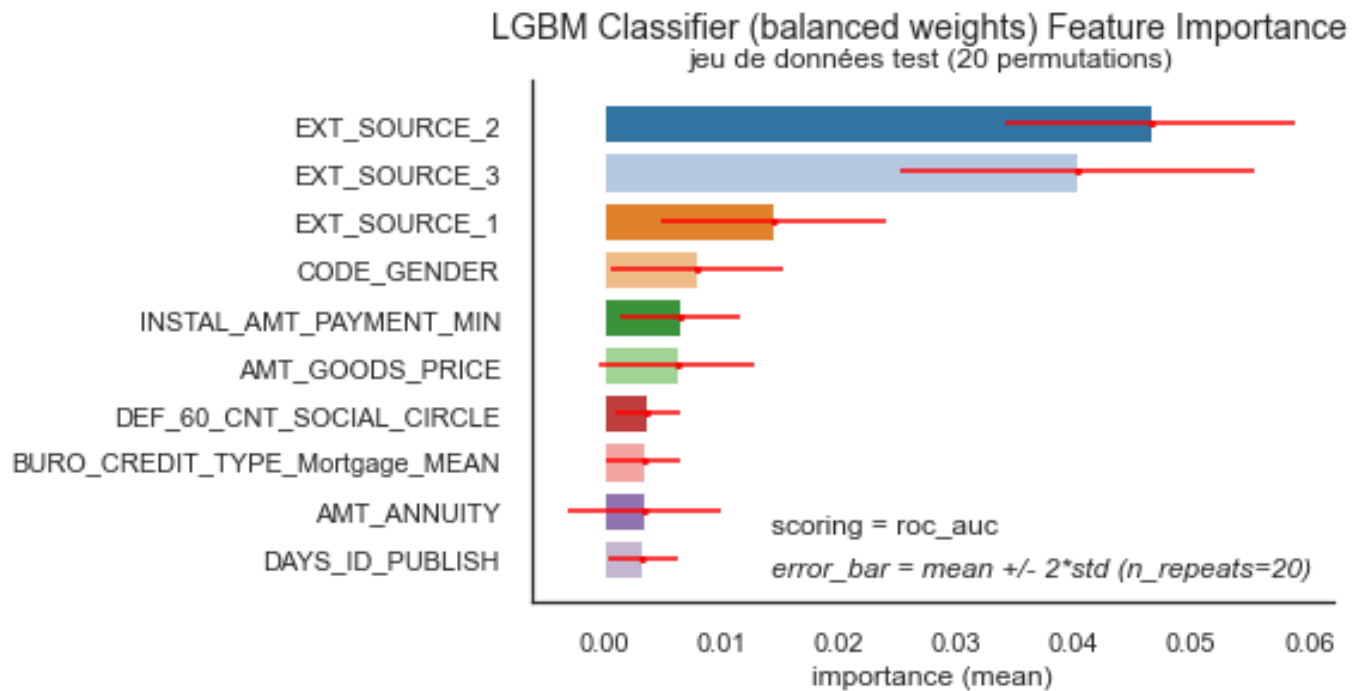
Pour optimiser le bénéfice pour la banque
l'ajustement du threshold de risque dépend du:

- coût de prêter à un mauvais payeur ($fn_loss = -10$) et
- coût de ne pas prêter à un bon client ($fp_loss = -0.5$)

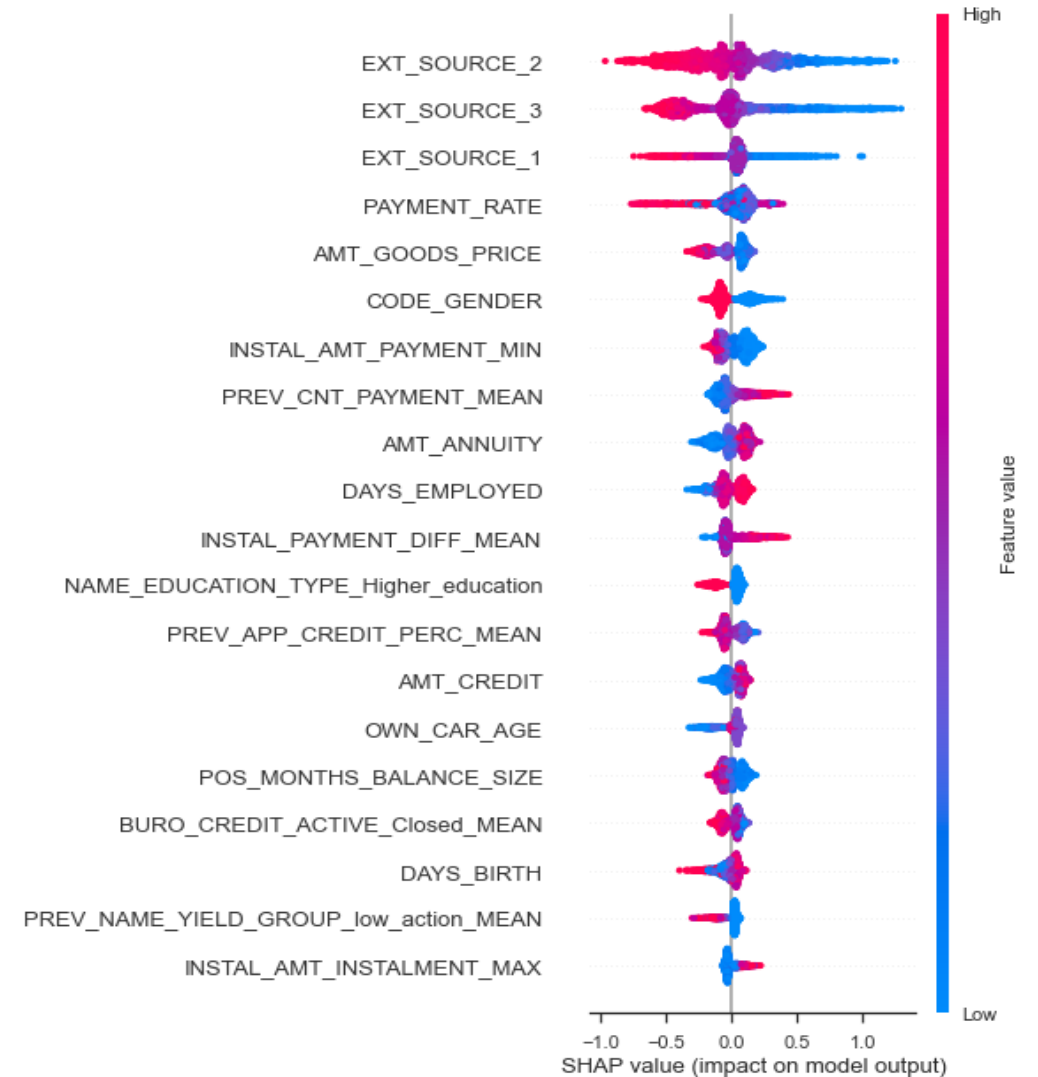


Feature Importances – interprétabilité globale

- Permutation importance



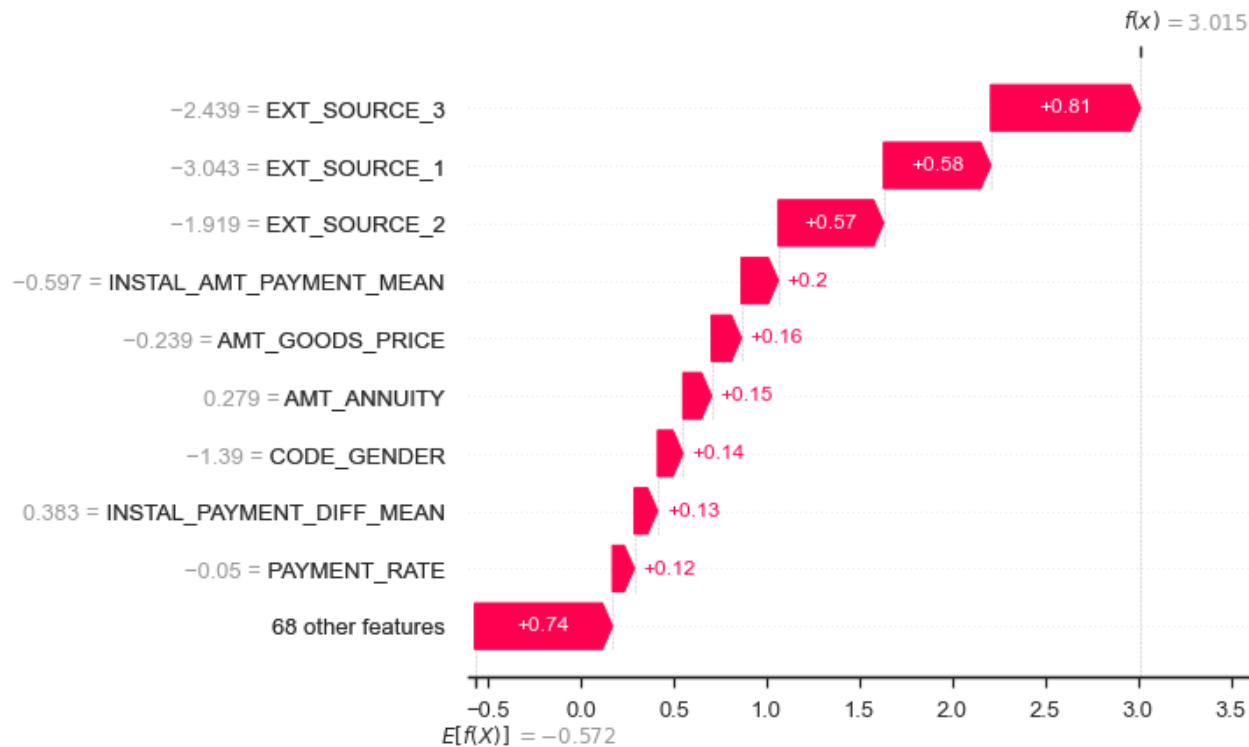
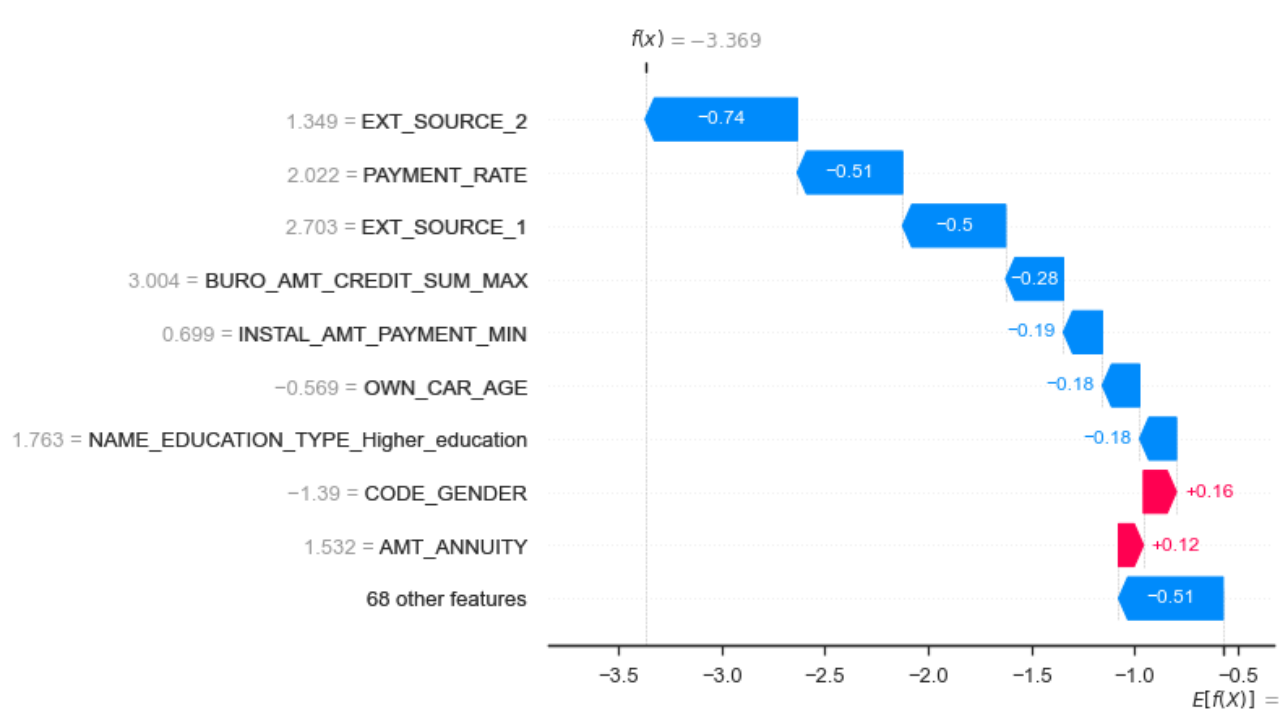
- SHAP – global feature importance



Interprétabilité locale – SHAP values

- Client ayant très peu de risque ($p=0.03$)

- Client ayant beaucoup de risque ($p=0.95$)



SHAP (Shapley Additive exPlanations)

04 Dashboard

API du dashboard

Fonction

- Gérer l'accès aux prédictions
- Gérer l'accès aux données (interprétabilité)
- Fournir un interface public pour plusieurs clients

REST API

- Réponses json aux requêtes GET

Code source dans dossier **api** sur dépôt :
<https://github.com/mrcreasey/oc-ds-p7-scoring-dashboard>

Instructions pour développement

- Voir README.md dans dossier **api**

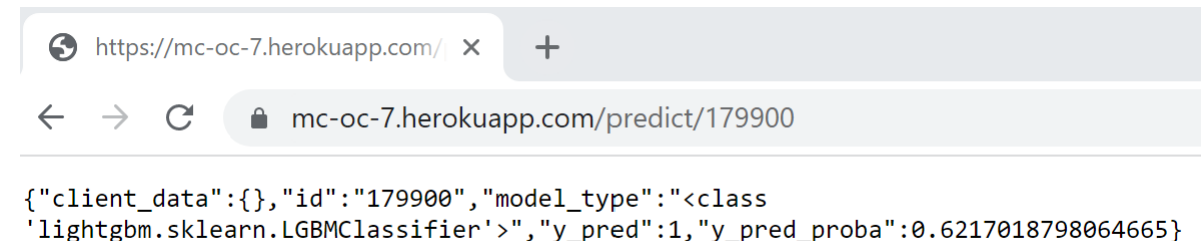
Flask application




Déploiement sur heroku



- <https://mc-oc-7.herokuapp.com/>



Visualisation du dashboard

- Visualisation des prédictions et données fournies par l'api
- Ecrit avec streamlit  Streamlit

Lien vers dashboard déploiement:

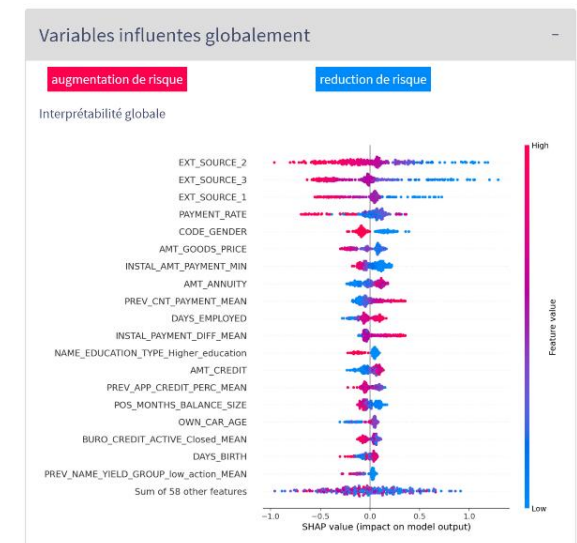
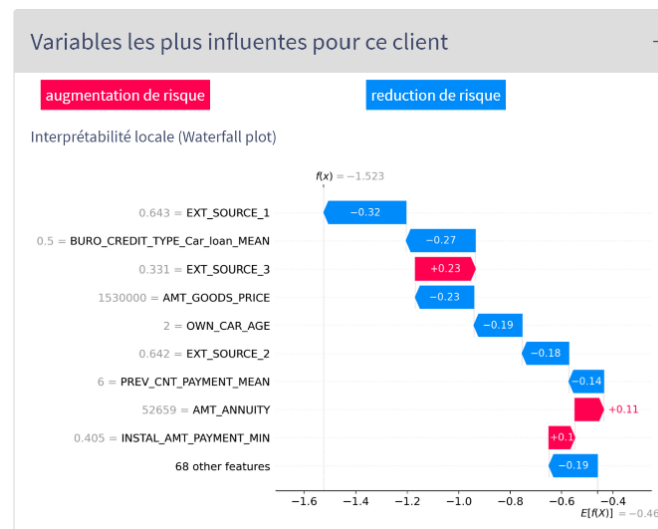
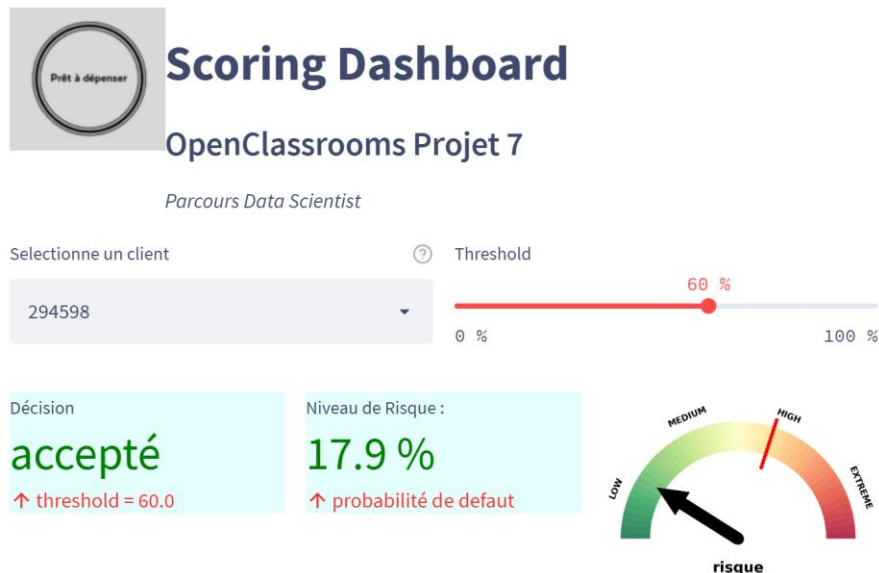
- <https://mrcreasey-oc-ds-p7-scoring-dashboard-dashboardmain-70agjx.streamlitapp.com/>

Code source dans dossier **dashboard** sur dépôt :

- <https://github.com/mrcreasey/oc-ds-p7-scoring-dashboard>

Instructions pour développement

- Voir README.md dans dossier **dashboard**



05 Conclusion et améliorations à faire

Conclusions

Meilleur modèle

- LGBMClassifier (class-weights='balanced'),
- Performance
 - ROC_AUC = 0.78

Fonction coût métier

- Vrais coûts des pertes à paramétrer.
- Bénéfice Optimum pour la banque
 - Discrimination threshold = 0.6

Déploiement

- API Flask sous heroku
- dashboard sous streamlit

Limites

- Performance très sensible à feature engineering / feature sélection
- SMOTE ne semble pas le plus adapté pour les grands jeux de données
 - très lent à créer des nouveaux points synthétiques
 - besoin de calculer les voisins sur toutes les colonnes
- Besoin de travailler avec un sous-échantillon de données

Améliorations à faire

Modélisation

- Feature création avec experts du métier
- Revoir stratégie pour traiter les valeurs manquantes
- Sélection de features à chaque modèle (Wrapper/Embedded)
- Optimiser la taille des échantillons
 - learning curves des modèles
- Explorer hyperparamètres des modèles

Déploiement

- Changer de Flask API vers fastapi
- Requirements.txt différents pour code, api, dashboard
 - Dépôts github ou branches séparées
- Ajouter authentification (accès au dashboard)
- Ajouter encryptage des données client
- Stocker les données client séparément de l'API - par exemple dans un S3 bucket sur AWS
- Situer le client dans les distributions pour les features les plus importantes

Questions

images: Mark Creasey

- mrcreasey@gmail.com

Code source: <https://github.com/mrcreasey/oc-ds-p7-scoring-dashboard>

- Merci !