Prêt à dépenser

Note méthodologique N°2 - Optimisation du modèle

Fonction coût métier & métrique d'évaluation

L'objectif du projet est de résoudre un problème de classification. Il s'agit de prévoir si un crédit appartient à la classe 0 (crédit accepté) ou bien à la classe 1 (crédit refusé).

Matrice de confusion

	crédit accepté (prédit 0)	crédit refusé (prédit 1)
crédit remboursé (classe 0)	Vrais négatifs	Faux positifs (à minimiser également)
crédit en défaut (classe 1)	**Faux négatifs** **(à minimiser en priorité)**	Vrais positifs

Nous sommes face à un problème d'optimisation dont les contraintes sont les suivantes:

- Le principal objectif est de **minimiser les faux négatifs**, cad les crédits prévus comme appartenant à la classe 0 (crédits acceptés) alors qu'ils appartiennent en fait à la classe 1 (ne seront pas remboursés). Il s'agit donc de **maximiser le pourcentage de vrais positifs.** C'est ce que fait le recall.

$$Recall = \frac{vrais\ positifs}{vrais\ positifs + faux\ négatifs}$$

- Refuser un crédit à un client qui l'aurait remboursé (faux positif) fera perdre du chiffre d'affaires à la société. Compte tenu de cette contrainte commerciale, il faut également **minimiser le taux de faux positifs**, soit maximiser la précision:

$$\text{Précision} = \frac{vrais\ positifs}{vrais\ positifs + faux\ positifs}$$

La priorité est donc de **minimiser le taux de faux négatifs.** Ces faux négatifs sont très pénalisants car un défaut de crédit coûte beaucoup plus cher que le coût d'opportunité d'un crédit non accordé (faux positif). Dans la fonction de coût, il faut pénaliser davantage les faux négatifs. Nous avons retenu des poids de 11 pour les faux négatifs contre 1 pour les faux positifs, soit une proportion équivalente à celle des classes 0 et 1 dans l'échantillon. Nous avons également testé la fonction F-Beta score de scikit-learn (metrics.fbeta_score). Le paramètre beta permet de fixer le poids accordé au recall par rapport à la précision.

Algorithme d'optimisation

Parmi les algorithmes testés affichant de bonnes performances (voir note N°1) , nous avons cherché à optimiser les hyperparamètres de régression logistique en utilisant les fonctions RandomizedGridSearchCV et GridsearchCV de scikit learn. Les régressions logistiques présentent plusieurs avantages. Elles permettent d'obtenir de bons scores de recall, sont facilement compréhensible et présentent l'avantage d'être rapide à entraîner.

Néanmoins, les modèles XGBoost permettent d'atteindre des performances supérieures pour des temps de calcul convenables comparé aux Random Forests.

Au final, nous sommes parvenus à atteindre un recall sur le test set d'environ 0.80 et un AUC ROC Curve de même niveau. Le score de crédit calculé (pourcentage de faux positifs et faux négatifs) descend lui sous 40%.

Nous avons enfin cherché à optimiser le seuil de probabilité à utiliser pour prévoir la classe 1. Par défaut, ce seuil est fixé à 50%. Avec un modèle XGBoost optimisé, nous avons réussi à légèrement améliorer la performance du modèle avec un seuil très légèrement inférieur (49.55%).