

Projet 7 : Implémentez un modèle de scoring

Lancelot LECLERCQ

5 mai 2022

Sommaire

1. Introduction
2. Analyse et traitement des données
3. Optimisation du modèle
4. Explication du modèle
5. Déploiement sur le cloud

Introduction

Problématique

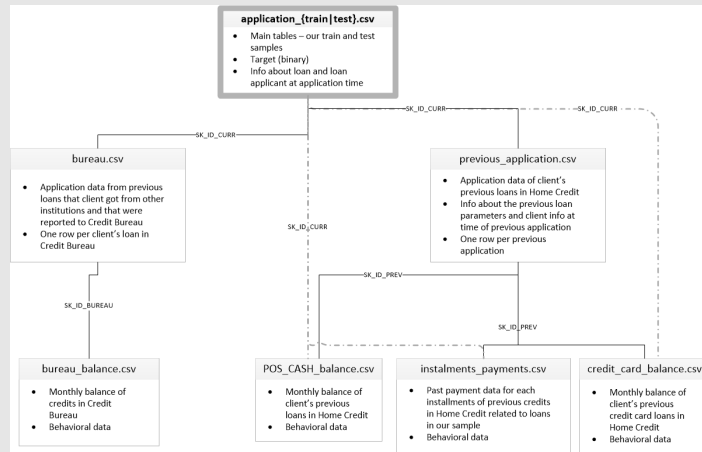
- L'entreprise Prêt à dépenser est une société financière qui propose des crédits à la consommation pour des personnes ayant peu ou pas du tout d'historique de prêt



- Objectifs
 - Mettre en œuvre un outil de “scoring crédit” pour calculer la probabilité qu'un client rembourse son crédit
 - Classifier la demande en crédit accordé ou refusé
 - développer un algorithme de classification en s'appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.)
 - Développer un dashboard interactif
 - expliquer de façon la plus transparente possible les décisions d'octroi de crédit,
 - permettre aux clients de disposer de leurs informations personnelles et de les explorer facilement

Données

- Principal fichier utilisé
application_{train||test}.csv



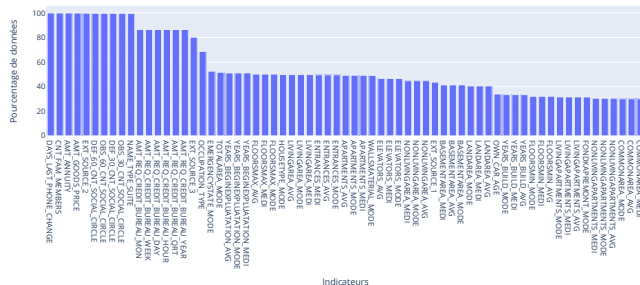
Analyse et traitement des données

Analyse et traitement des données

Exploration du jeu de données

- Certaines colonnes comportent un grand nombre de données manquantes
 - Nous utiliserons des modèles résistants à ces données manquantes comme XGBoost et LightGBM
- Encodage des variables catégorielles
 - par LabelEncoder pour les variables ayant 2 catégories
 - par `pandas.get_dummies()` pour les variables ayant plus de 2 catégories

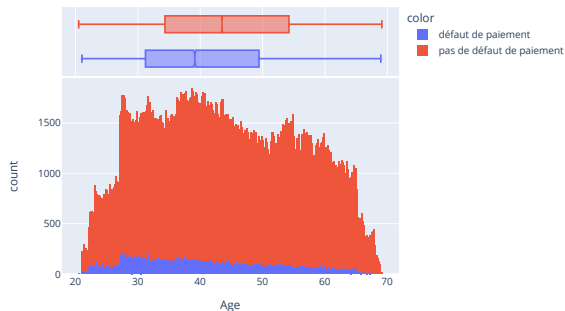
Pourcentage de données par colonnes comportant des données manquantes



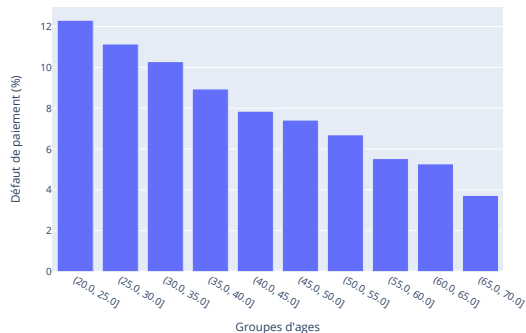
Analyse et traitement des données

Exploration du jeu de données

Histogramme du nombre de clients ayant ou non fait défaut en fonction de leur âge



Pourcentage de défauts de paiement en fonction des catégories d'âges

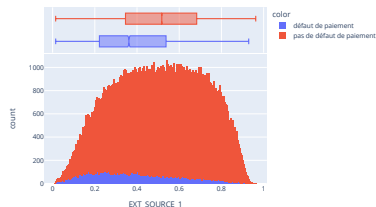


- L'âge des clients semble avoir un impact sur le fait que le client fasse défaut ou non

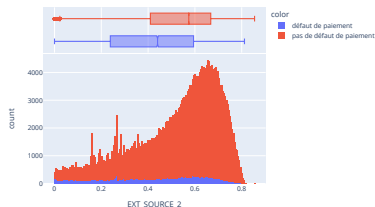
Analyse et traitement des données

Exploration du jeu de données

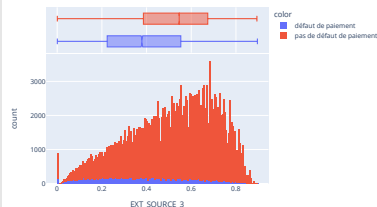
Histogramme du nombre de clients ayant ou non fait défaut en fonction de EXT_SOURCE_1



Histogramme du nombre de clients ayant ou non fait défaut en fonction de EXT_SOURCE_2



Histogramme du nombre de clients ayant ou non fait défaut en fonction de EXT_SOURCE_3



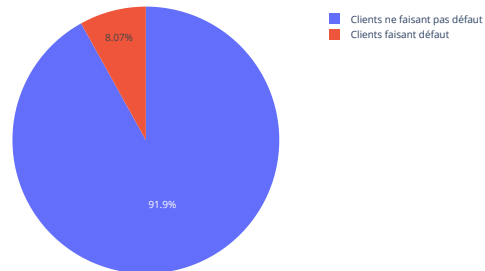
- Les données EXT_SOURCE semblent aussi avoir une certaine corrélation avec le fait que le client fasse défaut

Analyse et traitement des données

Exploration du jeu de données

- Données sont déséquilibrées : clients faisant défauts peu nombreux par rapport à ceux ne faisant pas défaut
- Classer tous les clients comme ne faisant pas défaut
⇒ score honorable avec seulement 8% d'erreurs
- Utilisation de la librairie imblearn ⇒ rééchantillonnage de notre jeu de données

Part de clients faisant ou non défaut
dans la colonne cible (TARGET)



Analyse et traitement des données

Rééchantillonnage du jeux de données

Réalisation d'une régression logistique pour essayer les différentes méthodes de rééchantillonnage

- Méthodes de sous-échantillonnages :

- On réduit le nombre de clients ne faisant pas défaut pour en avoir autant que de client faisant défaut
- RandomUnderSampler choisi ces derniers au hasard
- TomekLinks conserve un certain nombre de clients par groupe de clients similaire (repose sur les KNN)

- Méthodes de sur-échantillonnages :

- On multiplie le nombre de clients faisant défaut pour en avoir autant que des clients ne faisant pas défaut
- RandomOverSampler dédouble des clients faisant défaut au hasard
- SMOTE créé de nouveaux clients à partir de groupe de clients similaires

- Méthodes combinant le sur- et le sous-échantillonnage

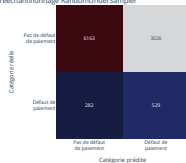
- SMOTEENN
- SMOTETomek

Analyse et traitement des données

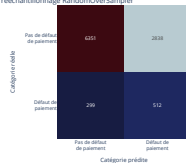
Rééchantillonnage du jeux de données

	Prédit	
	0	1
Réal	0	TN FP
	1	FN TP

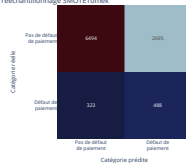
Matrice de confusion de la classification par régression logistique et rééchantillonnage RandomUnderSampler



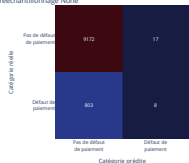
Matrice de confusion de la classification par régression logistique et rééchantillonnage RandomOverSampler



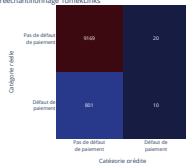
Matrice de confusion de la classification par régression logistique et rééchantillonnage SMOTETomek



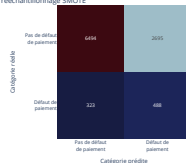
Matrice de confusion de la classification par régression logistique et rééchantillonnage None



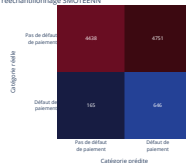
Matrice de confusion de la classification par régression logistique et rééchantillonnage TomekLinks



Matrice de confusion de la classification par régression logistique et rééchantillonnage SMOTE



Matrice de confusion de la classification par régression logistique et rééchantillonnage SMOTEENN



- TomekLinks n'est pas plus efficace que sans rééchantillonnage
- SMOTEENN fait un grand nombre de faux positifs
- SMOTEENN a le plus grand nombre de vrais positifs (TP)

Optimisation du modèle

Optimisation du modèle

GridSearch

Pour chaque solution de rééquilibrage :

RandomOverSampler, SMOTE, RandomUnderSampler, TomekLinks, SMOTEENN, SMOTETomek



GridSearch pour les deux modèles :

XGBoost Classifier, LightGBM Classifier



Nous retenons le meilleur résultat pour un couple : solution de rééquilibrage/modèle

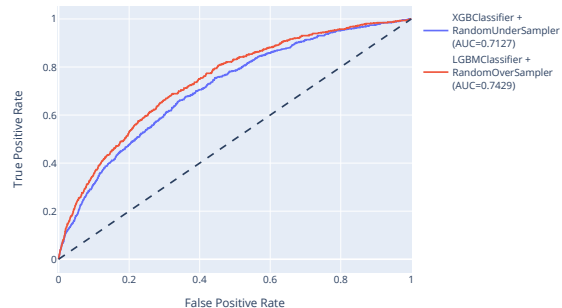
Nous avons fait différents essais d'optimisation

- 1^{er} essai avec optimisation de l'AUC
- 2nd essai avec optimisation de la métrique métier (F_β -score)

Optimisation du modèle

Courbe ROC

- La courbe ROC représente les vrais positifs en fonction des faux positifs
- Plus la courbe est proche du coin supérieur gauche meilleur est le modèle
- L'aire sous la courbe (AUC) nous donne une valeur numérique pour comparer ces modèles

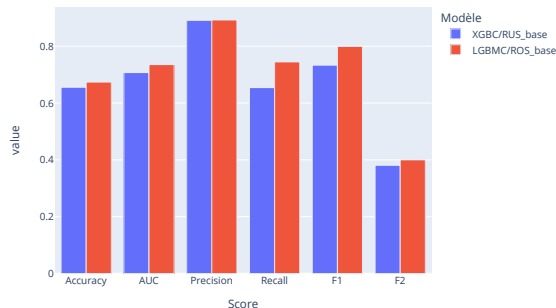


Optimisation du modèle

Différentes métriques utilisées

- Accuracy : précision de la classification (somme des éléments bien classé sur le nombre total d'éléments)
- AUC : Area Under the Curve, aire sous la courbe ROC
- Precision : part de vrais positifs dans les prédictions positives
- Recall : part de vrais positifs dans les éléments réellement positifs
- F1 : moyenne harmonique de la précision et du rappel
- F2 : idem F1 avec un facteur $\beta=2$, permettant de mettre plus de poids sur le rappel

Scores pour les différents modèles sans polynomial features

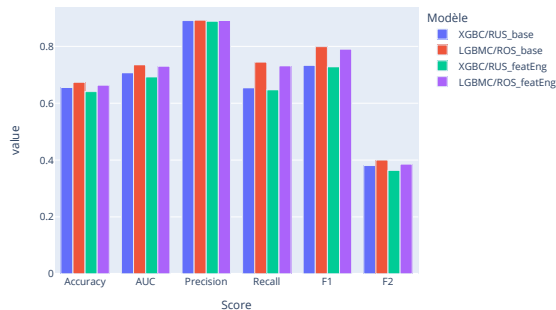


Optimisation du modèle

Création de variables polynomiales

- Afin d'améliorer les scores des modèles nous avons essayé de créer des variables polynomiales à partir des colonnes les plus corrélées avec la cible
- L'amélioration n'est pas pertinente nous n'avons donc pas conservé ces variables pour notre modèle final

Scores pour les différents modèles



Optimisation du modèle

Métrique métier

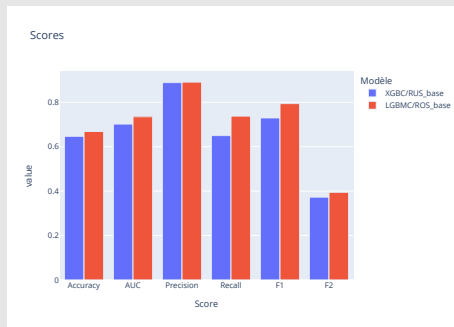
- But

- Diminuer le nombre de faux négatifs (prédit 0, réel 1) afin d'éviter de manquer des clients qui pourraient potentiellement faire défaut
- Améliorer le recall

		Prédit		
		0	1	
Réal	0	TN	FP	Precision
	1	FN	TP	
		Recall		

- Outil

- Utilisation du F_β -score qui permet d'ajouter du poids respectivement au rappel lorsque le facteur β est >1 ou à la précision lorsque le facteur β est <1
- Utilisation de $\beta=2$



Explication du modèle

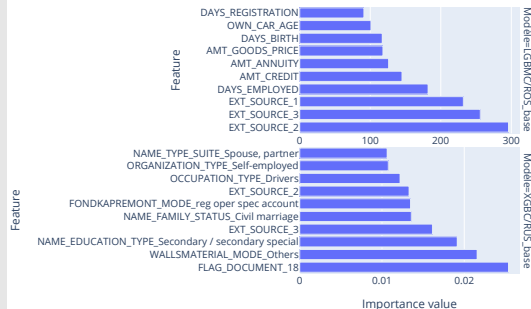
Explication du modèle

Importance des variables

- Les modèles nous rendent compte de la part des variables dans leurs résultats

Les variables EXT_SOURCE se retrouve dans les deux cas par exemple

Importances des variables utilisées par les différents modèles

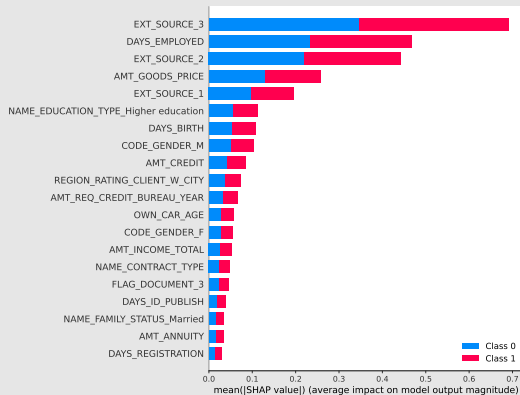


Explication du modèle

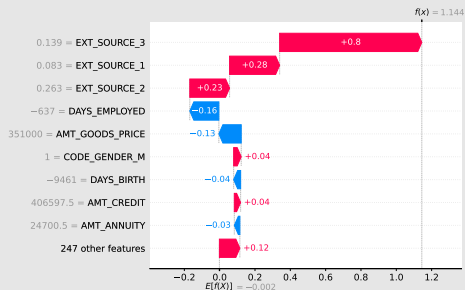
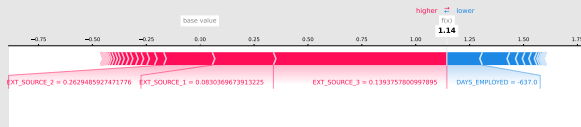
Utilisation de SHAP

- La librairie SHAP permet d'expliquer le fonctionnement du modèle de manière plus poussée.
- Exemples sur le modèle LightGBM :

Fonctionnement global du modèle \Rightarrow part des variables utilisées lors du classement en général



Fonctionnement local du modèle \Rightarrow part des variables utilisées lors du classement d'un élément particulier



Déploiement sur le cloud

Déploiement sur le cloud

API

- Utilisation de Flask
- Entraînement du modèle et prédictions
- Une URL pour chaque type de données avec une ou plusieurs clés permettant des requêtes sur des clients ou des valeurs particulières
- Exemple pour les données des clients pour lesquelles on renseigne l'id du client :
`http://localhost:5000/ID_clients/infos_client?id=<identifiant>`

```
@app.route("/ID_clients/infos_client/", methods=["GET"])
def show_data():
    ID_client = request.args.get("id", default=100001, type=int)
    data_client = app_test[app_test.SK_ID_CURR == int(ID_client)].set_index('SK_ID_CURR')
    data_reponse = json.loads(data_client.to_json(orient='index'))
    return jsonify(data_reponse)
```

Déploiement sur le cloud

Dashboard

- Utilisation de Dashboard
- Requêtes à l'API afin d'obtenir les données au format JSON
- Exemple pour une requête concernant les données d'un client :

```
URL_API = 'http://localhost:5000/'
```

```
@app.callback(Output('infos_client', 'data'), Input('ID_choosed', 'value'))
def get_client_infos(idclient):
    url = URL_API + 'ID_clients/infos_client/?id=' + str(idclient)
    data = requests.get(url).json()
    return data
```


Déploiement sur le cloud

Déploiement

- Déploiement sur Heroku

- `heroku git:remote -a bank-scoring-dash`
- `git add .`
- `git commit -am 'launch in the cloud'`
- `git push heroku master`

