P7 : Implémentez un modèle de scoring

...

06 Décembre 2022

Asmae RAJI KARROUCHI

Objectif du projet :
Créer un modèle de scoring et le publier en ligne avec un dashboard

Problématique

Problématique

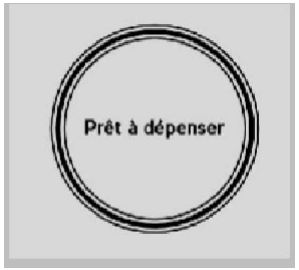
Problème :

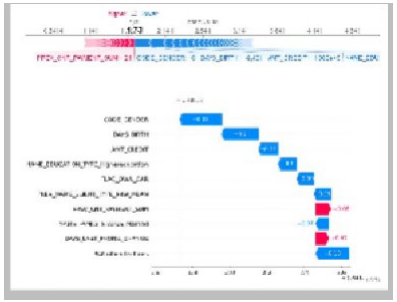
- Une banque nous demande de déterminer si un client va réussir ou non à rembourser son prêt

- On nous demande une API ainsi qu'un dashboard

Objectif :

- Créer un modèle prédictif qui nous donne un score par client
- Créer une API et un Dashboard accessibles et compréhensibles en ligne pour les employés





Dataset

Dataset - Description

Kaggle - Home Credit Defaut Risk :

- Ces datas sont présentes sur Kaggle sous la forme d'une compétition
- On possède 10 fichiers
- Une partie (application_test) n'est pas utilisable pour entraîner le modèle car on ne connaît pas la variable cible de ces entités

Feature Engineering

Feature Engineering

Utilisation d’un Kernel :

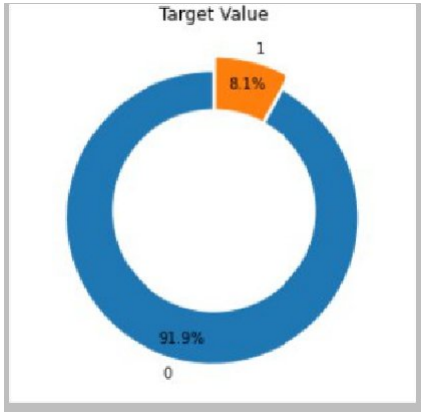
- Pour gagner du temps, nous n’effectuerons pas le cleaning et feature engineering
- On utilise du code déjà rédigé par un utilisateur Kaggle qui obtient par la suite de bons résultats
- Cela nous donne un seul dataset assez conséquent (307507 lignes × 797 colonnes)
- On clean quand même ce dataset (valeurs nulles notamment)
- Résultat : 267148 lignes × 357 colonnes

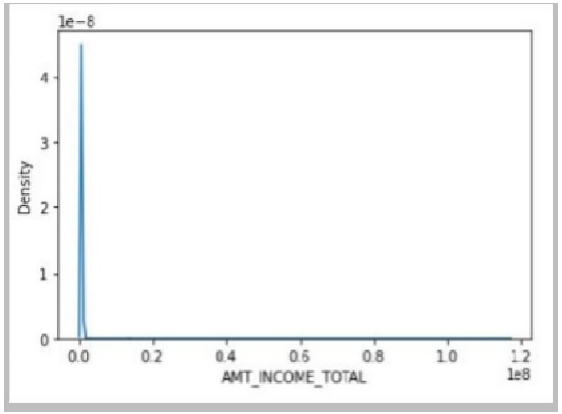
Analyse Exploratoire

Analyse Exploratoire

Analyse de graphs

- On voit plusieurs graphs intéressant
- Le premier nous montre que notre TARGET est déséquilibrée
- Le deuxième nous montre une répartition des salaires “skewed”





Tests des différents modèles

Présentation des modèles

Cost-Sensitive Logistic Regression

- Cost-Sensitive pour le problème de dataset déséquilibré
- Modèle très courant pour obtenir une probabilité qu'un évènement se produise

Cost-Sensitive Decision Trees

- Cost-Sensitive pour le problème de dataset déséquilibré
- Utilisation d'arbres de décision avec les variables du dataset pour arriver à un résultat

Présentation des modèles

Cost-Sensitive XGBoost

- Cost-Sensitive pour le problème de dataset déséquilibré
- Version accessible du Gradient Boosting, méthode qui va utiliser plusieurs arbres de décision ensemble
- Très lent

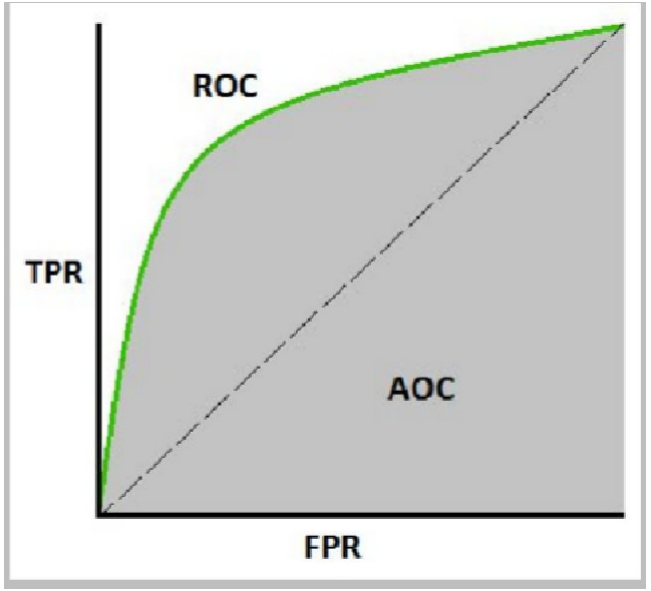
Cost-Sensitive LightGBM

- Cost-Sensitive pour le problème de dataset déséquilibré
- Version légère du Gradient Boosting. Diffère dans sa manière d'utiliser les arbres de décision
- 10x plus rapide que le XGBoost

Métriques utilisées - AUC - ROC

Définition

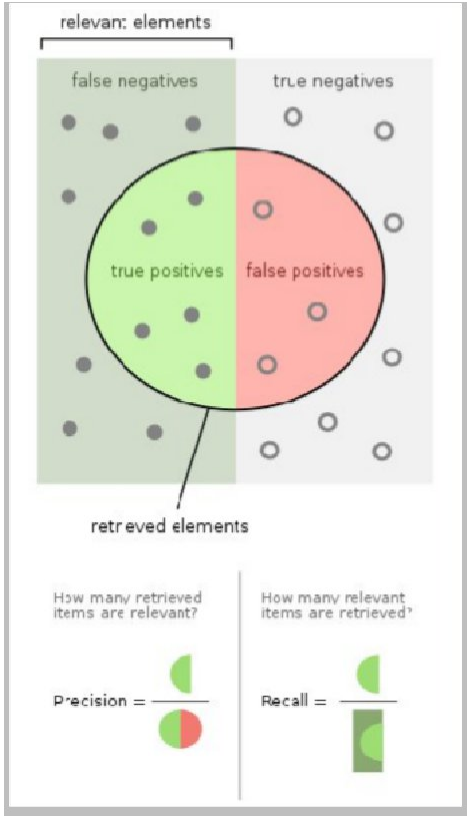
- ROC (Receiver Operating Curve) est formée en mettant le False Positive Rate en abscisse et le True Positive Rate en ordonnée
- AUC (Area Under the Curve) est calculée en prenant l'aire totale sous la courbe (les max étant à 1 sur les deux axes, on obtient un chiffre entre 0 et 1)
- Nous montre à quel point notre modèle distingue les deux classes



Métriques utilisées - Recall (Sensibilité)

Définition

- Donne le taux de vrais positifs que l'on a réussi à deviner (TP/TP+FN)
- Dans notre cas, le nombre de clients qu'on a deviné "mauvais pour le business" et qui le sont réellement sur tous ceux qui le sont réellement
- Faire attention aux termes "Négatif" et "Positifs" qui veulent ici dire le contraire du vocabulaire métier



Métriques utilisées - F-Scores

Définition

- Calcule la justesse d'un modèle en utilisant la précision et le recall
- Le f Beta Score introduit un paramètre Beta qui nous permet de modifier l'importance de la précision ou du recall
- Ici, on teste avec un f10-Score

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

Modélisation - Résultats

Comparaison des 4 modèles sur l'AUC-ROC

- On utilise des Cross Validations pour essayer de trouver grossièrement les bons hyperparamètres sur chaque modèle.
- De nouveau, XGBoost est très lent
- Light GBM ressort comme le meilleur

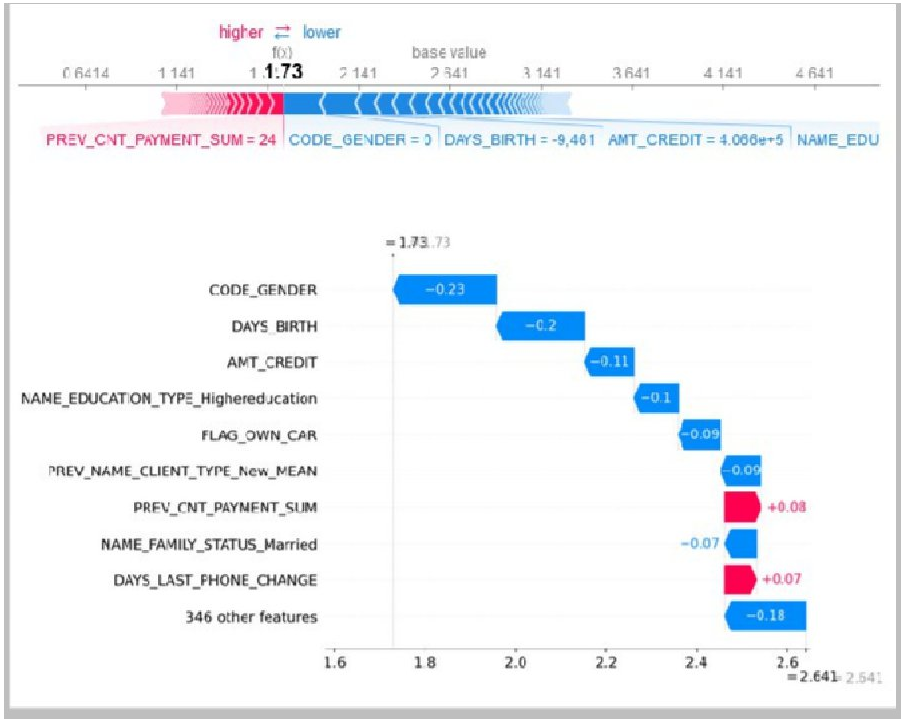
Modèle	Temps moy. Training (s)	Temps moy. Fitting (s)	Temps moy. Scoring (s)	Score de Test
Logistic Regression	0,58	7,98	0,05	0,53
Decision Tree	0,61	1,29	0,05	0,58
XGBoost	0,71	45,31	0,17	0,64
Light GBM	0,71	4,98	0,32	0,64

Feature Importance

SHAP

Définition

- shap est une librairie qui permet de visualiser une explication d'un modèle prédictif
- Elle nous offre plusieurs graphs comme le "force plot" et le "waterfall"



Dashboard & API

Dashboard & API - Choix Techniques

Librairies et outils choisis:

1. Explication visuelle du modèle : shap (https://shap.readthedocs.io/)
2. Créer une API : FastAPI (https://fastapi.tiangolo.com/)
3. Publier l'API : Uvicorn, Gunicorn (https://www.uvicorn.org/)
4. Héberger l'API (et le dashboard) : Heroku (https://www.heroku.com/)
5. Créer le dashboard : Streamlit (https://streamlit.io/)

API

Définition

- Application qui permet de communiquer entre deux serveurs / un serveur et un utilisateur.
- Ici, nous permet de récupérer une prédiction faite par notre modèle en ligne
- https://asmaekarouchi-api.herokuapp.com/

Résultats API et Utilisation

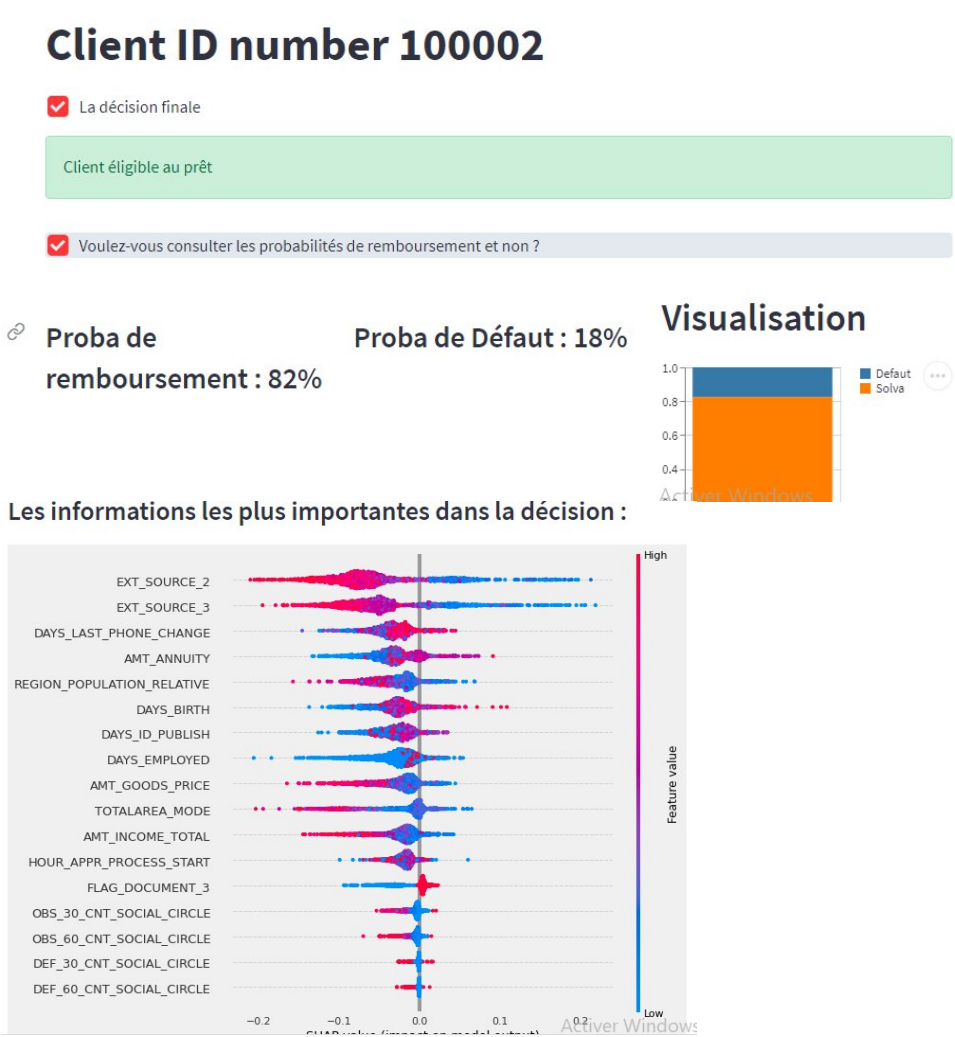
Lien de l'API
https://asmaedashboard1.herokuapp.com/

```
@st.cache
def load_prediction(sk_id):
    with urllib.request.urlopen(DATA_URL + str(sk_id)) as url:
        data = json.loads(url.read().decode())
    real_data = float(data['message'].split(' ')[0].replace('[',''))
    return real_data
```

Dashboard

Features

- Utilise l'API
- Permet de choisir son client
- Montre le score de prédiction que le client rembourse son prêt.
- Montre des explications visuelles de pourquoi avec shap
- Montre ou se trouve le client par rapport aux autres sur une variable



Améliorations possibles

Points d’améliorations

- Une plus grosse mémoire serveur permettra d’utiliser toutes les données
- Continuer le travail sur les hyperparamètres / trouver un autre modèle plus performant
- Personnaliser le pré-traitement des données

Conclusion

- Notre modèle est le LightGBM est obtient un f10-score de 0.53
- Une API et un Dashboard on été mis en ligne
- Plusieurs problèmes ont été rencontrés et plusieurs axes d’amélioration sont envisageable