Implémenter un modèle de scoring

Création le 10/01/2019

Julien Di Giulio

SOMMAIRE

I - PRESENTATION

- Présentation du projet
- Plan d'actions

II- ETUDE DES DONNEES

- Présentation des données
- Présentation du Notebook Kaggle

III – MODELISATION

- Entraînement et optimisation
- Analyse des résultats

IV – DASHBOARD

- Construction en local
- Déploiement sur le Cloud

V – CONCLUSION

- Résumé
- Questions - Réponses

I - PRESENTATION

PROJET

Etude d'un modèle de Scoring

Prêt à dépenser souhaite ***développer un modèle de Scoring de la probabilité de défaut de paiement du client*** pour étayer la décision d'accorder ou non un prêt à un client potentiel.

Développement d'un dashboard

Développement d'un Dashboard interactif pour que les chargés de relation client puissent à la fois expliquer de façon la plus transparente possible les décisions d'octroi de crédit.

Le dashboard doit permettre de :

1. Visualiser le score pour chaque client
2. Visualiser des informations descriptives relatives à un client
3. Comparer les informations descriptives relatives à un client à l'ensemble des clients ou à un groupe de clients similaires

Lien données :
https://www.kaggle.com/c/home-credit-default-risk/data

Demandes et suggestion du manager

- Partir d'un kernel Kaggle pour faciliter l'étude et la préparation des données.
- Réaliser une note méthodologique expliquant en détails la construction du modèle.
- Déploiement du dashboard sur le Cloud.

PLAN D'ACTIONS

Etudes du Kernel Kaggle

- Analyse du Notebook téléchargé

Construction du modèle de prédiction

- Adaptation du Notebook avec les besoins du Projet

Note technique

- Rédaction de la note méthodologique du modèle

Dashboard

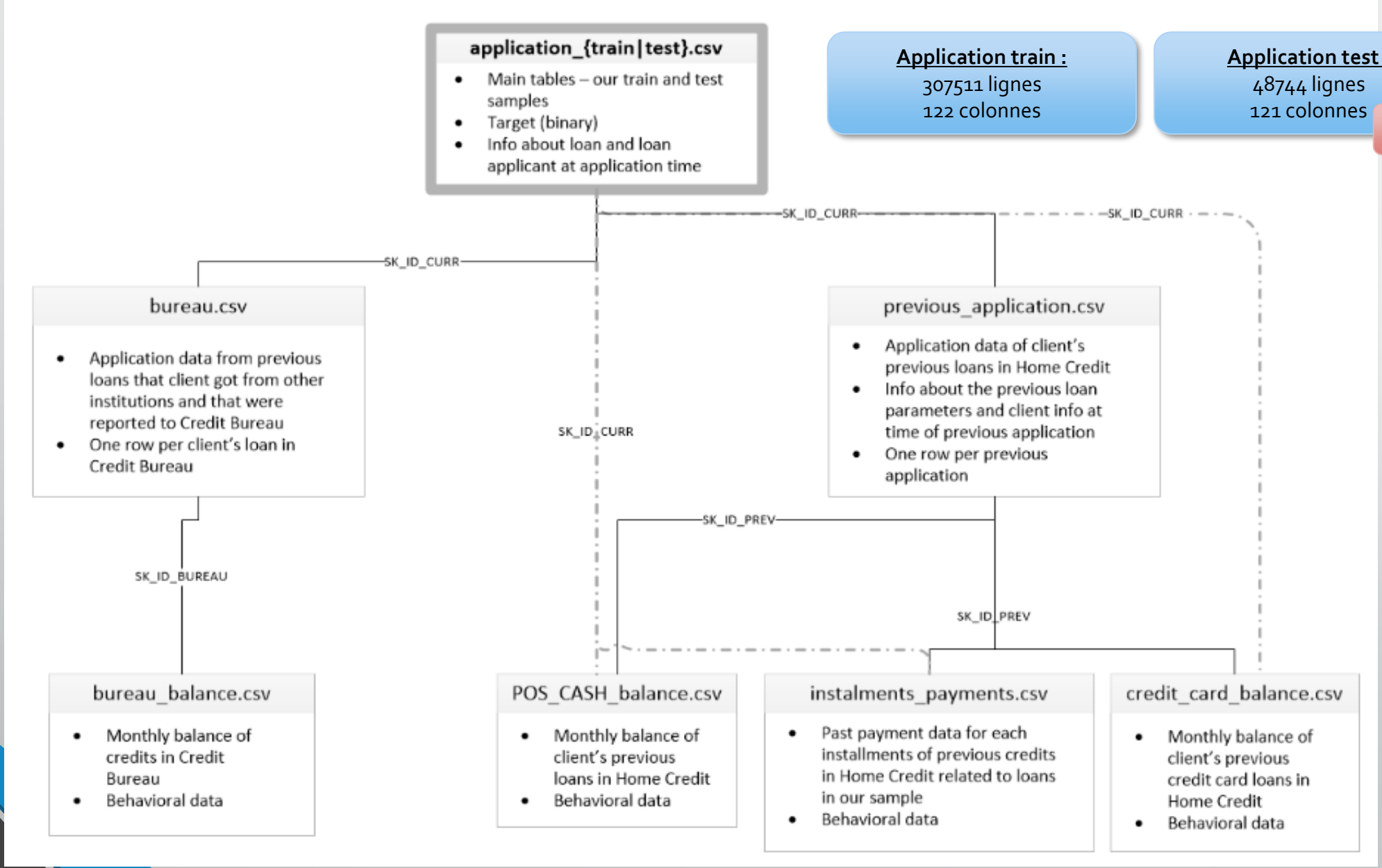
- Réalisation du dashboard
- Réalisation de l'API correspondante
- Sauvegarde sur un dépôt GitHub

Déploiement

- Déploiement du modèle sur le Cloud

II – ETUDE DES DONNEES

PRESENTATION DES DONNEES



PRESENTATION DU NOTEBOOK KAGGLE

Data train
Data test

- Rappel : "test.csv" est le dataset que nous utilisons pour simuler un nouveau client dans la base. Toutefois il convient que ces deux datasets aient la même structure à l'issu du feature engineering.

Valeurs manquantes

- Traitement par imputation de la médiane

Encodage variables

- Label encoding pour les variables à 2 catégories.
- One Hot Encoding pour les variables à plus de deux catégories.

Alignement datasets

- Alignement des datasets "train" et "test" pour conserver des structures identiques.

Création de variables

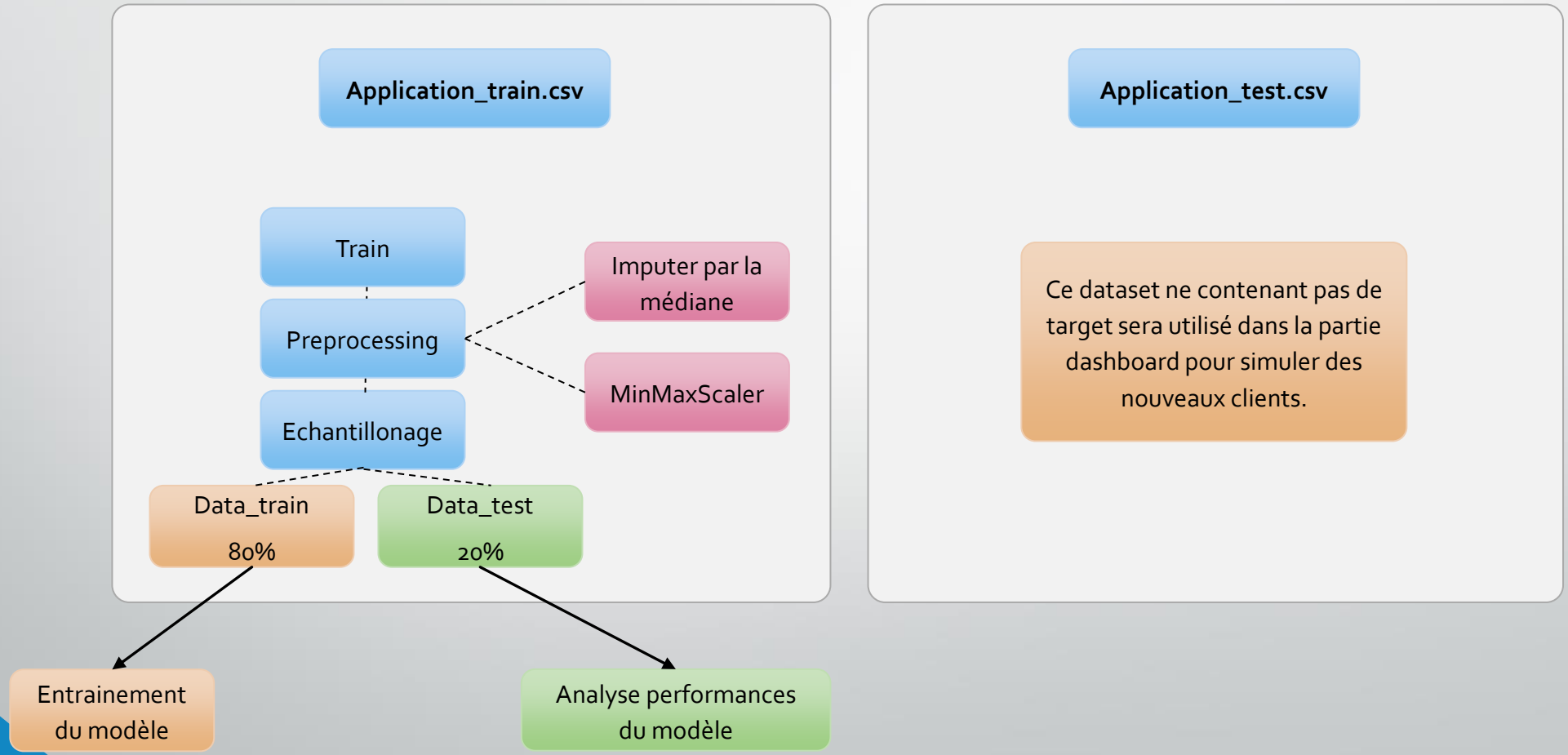
- Remplacement des outliers par des valeurs nulles. Ensuite les valeurs sont imputées par la médiane dans le Preprocessing.
- Ajout d'une "flag feature" pour identifier les lignes qui contiennent les outliers.

Hypothèses

- Création de deux hypothèses de feature engineering :
  - *"Polynomial Features"* : Amélioration de la correlation des variables EXT SOURCES avec la target
  - *"Domain Features"* : Construction de variables s'appliquant plus au domaine de la banque comme :
    - "CREDIT_INCOME_PERCENT"
    - "ANNUITY_INCOME_PERCENT"
    - "CREDIT_TERM"
    - "DAYS_EMPLOYED_PERCENT"

III – MODELISATION

PREPROCESSING



ENTRAINEMENT ET OPTIMISATION

3 hypothèses de features engineering

1 hypothèse de features engineering

1 hypothèse de features engineering

Equilibrage des données

- Utilisation de la librairie imblearn
- Under-Sampling
- 24825 individus dans chaque classe

Choix de la meilleure hypothèse

- Utilisation d'une régression logistique comme baseline
- Hypothèse retenue : *Domain Features*

Essais de modélisation

- Mise en place de trois classifieurs :
  - Regression logistic
  - Random Forest
  - XGBoost
- Essai de modélisation sans validation croisée

Optimisation du modèle le plus prometteur

- Modèle retenu : XGBoost
- Optimisation par RandomizedSearchCV
- 6 hyperparamètres testés *(Voir note technique)*

Le modèle est entrainé sur un jeu de données que l'on a équilibré mais le jeu de test n'a pas été équilibré à son tour.

Ceci dans le but de ne pas fausser les résultats.

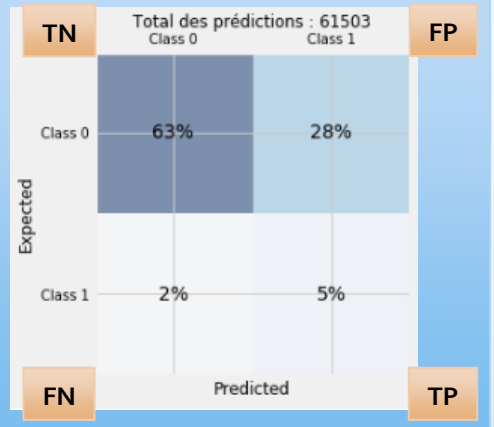
ANALYSE RESULTATS

Métriques pour un modèle de classification :

1. **Accuracy** : La précision du modèle
2. **Precision** : Performance du modèle quand celui-ci déclare une classe 1.
3. **Recall** : Pourcentage de détection des classes 1.
4. **F1_score** : Moyenne *harmonique* de la précision et du rappel.

La matrice de confusion :

La matrice de confusion consiste à compter le nombre de fois où des observations de la classe 0 ont été rangées dans la classe 1. Par exemple, si nous voulons connaître le nombre de fois où le classifieur à bien réussi à classer une classe 1, on examinera la cellule à l'intersection de la ligne 1 et de la colonne 1.

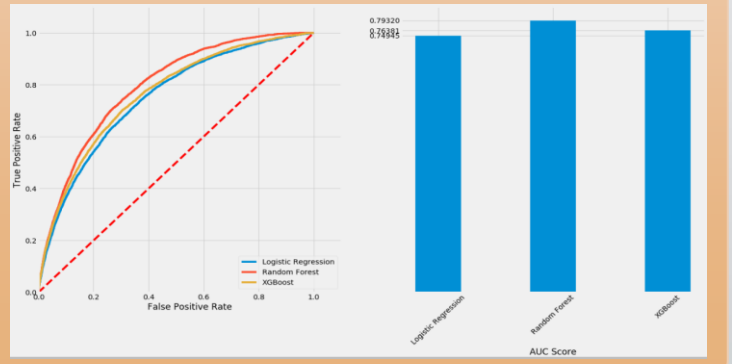


La courbe ROC et le score AUC :

La courbe ROC (Receiver Operating Characteristic) est un outil communément utilisé avec les classifieurs binaires. Elle croise le taux de TP avec le taux de FP.

Un bon classifieur aura sa courbe qui s'approche le plus possible du coin supérieur gauche du graphique.

Une autre façon de comparer des classifieurs consiste à mesurer l'aire sous la courbe (Area Under the Curve ou AUC). Un classifieur parfait aurait un score AUC égal à 1, tandis qu'un classifieur purement aléatoire aurait un score AUC de 0.5.



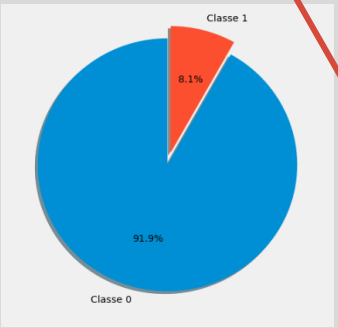
Explication des targets / Déséquilibre de la population:

Nous avons à faire à un problème de classification binaire où la population est fortement déséquilibrée.

Explication des targets :

- **Target = 0 :** Client ne représentant pas de risque de faillite
- **Target = 1 :** Client représentant un risque de faillite pour l'entreprise

Il y a 92% de clients ne représentant pas de risque de faillite dans notre population.



Analyse de notre Baseline sur une population déséquilibrée :

On constate que le modèle ne prédit que des 0. Son accuracy est très bonne mais nous cherchons à prédire si une Target sera égale à 1.

	Accuracy	Precision	Recall	F1_score
sans_feat_eng	0.92	0	0	0
poly_feat	0.92	0	0	0
domain_feat	0.92	0	0	0

Nous focaliserons donc notre performance de modèle sur la précision et le rappel.

ANALYSE RESULTATS

Analyse des métriques

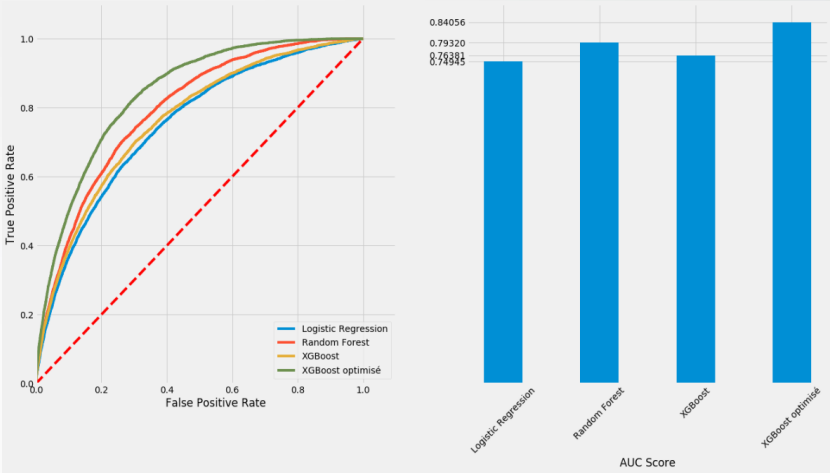
	Accuracy	Precision	Recall	F1_score
Logistic regression	0.69	0.16	0.68	0.26
Random Forest	0.68	0.17	0.76	0.27
XGBoost	0.69	0.16	0.71	0.27
XGBoost optimisé	0.71	0.19	0.84	0.31

$$\text{Rappel} = \frac{VP}{VP + FN}$$

$$\text{Précision} = \frac{VP}{VP + FP}$$

On constate que le modèle arrive à détecter 84% des classes 1, mais qu'il n'a raison que dans 19% des cas quand il en détecte 1.

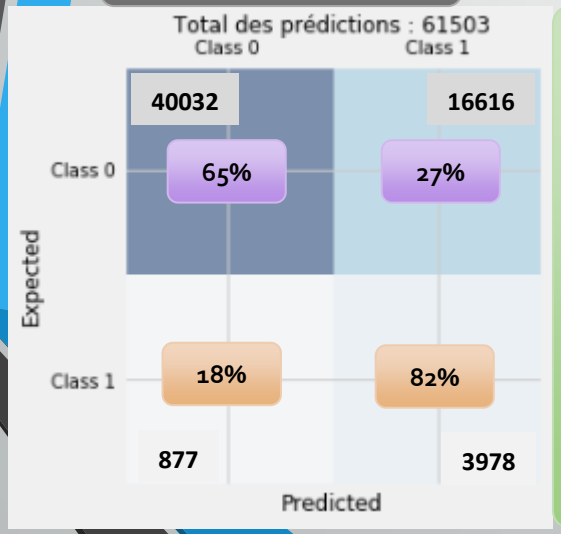
Courbe ROC et score AUC



On remarque bien que la courbe ROC du modèle XGBoost optimisé reste quand même assez loin du coin supérieur gauche du graphique. Cela donne une représentation visuelle de la performance globale du modèle.
On constate aussi que l'optimisation nous a permis d'améliorer le modèle.

Analyse de la matrice de confusion

92% de targets 0
8% de targets 1



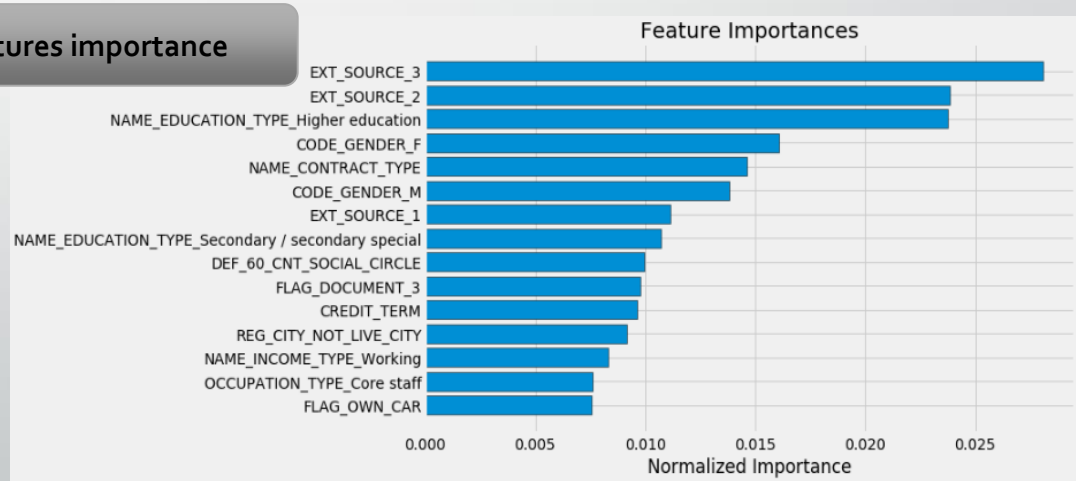
Il y 61503 individus dans le jeu de test dont 56648 classés 0 et 4855 classés 1.

82% d'individus de classe 1 trouvés.

30% d'individus de classe 0 sont détectés en classe 1.

*Le modèle alerte **trop** souvent sur le risque de faillite d'un client.*

Features importance



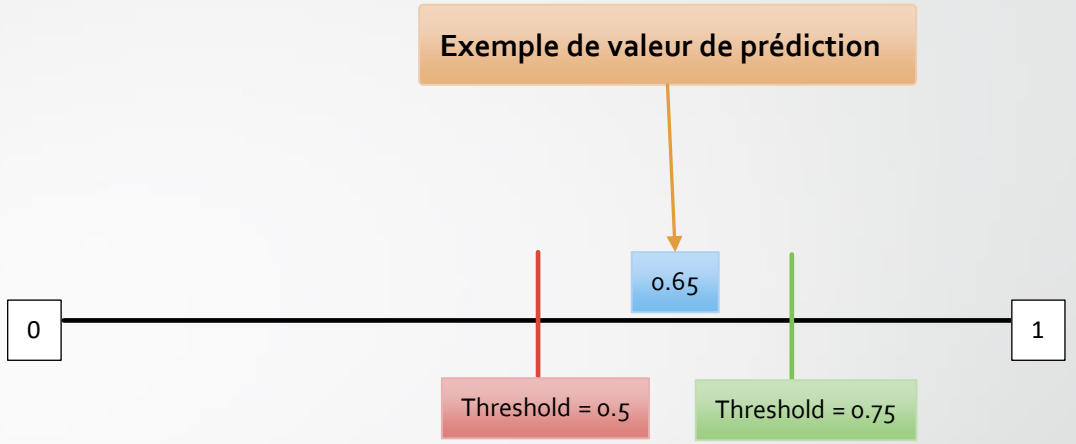
On constate que ce sont les ressources extérieures qui ont le plus d'importance pour les prédictions, bien que leur pourcentage n'est pas élevé : ≈2%

ANALYSE RESULTATS

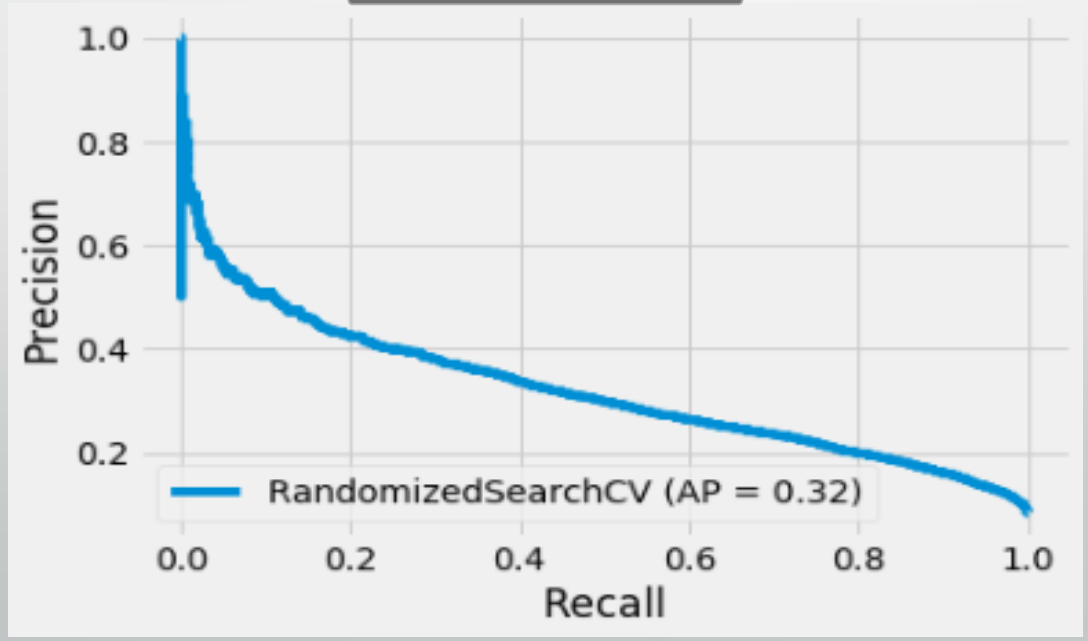
Analyse de la courbe Précision/Rappel

La courbe de Précision/Rappel nous permet de visualiser le meilleur compromis que l'on puisse avoir avec notre modèle. Le compromis Précision/Rappel se définit grâce au *threshold* (seuil de décision).

Explication du treshold :
Le seuil de décision est une valeur que nous fixons et qui va limiter qu'une valeur appartient à la classe 0 ou à la classe 1.



Courbe Précision/Rappel



On constate que le meilleur compromis que l'on pourrait trouver en réglant la valeur du seuil est :
Precision : ≈0.35
Recall : ≈0.4

IV – DASHBOARD

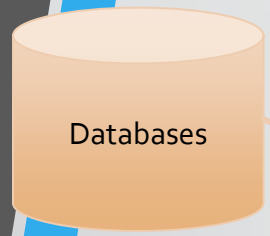
CONSTRUCTION EN LOCAL

Flask

Flask *est un framework web qui permet de réaliser des sites dynamiques, mais nécessite une extension de type Dash pour coder le Frontend.*

Streamlit

Streamlit *est une librairie python qui permet de coder la partie Frontend par l'intermédiaire de WebSocket, mais à l'avantage d'intégrer Tornado, un équivalent à Flask, pour servir les données HTTP.*
Autrement dit, c'est une solution tout en un, qui a comme autre avantage de fonctionner en Python pur.



1ère étape : Démarrer le serveur

API avec Flask

URL LOCALE :
http://localhost:5000/

Format de transfert : JSON

DASHBOARD avec Streamlit

URL LOCALE :
http://localhost:8501

2ème étape : Afficher le dashboard

Fichier API.py

Partie « Back-End » du DASHBOARD.

C'est dans ce fichier que sont effectuées toutes les opérations non graphiques (chargement des données, entrainement des modèles, prédictions, …)

Contient tous les end points pour interagir avec d'autres logiciels.

Exemple de requête envoyée à l'API

```
# Requête permettant de récupérer les informations du client sélectionné
infos_client = requests.get(URL_API + "infos_client", params={"id_client":id_client})
```

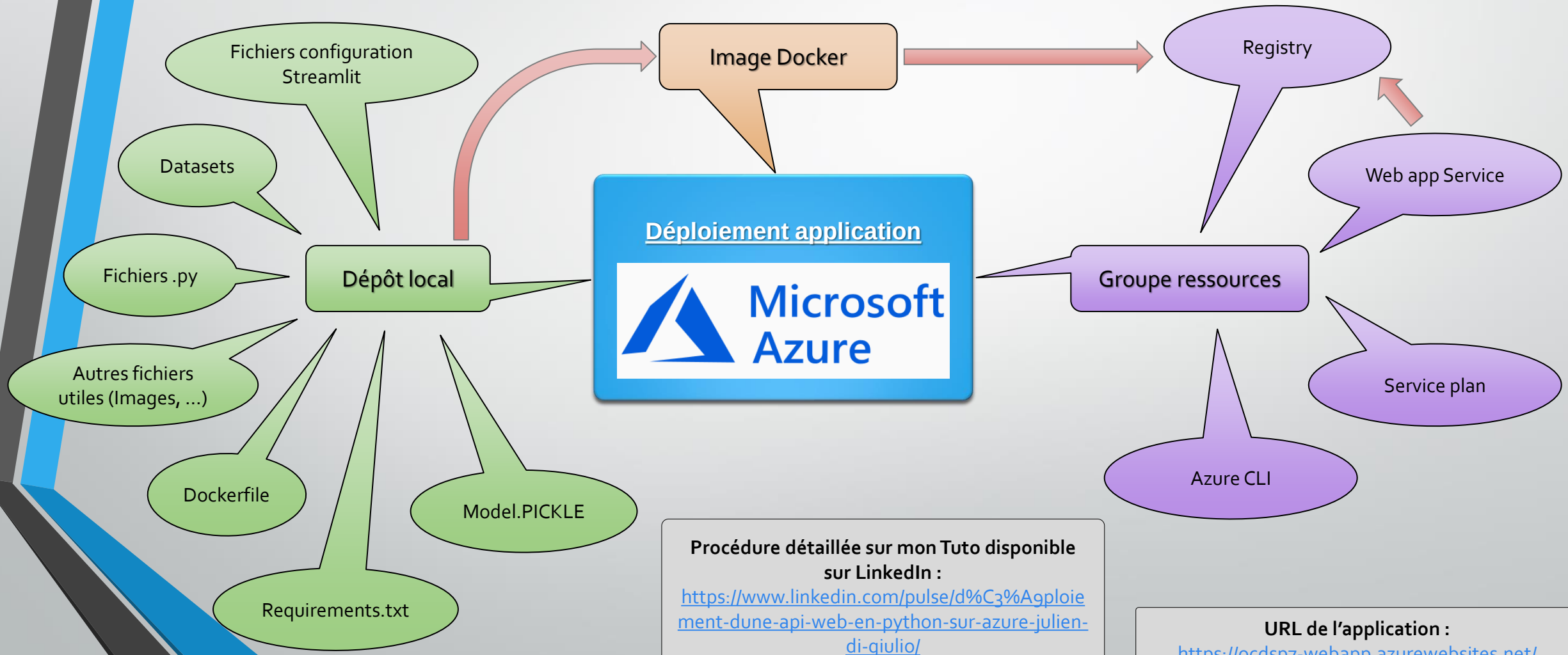
URL résultante reçue par l'API :
http://localhost:5000/infos_client?id_client=100101

Fichier DASHBOARD.py

Partie « Frontend » du DASHBOARD.

C'est dans ce fichier qu'est codée la partie graphique de la page web qu'utilisera la chargé de clientèle.

DEPLOIEMENT SUR LE CLOUD



Procédure détaillée sur mon Tuto disponible sur LinkedIn :
https://www.linkedin.com/pulse/d%C3%A9ploiement-dune-api-web-en-python-sur-azure-julien-di-giulio/

URL de l'application :
https://ocdsp7-webapp.azurewebsites.net/

V – CONCLUSION

RESUME

CLASSIFICATION

- Construction d'un modèle de classification binaire à partir d'un Kernel de départ téléchargé sur Kaggle.
- Une population fortement asymétrique (92% - 8%)
- Les résultats de prédiction ne sont pas satisfaisant, avec comme scores 0.4 de recall pour 0.35 de précision.

API / DASHBOARD

- Création d'une API web avec Flask pour le côté serveur, et Streamlit pour le côté dashboard.
- Construction d'une image Docker déployée dans le Registry Azure.
- Configuration du webapp service pour mettre en ligne l'application.

AXES D'AMELIORATION

Classification :

- Le feature engineering ne tient compte que d'un seul dataset sur les huit disponibles. Un Kernel de départ plus approprié à notre problème permettrait surement d'améliorer les scores. Idéalement, une étude personnalisée des données.
- La méthode SMOTE pour l'équilibrage des données est plus performante que celle utilisée dans ce projet, mais beaucoup plus longue en traitement. Lien utile avec les différentes stratégies de resampling.
- Une recherche de performances de prédiction plus approfondie, avec réseaux de neurones par exemple.
- Une optimisation plus fine en étudiant plus en détails chaque hyperparamètre.

Déploiement :

- Azure n'est pas le fournisseur de Cloud le plus simple pour déployer une application Web Python. Heroku et AWS semblent plus appropriées.

PROFIL GitHub

L'ensemble des fichiers de ce projet ont été stockés sur mon compte GitHub : https://github.com/JulienDiGiulio/OpenClassRooms_Projet7.git

Lignes de commandes dans le dépôt local sur ma machine :

Git init
Git add .
Git commit « Name commit »
Git remote « Link Github repo »
Git push –u origin master

Fichier .gitignore.txt pour ignorer certains fichiers csv

QUESTIONS - REPONSES