Implémentez un modèle de scoring

Élaborer un modèle prédisant la probabilité de défaut de paiement de clients de “Prêt à Dépenser”
Déployer le modèle via une API

Date de la soutenance : 23/05/2024

Antoine Arragon

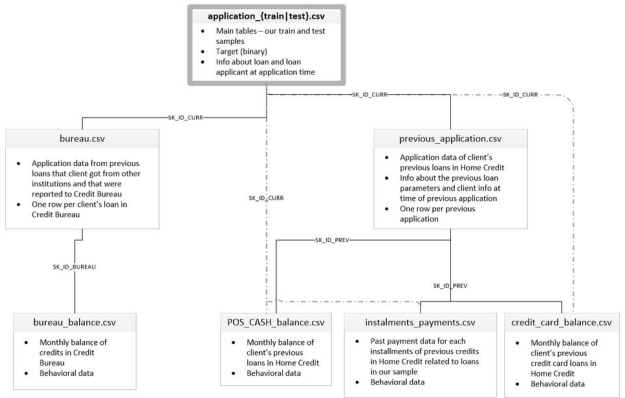
Objectifs

L'entreprise Prêt à Dépenser souhaite mettre en oeuvre un outil de "**scoring crédit**" permettant de calculer la **probabilité qu'un client rembourse son crédit** sans difficulté. Cette probabilité permettant ensuite de **classer la demande** en "crédit accepté" ou "crédit refusé".

- Analyse exploratoire et création de données pertinentes ;
- Démarche de modélisation dans un souci de minimisation des coûts engendrés par de mauvaises prédictions ;
- Analyse des variables les plus importantes pour le modèle de prédiction ;
- Création d'une API permettant de requêter par identifiant client et de récupérer la probabilité de défaut et le statut de la demande.

Présentation rapide des données

- Dataset constitué de plusieurs tables :



- Deux grands types d'informations :
  - Administratives : données relatives à l'âge, au sexe, au métier... des clients ;
  - Bancaires :
    - Informations provenant de l'historique interne de HomeCredit ;
    - Informations provenant d'autres institutions financières
- Regroupement des données par client, via différentes agrégations de variables;
- Dimensions du jeu de données final : 307505 lignes (clients) et 287 colonnes (variables).

Plan :

1. Démarche de modélisation
    - 1.1. Spécificités du problème
    - 1.2. Méthodologie
    - 1.3. Recherche du meilleur modèle et des meilleurs hyperparamètres
    - 1.4. Evaluation et sélection du meilleur modèle
    - 1.5. Analyse de la feature importance globale et locale

2. Déploiement du modèle retenu via une API
    - 2.1. Démarche de création et utilisation de l'API
    - 2.2. Tests unitaires
    - 2.3. Analyse de Datadrift

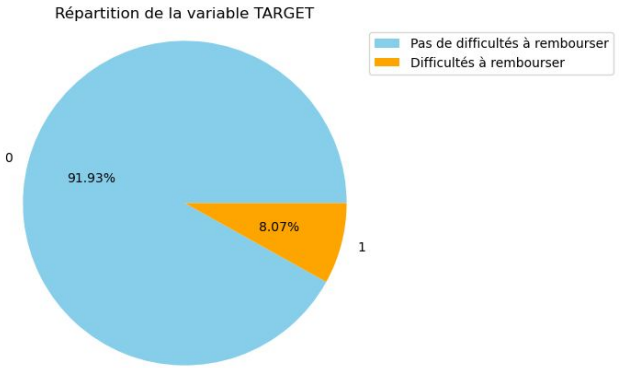
3. Requêtes API

Limites et conclusion

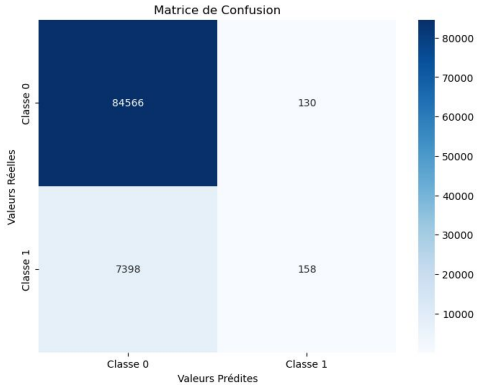
1. Démarche de modélisation

1.1. Spécificités du problème

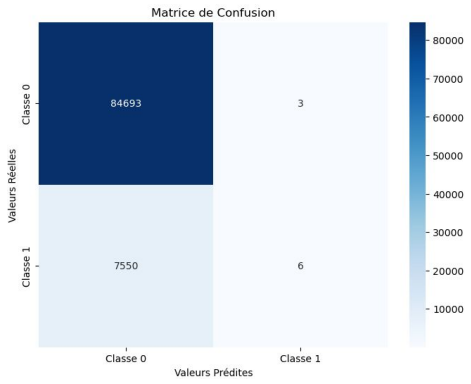
Déséquilibre de la distribution de la variable cible :



Tendance des modèles à prédire la classe “dominante” :



Régression logistique



Random Forest Classifier

1. Démarche de modélisation

1.1. Spécificités du problème

Implications de ce déséquilibre de la distribution de la variable cible :

- Optimisation du **seuil de prédiction** ;
- Sélection de métriques d'évaluation plus appropriées que l'accuracy :
  - Utilisation du **recall** et du **roc_auc_score** notamment ;
- Utilisation du paramètre '**class_weight**' de certains modèles :
  - Etabli à 'balanced' -> pénalise + fortement les erreurs sur la classe minoritaire ;
- Utilisation de méthode de '**sampling**' :
  - SMOTE : méthode d'oversampling ;
  - RandomUnderSampler : méthode d'undersampling ;
  - Distribution 'post sampling' :

```
# distribution target après le sampling :
y_train_samp.value_counts(normalize=True)
✓ 0.0s

TARGET
0    0.666667
1    0.333333
Name: proportion, dtype: float64
```

1. Démarche de modélisation

1.1. Spécificités du problème

Implications métiers différentes dans les erreurs de prédictions :

Hypothèse : faux négatif 10 fois + coûteux qu'un faux positif.

- Création d'une **fonction de coût "métier"** ;
- Optimisation des modèles via cette fonction ;
- Optimisation du **seuil de prédiction** ;
- **Calibration** si nécessaire.

Fonctions de coût/score métier :

```python
def custom_cost(y_true, y_pred):
    # On récupère chaque élément de la matrice de confusion :
    tn, fp, fn, tp = confusion_matrix(y_true, y_pred).flatten()
    # Pénalisation des faux négatifs 10 fois plus que les faux positifs
    cost = 10 * fn + fp
    return cost
```

```python
def std_custom_score(y_true, y_pred):
    # Coût total maximal possible
    max_cost = 10 * sum(y_true == 1) + sum(y_true == 0)

    # Coût standardisé entre 0 et 1
    std_cost = 1 - custom_cost(y_true, y_pred) / max_cost
    return std_cost
```

1. Démarche de modélisation

1.2. Méthodologie

Point sur l'EDA :

1) **application_train :**
   - Gestion des **valeurs incohérentes** : suppression ou modification;
   - **Suppression** des colonnes avec **+ 50% de valeurs manquantes** (exception 'Ext source scores) ;
   - Création de quelques variables (INCOME_PER_PERSON, CREDIT_INCOME_RATIO) ;
   - **Imputation** des **variables numériques** par **médiane** ;
   - **Imputation** des **variables catégorielles** par **mode** ;
   - **Encodage des variables catégorielles** par LabelEncoder si modalités < 3, One Hot Encoder sinon.

2) Travail sur les autres jeux de données afin de réaliser des agrégations et d'aboutir à une ligne par client :

- Mêmes imputations que sur application_train
- Même encodage des variables catégorielles

3) **Merge des différents dataframes ;**

4) **Imputation par 0 des valeurs manquantes ;**

5) **Application des mêmes étapes au jeu de données application_test ;**

1. Démarche de modélisation

1.2. Méthodologie

- **Type de problème :** Classification binaire supervisée
- **Jeu de données :** trainset réalisé lors de l'EDA qu'on split : 30% test - 70% train
- **Baseline :** DummyClassifier
- Recherche des meilleurs hyperparamètres via **hyperopt**
- **Modèles comparés :**
  - Logistic Regression
  - SVC
  - RandomForestClassifier
  - XGBClassifier
  - LightGBMClassifier
- **Pipeline :**
  - MinMaxScaler ;
  - Smote(0.1) et RandomUndersampler(0.5)

- Métriques techniques :
  - Accuracy : fréquence des prévisions correctes ;
  - Recall (sensitivity) : capacité du modèle à prédire les vrais positifs ;
  - Roc_auc_score : capacité du modèle à bien distinguer les classes à prédire
  - Visualisation par matrice de confusion : met bien en évidence les TP, FP, TN, FN.
- Score Métier :
  - Fonction de coûts :
  - Score standardisé :
  - Visualisation par "matrice de confusion des coûts".

1. Démarche de modélisation

1.3. Recherche du meilleur modèle et des meilleurs hyperparamètres

Utilisation d'hyperopt :

- Définition d'un espace de recherche: modèles et hyperparamètres à tester ;
- Définition d'une fonction objectif :
  - Pipeline (Scaling, sampling, modèle)
  - Cross-validation avec score métier à optimiser
  - Fonction de coût à minimiser (fonctionnement d'hyperopt)
- Appel de la fonction fmin qui lance la recherche en prenant en compte les éléments précédents ;
- Tracking de l'ensemble via MlFlow

```
# Defining Search Space
space = hp.choice('classifiers', [
    {
    'model':LogisticRegression(random_state=random_state, class_weight='balanced', max_iter=250, solver='saga'),
    'params':{
        'model__penalty': hp.choice('lr.penalty', [None, 'l1', 'l2']),
        'model__C': hp.choice('lr.C', np.arange(0.0001,1.0,0.01)),
        'model__tol': hp.choice('lr.tol', np.arange(0.0001,1.0,0.01))
    }
    },

    {
    'model':SVC(random_state=random_state, class_weight='balanced', probability=True, max_iter=250),
    'params':{
        'model__C': hp.choice('svc.C', np.arange(0.0001,1.0,0.01)),
        'model__gamma': hp.choice('svc.gamma', ['scale', 'auto']),
        'model__tol': hp.choice('svc.tol', np.arange(0.0001,1.0,0.01))
    }
    },
```

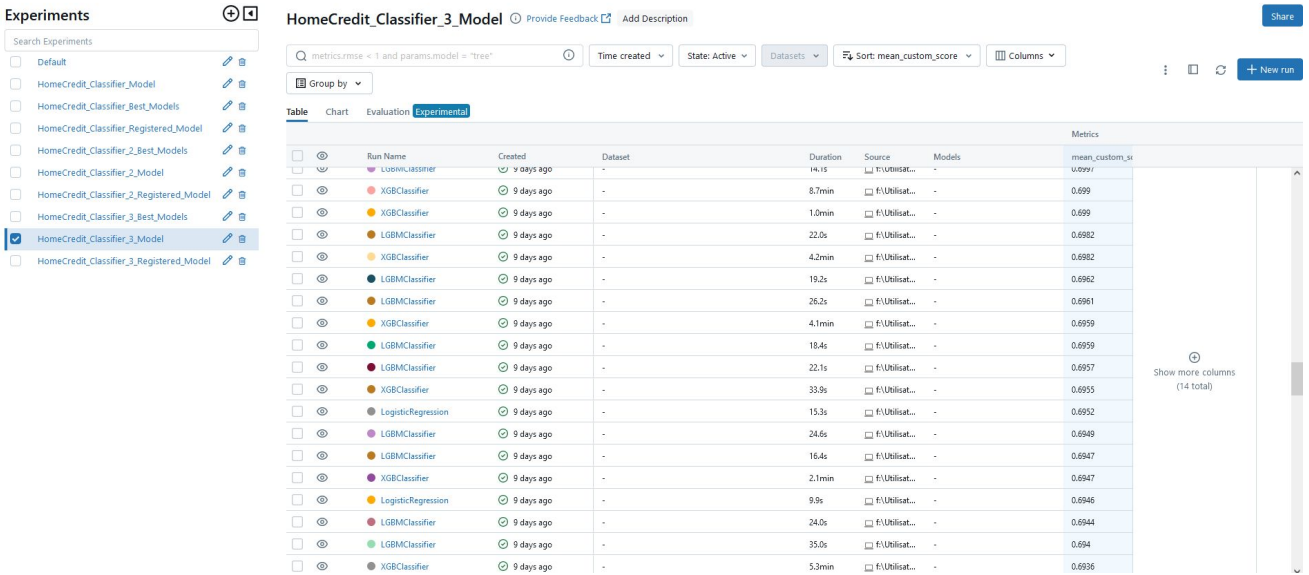
```
# Defining Objective function whose loss we have to minimize
def objective(args):

    # Extracting model name
    model_name = args['model'].__class__.__name__

    # Initialize model pipeline
    pipe = imbpipeline(steps=[
        ('scaler', MinMaxScaler()),
        ('smote', SMOTE(sampling_strategy=0.1, random_state = 42)),
        ('undersampling', RandomUnderSampler(sampling_strategy=0.5, random_state=42)),
        ('model', args['model']) # args[model] will be sent by fmin from search space
    ])
```

```
kfold = StratifiedKFold(n_splits=5, shuffle=True, random_state=random_state)
scores = cross_val_score(pipe, X_train, y_train, cv=kfold, n_jobs=2, scoring=std_custom_cost_score)
```

1. Démarche de modélisation

1.3. Recherche du meilleur modèle et des meilleurs hyperparamètres

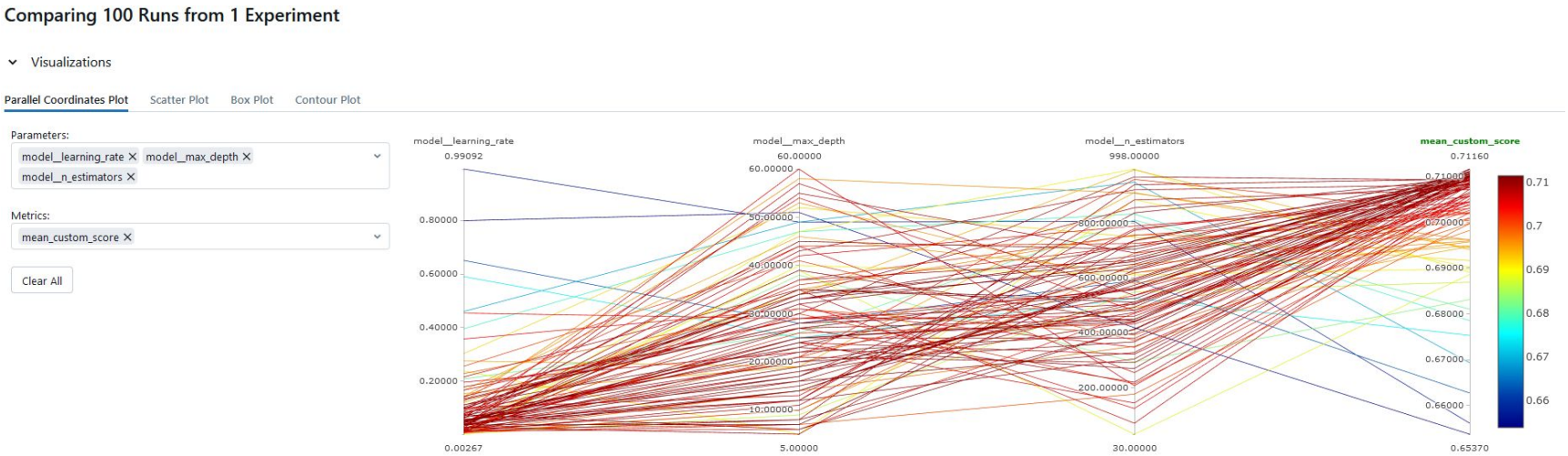
Tracking via MlFlow :



1. Démarche de modélisation

1.3. Recherche du meilleur modèle et des meilleurs hyperparamètres

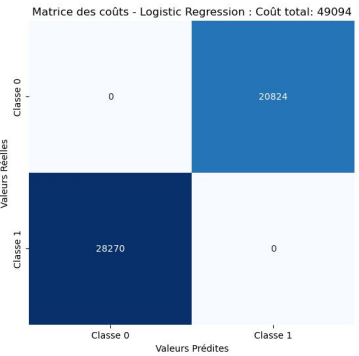
Tracking via MlFlow :

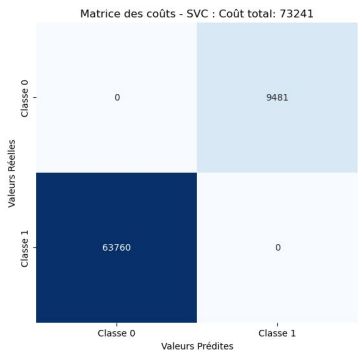


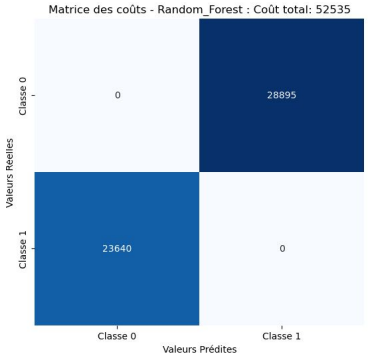
1. Démarche de modélisation

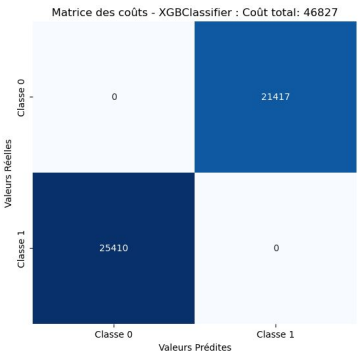
1.4. Evaluation et sélection du meilleur modèle

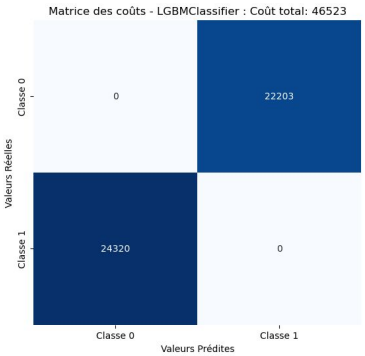
modèle	hyperparamètres	eval_acc	eval_prec	eval_recall	eval_f1_score	eval_custom_fbeta	eval_roc_auc	eval_custom_cost	eval_custom_score	training_time
LGBMClassifier	{'colsample_bytree': 0.9803840371703639, 'lear...	0.732960	0.187507	0.678137	0.293782	0.511230	0.707994	46523	0.709696	3.7104
XGBClassifier	{'colsample_bytree': 0.5617809435435263, 'gamm...	0.740298	0.189732	0.663711	0.295104	0.504753	0.705421	46827	0.707799	4.2559
Logistic Regression	{'C': 0.2601, 'penalty': 'l1', 'tol': 0.0201}	0.743626	0.185066	0.625860	0.285663	0.479145	0.689996	49094	0.693653	3.3480
Random Forest	{'max_depth': 30, 'max_samples': 0.60247682123...	0.661156	0.152316	0.687136	0.249358	0.482239	0.672987	52535	0.672181	20.4866
SVC	{'C': 0.8001, 'gamma': 'auto', 'tol': 0.9901}	0.828112	0.110684	0.156167	0.129549	0.137070	0.522113	73241	0.542975	42.9010





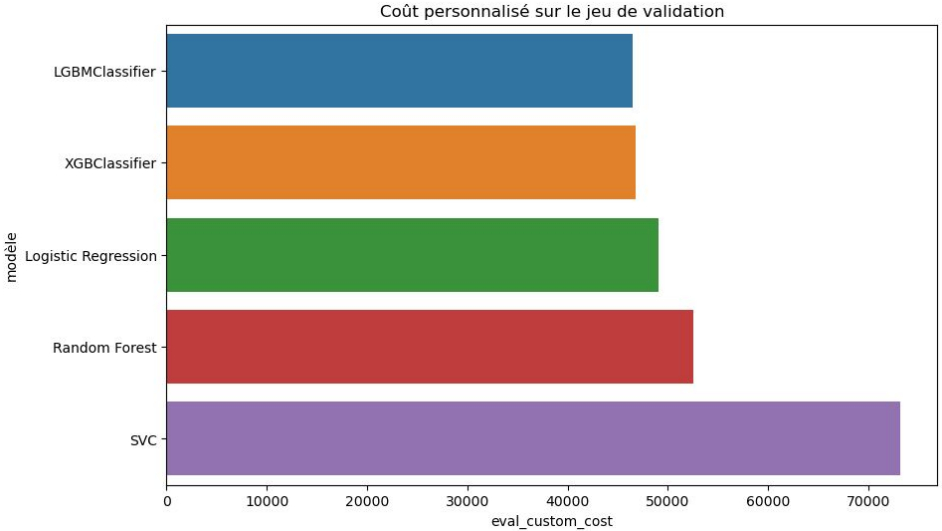


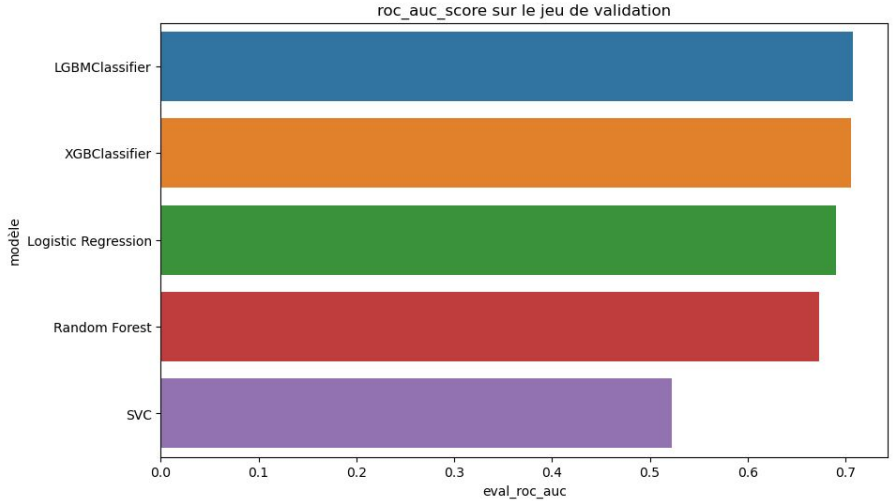




1. Démarche de modélisation

1.4. Evaluation et sélection du meilleur modèle

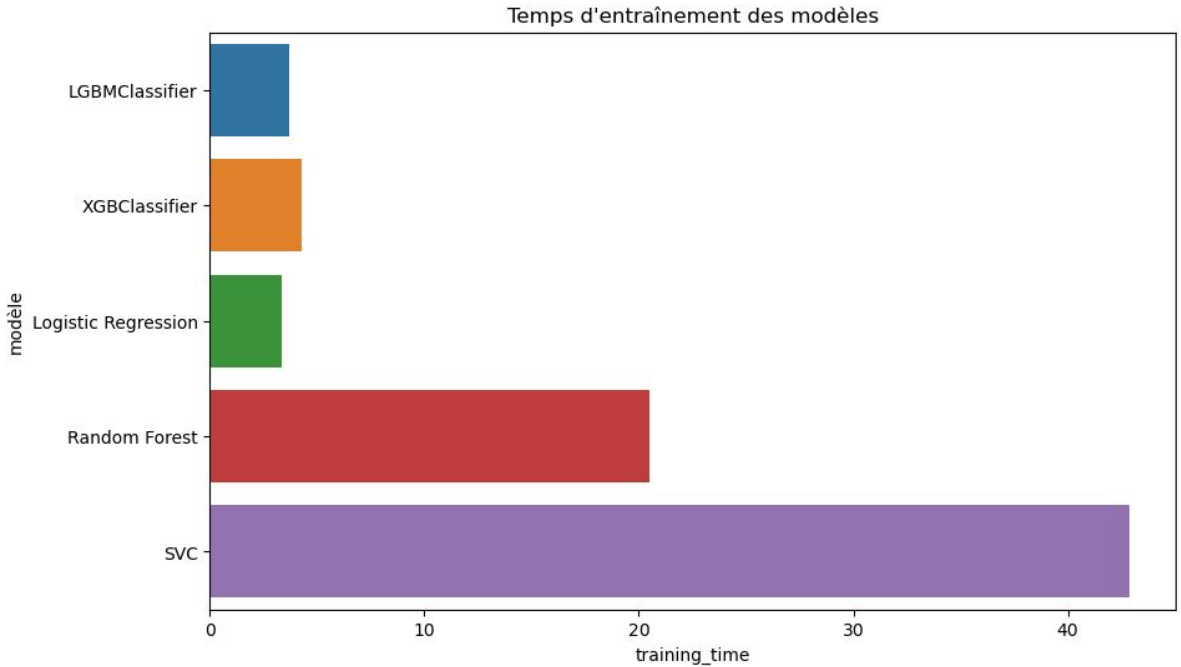




Intégration d'une fonction optimisant le seuil de probabilité de défaut en lien avec notre score métier.

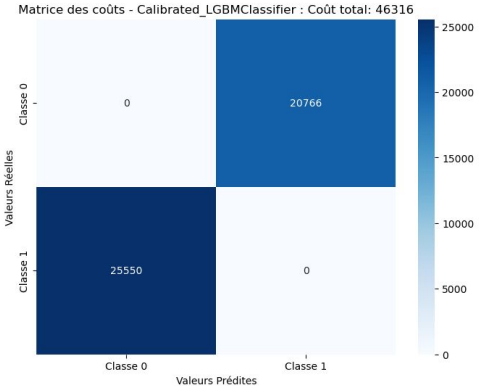
1. Démarche de modélisation

1.4. Evaluation et sélection du meilleur modèle



Modèle retenu : LGBMClassifier

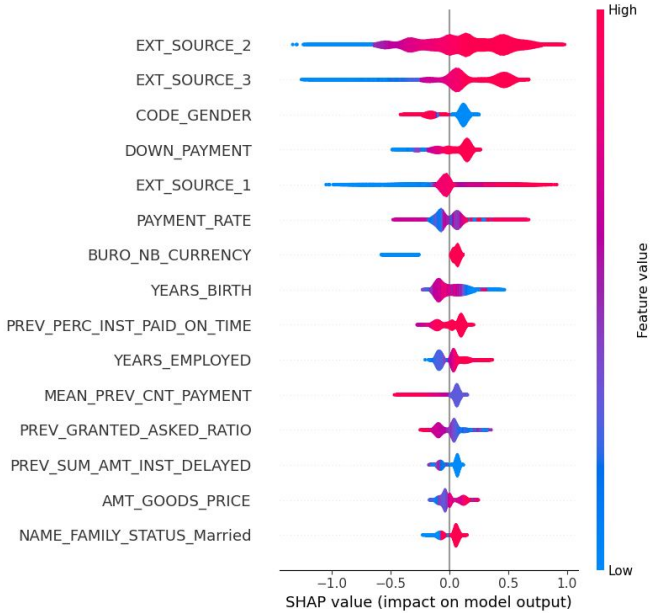
➢Application d'une calibration des probabilités prédites via CalibratedClassifierCV



1. Démarche de modélisation

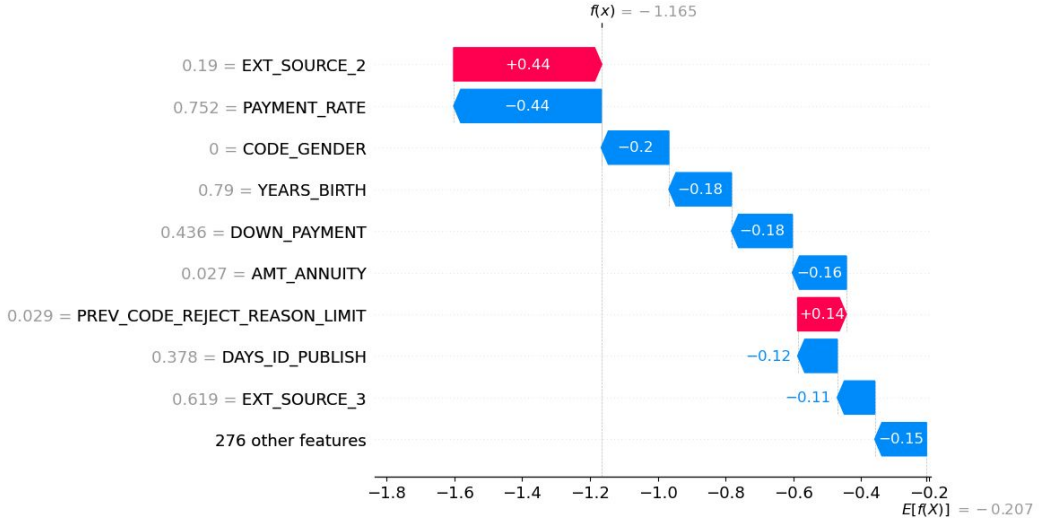
1.5. Analyse de la feature importance globale et locale

Top 15 des features - prédiction classe 0 :



Global

Principales features expliquant le remboursement d'un individu en particulier :



Local

2. Déploiement du modèle retenu via une API

2.1 Démarche de création et utilisation de l'API

- Utilisation d'une API Flask
- Objectifs :
  - Récupérer le pipeline et le meilleur modèle des étapes précédentes ;
  - Utiliser un échantillon du jeu de données d'entraînement ;
  - Configurer plusieurs routes de manières à faire les requêtes suivantes :
    - Accéder à une liste des identifiants clients ;
    - Faire appel au modèle et obtenir la probabilité de défaut et le statut d'un client sélectionné (obtenir via la même requête des informations sur le profil du client) ;
    - Avoir une vue sur la feature importance globale du modèle utilisé ;
    - Avoir une vue sur la feature importance locale du client sélectionné.
- Déployer l'API sur le cloud via Github Actions et Heroku.

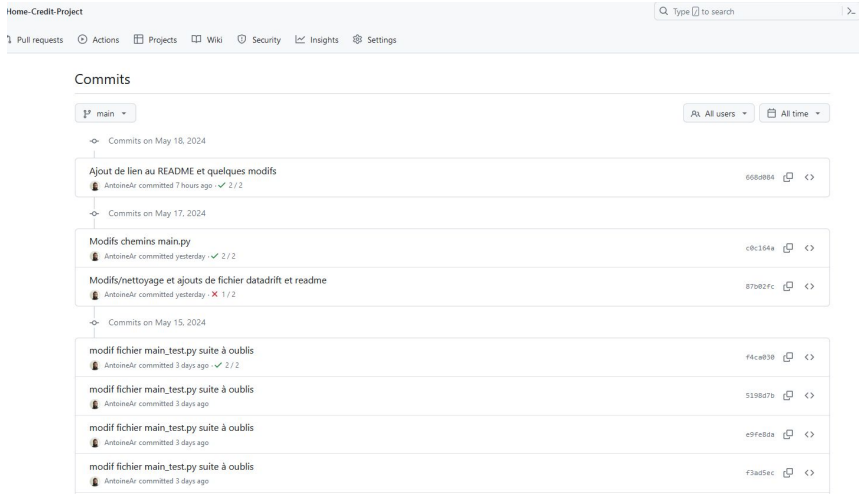
2. Déploiement du modèle retenu via une API

2.1 Démarche de création et utilisation de l'API

Lien vers le repository github : https://github.com/AntoineAr/Home-Credit-Project

```python
@app.route("/")
def welcome():
    return ("Bienvenue sur l'API de prédiction de défaut de paiement !\n\n")

# Route qui affiche la liste des IDs clients valides :

@app.route('/list_ids')
def print_id_list():
    return f'Liste des id clients valides :\n\n{(clients_ids)}'

# Route qui affiche les informations d'un client et sa probabilité de défaut de paiement :
@app.route('/prediction/<int:client_id>')
def prediction(client_id):
    if client_id in clients_ids:
        client_data = features.loc[client_id].values.reshape(1, -1)
        proba = model.predict_proba(client_data)[0, 1]

        client_infos = get_client_infos(client_id, "./data/subset_train_brut.csv")

        customer_pred = {
            'id': client_id,
            'probabilité_défaut': proba.round(2),
            'statut': 'non risqué' if proba <= threshold else 'à risque',
            'client_infos' : client_infos
        }

        return jsonify(customer_pred)
    else:
        return 'Client_id non valide.'
```

Extrait du script de l'API

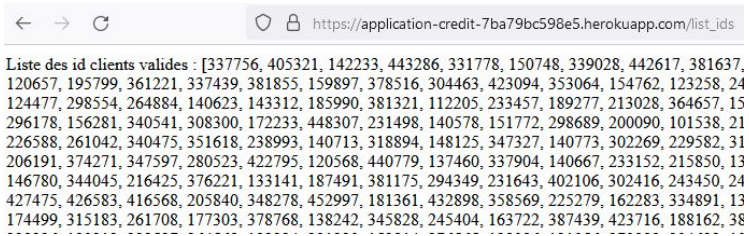


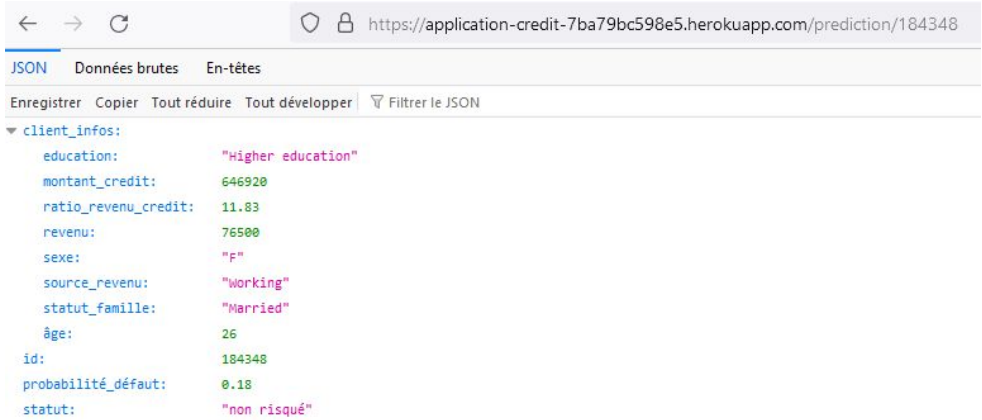
Différents commit effectués sur github.

2. Déploiement du modèle retenu via une API

2.1 Démarche de création et utilisation de l'API

Lien vers l'API déployée sur heroku : https://application-credit-7ba79bc598e5.herokuapp.com/



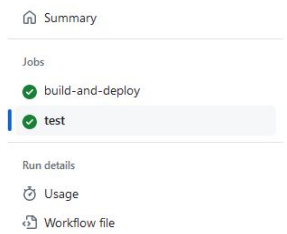


2. Déploiement du modèle retenu via une API

2.2. Tests unitaires

Utilisation de pytest intégré au workflow de déploiement via Github Actions :

```python
# Fonction qui teste quelques ID clients pour vérifier qu'ils sont bien valides :
client_id_test = [146124, 242167, 343897]
def test_ids_client():
    df = load_data("./data/subset_train.csv")
    scaler, model, explainer = load_scaler_model_explainer()
    features = prepare_data(df, scaler)
    clients_ids = get_clients_ids(features)
    for client_id in client_id_test:
        assert client_id in clients_ids

# Fonction qui teste le bon fonctionnement d'une prédiction pour un client à risque :
def test_prediction_client_risk():
    # URL de l'app :
    url = 'https://application-credit-7ba79bc598e5.herokuapp.com/prediction/'
    # ID client connu pour être à risque
    client_id_risk = 343897
    url_client = url + str(client_id_risk)
    # Envoi d'une requête GET avec l'ID client
    response = requests.get(url_client)
    # Vérifier que la requête a réussi (code de statut HTTP 200)
    assert response.status_code == 200
    # Analyser la réponse JSON
    response_json = response.json()
    # Vérifier que la réponse indique que le client est à risque
    assert response_json['statut'] == 'à risque'
```



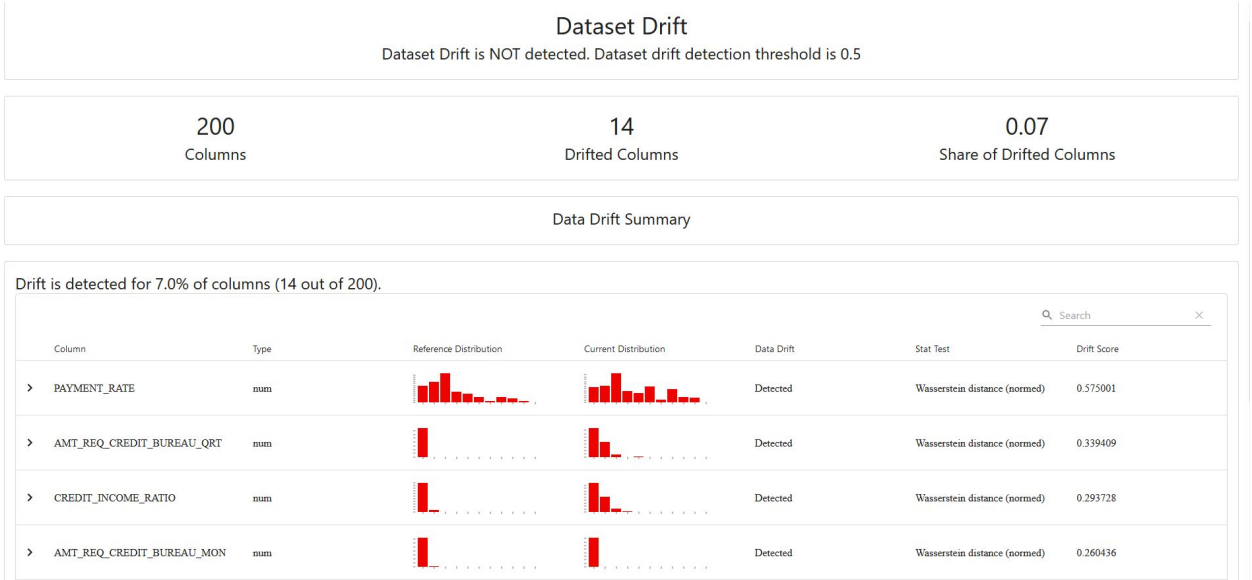
Extrait du script de test

2. Déploiement du modèle retenu via une API

2.3. Analyse de Datadrift

Utilisation de la librairie Evidently

1e utilisation pour une détection de drift entre les features présentes dans les jeux de données application_train et application_test :

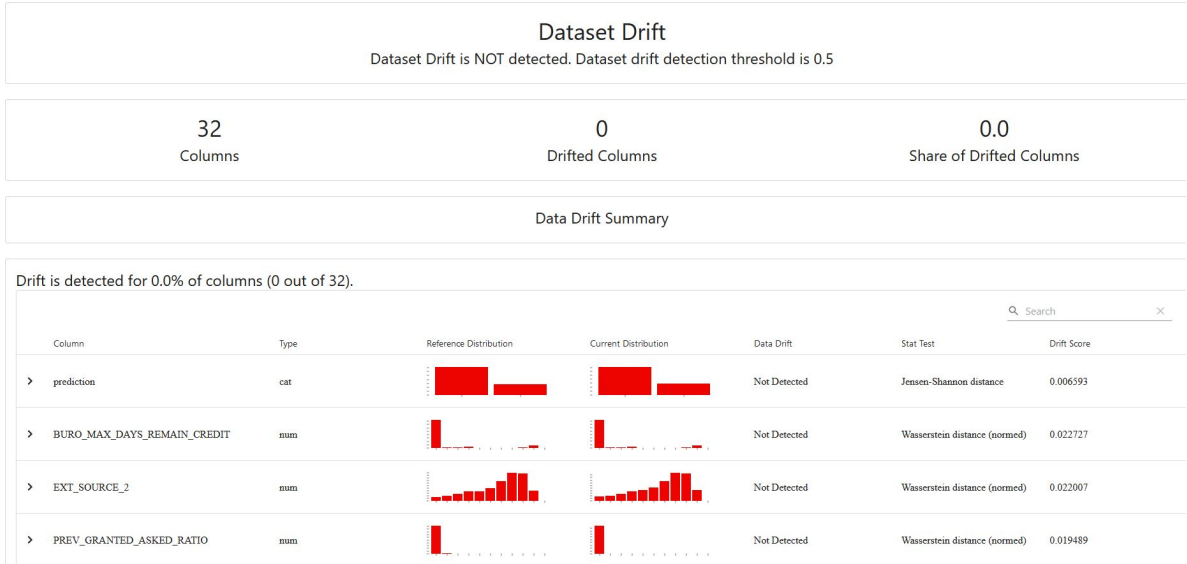


2. Déploiement du modèle retenu via une API

2.3. Analyse de Datadrift

Utilisation de la librairie Evidently

- Réalisation d'une autre analyse en utilisant un échantillon du jeu de données créé pour la modélisation ;
- Sélection des 30 features les plus importantes selon SHAP ;
- Inclusion de colonnes 'prediction' et 'target' dans la détection du drift :



3. Requêtes API

Présentation de l’API sur le cloud

Limites et conclusion

Projet qui a permis d'établir une proof of concept mais qui reste à affiner et à améliorer sur plusieurs points :

- L'EDA qui pourrait être plus poussée (réflexion sur d'autres variables à créer, sélection de features ?) ;
- Effectuer plus de tests pour trouver la méthode qui gère le mieux le déséquilibre des classes ;
- Essayer d'améliorer les performances du modèles ;
- Faire appel à un expert métier pour une définition plus juste des coûts à prendre en compte ;
- L'utilisation de SHAP a révélé un biais discriminant sur le genre, variable à conserver ? Modèle à retravailler de manière à ce que cette variable ne soit pas aussi discriminante ?
- Ajouter différentes procédures de tests et levées d'exceptions dans la création et le déploiement de l'API.

Merci de votre attention.

Bibliothèques principales utilisées :

Bibliothèques générales :

- matplotlib 3.7.2
- numpy 1.24.3
- pandas 2.0.3
- seaborn 0.12.2

ML :

- scikit-learn 1.3.0
- shap 0.44.1
- hyperopt 0.2.7
- libxgboost 2.0.3
- lightgbm 4.3.0

API :

- flask 3.0.3
- requests 2.31.0
- pytest 8.2.0

Datafrift :

- evidently 0.4.22