Projet 7 : Implémentez un modèle de scoring

Lancelot LECLERCQ

5 mai 2022

Sommaire

Introduction

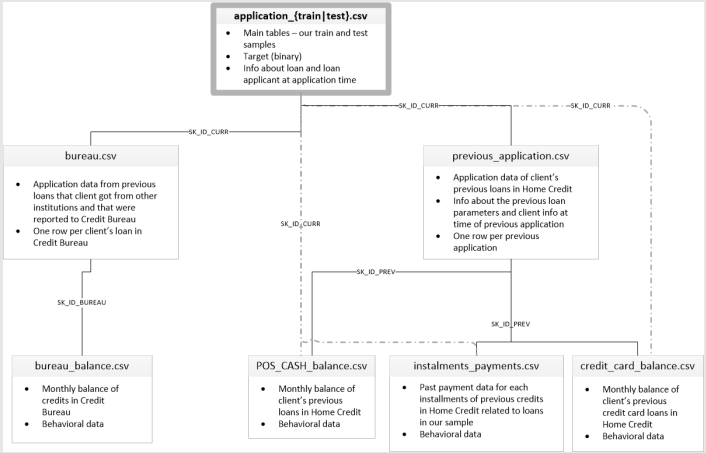
Problématique

- 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

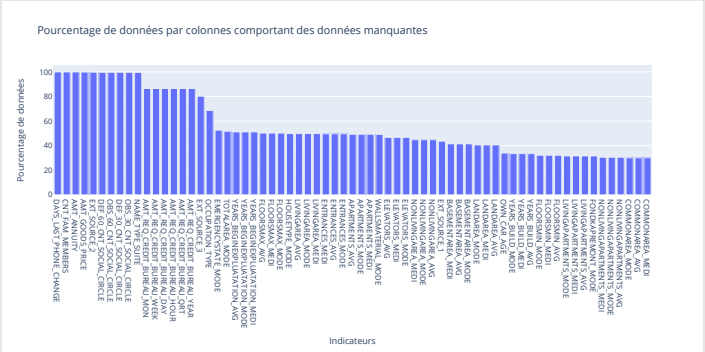


Analyse et traitement des données

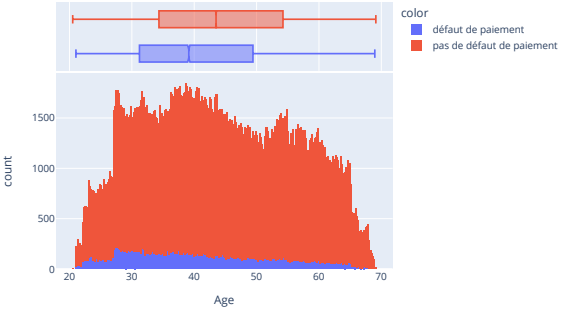
Analyse et traitement des données
Exploration du jeu de données

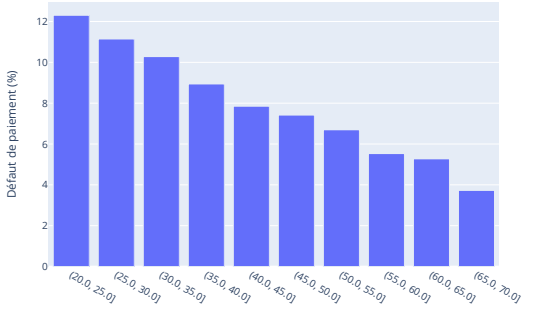
- Certaines colonnes comportent un grand nombre de données manquantes
  - Nous utiliserons des modèles résistants à ces données manquantes comme XGBoost et LightGBM

- Encodage des variables catégorielles
  - par LabelEncoder pour les variables ayant 2 catégories
  - par pandas.get_dummies() pour les varibles ayant plus de 2 catégories



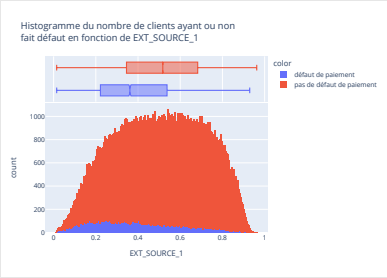
Analyse et traitement des données
Exploration du jeu de données

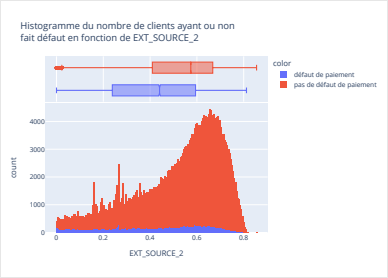


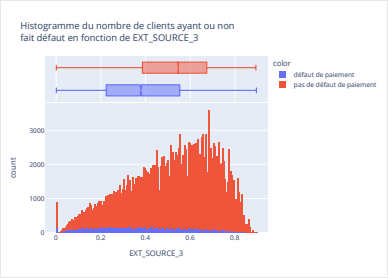


- L'age des clients semble avoir un impact sur le fait que le client fasse défaut ou non

Analyse et traitement des données
Exploration du jeu de données



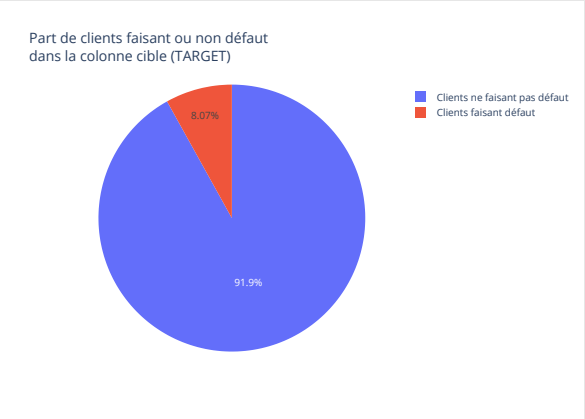




- Les données EXT_SOURCE semblent aussi avoir une certaines corrélation avec le fait que le client fasse défaut

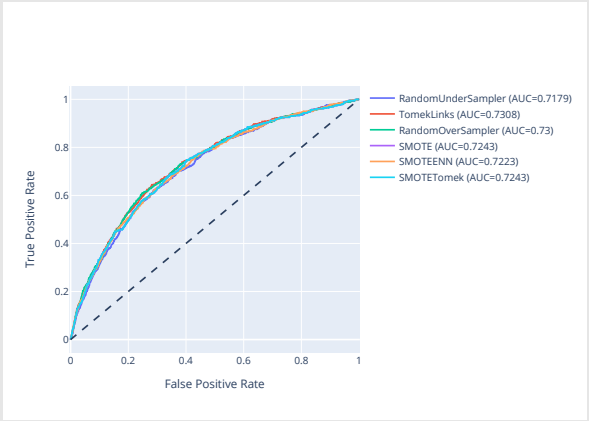
Analyse et traitement des données
Exploration du jeu de données

- Comme on a pu le voir les données sont déséqulibrées du fait que les clients faisant défauts sont peu nombreux par rapport à ceux ne faisant pas défaut
- Classer tous les clients comme ne faisant pas défaut permettrait d'avoir un score honnorable avec seulement 8% d'erreurs
- Nous avons donc utilisé la librairie imblearn qui permet de rééchantillonner notre jeu de données

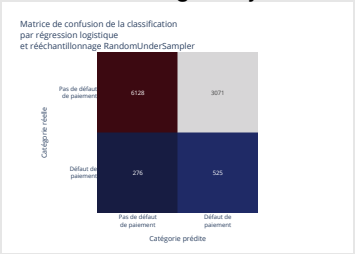


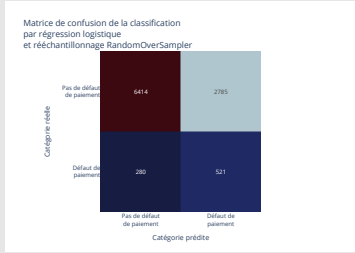
Analyse et traitement des données
Rééchantillonnage du jeux de données

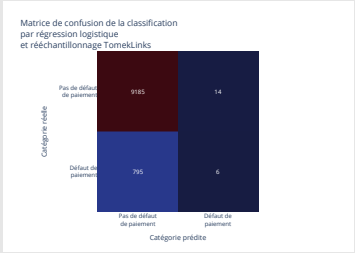
- Pour essayer les différentes méthodes de rééchantillonnage nous avons réaliser une régression logistique sur les données rééchantiollonnées avec les différents outils
  - RandomUnderSampler et TomekLinks : méthodes de sous-échantillonnages
    - on conserve le même nombre de clients ne faisant pas défaut que de client faisant défaut
    - RandomUnderSampler choisi ces dernier au hasard
    - TomekLinks conserve un certains nombre de clients par groupe de clients similaire (repose sur les KNN)
  - RandomOverSampler et SMOTE : méthodes de sur-échantillonnages
    - on multiplie le nombre de clients faisant défaut
    - RandomOverSampler dédouble des clients faisant défaut au hasard
    - SMOTE créé de nouveaux clients à partir de groupe de clients similaires
  - SMOTEENN et SMOTETomek sont des méthodes combinant le sur- et le sous-échantillonnage
- Les scores AUC semblent plutôt bon ($>0{,}7$)

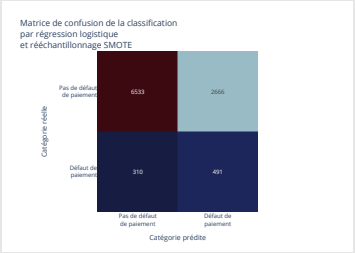


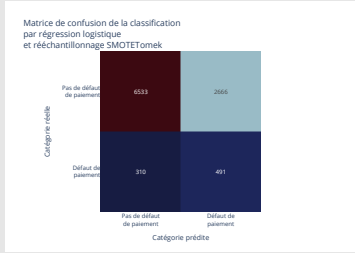
Analyse et traitement des données
Rééchantillonnage du jeux de données

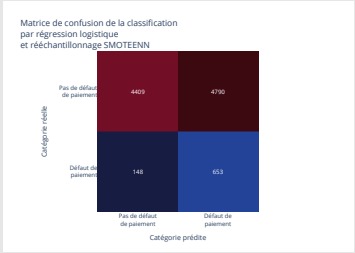












- Les 2 cas de droites ne sont pas intéressant car le premier classe tout les clients comme ne faisant pas défaut et le second classe 50/50

Analyse et traitement des données
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

Optimisation du modèle

Optimisation du modèle
GridSearch

Pour chaque solution de rééquilibrage retenues :

RandomOverSampler, RandomUnderSampler, SMOTE, SMOTETomek

⇓

GridSearch pour les deux modèles :

XGBoost Classifier, LightGBM Classifier

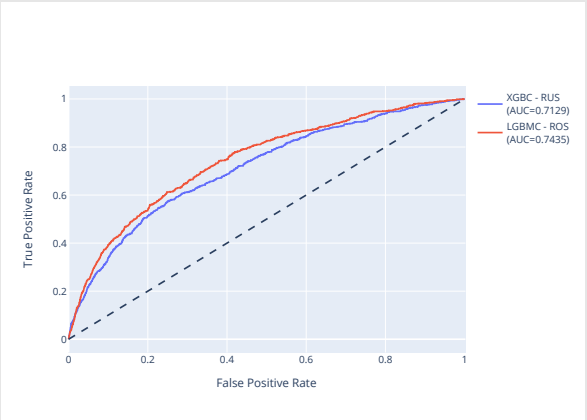
⇓

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

Nous avons fait différents essais d'optimisation avec plusieurs métriques

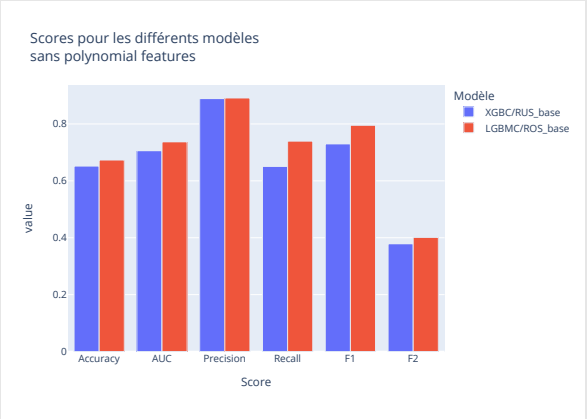
Optimisation du modèle
Courbe ROC

- 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

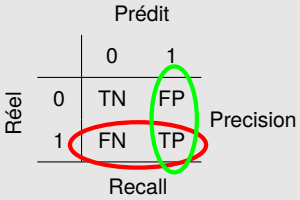
- Accuracy : précision de la classification (somme des éléments bien classé sur le nombre total d'éléments)
- AUC : Area Under the Curve, aire sous la courbe ROC
- Precision : part de vrai positifs dans les prédictions positives
- Recall : part de vrai positifs dans les éléments vraiments positifs
-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



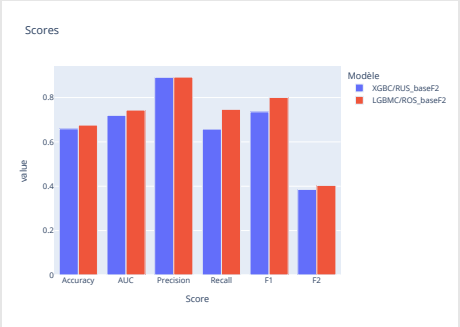
Optimisation du modèle
Métrique métier

- 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



- Outil
  - Utilisation du F_β-score qui permet d'ajouter du poid respectivement au recall lorsque le facteur β est >1 ou à la precision lorsque le facteur β est <1
  - Utilisation de β=2



Déploiement sur le cloud

Déploiement sur le cloud
API

- Utilisation de Flask
- Entrainement du modèle et prédictions
- Une URL pour chaque type de données avec une ou plusieurs clé 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'optenir les données au format JSON
- Exemple pour une requête consernant 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`

Conclusion

Conclusion

- Utilisation de données déséqulibrées
- Optimisation des modèle à l'aide d'une GridSearch
- Utilisation d'une métrique plus particulière afin d'optimiser le modèle en fonction des attentes du métier
- Il aurait pû être intéressant d'optimiser plus d'hyperparamètres afin d'améliorer un peu plus nos résultats
- Le déploiement dans le cloud permet un accès rapide à ces données concernant les clients et les prédictions qu'ils fassent ou non défaut
- Ces données sont accompagnées de graphiques permettant de comparer ces clients avec d'autres clients similaires ou à l'ensemble des clients