# Note Méthodologique :

* La méthodologie d'entraînement du modèle (2 pages maximum)
* La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation (1 page maximum)
* L’interprétabilité globale et locale du modèle (1 page maximum)
* Les limites et les améliorations possibles (1 page maximum)

## Présentation du projet :

L’entreprise « **Prêt à dépenser »** souhaite **mettre en œuvre un outil de “scoring crédit” pour calculer la probabilité** qu’un client rembourse son crédit, puis classifie la demande en crédit accordé ou refusé. Elle souhaite donc développer un **algorithme de classification** en s’appuyant sur des sources de données variées (données comportementales, données provenant d'autres institutions financières, etc.)

Un Dashboard interactif permettra d’afficher les résultats pour les chargés de relation client.

Dans cette note méthodologique, nous trouverons :

* La méthodologie d'entraînement du modèle
* La fonction coût métier, l'algorithme d'optimisation et la métrique d'évaluation
* L’interopérabilité globale et locale du modèle
* Les limites et les améliorations possibles

## Les données :

Les données sont disponible sur le site : <https://www.kaggle.com/c/home-credit-default-risk/data>

Dans le fichier TRAIN, nous retrouvons la variable cible « TARGET » qui servira à l’entrainement du modèle.

* Solvable = 0
* Non-solvable = 1

Dans le fichier TEST, nous retrouvons les mêmes données que dans le fichier TRAIN, mais sans la variable « TARGET »

Nous sommes dans une **problématique de classification avec une cible déséquilibrée** (90/10)

Détail des données :



Pour la préparation des données je me suis inspiré du traitement proposé ici : [LightGBM with Simple Features | Kaggle](https://www.kaggle.com/jsaguiar/lightgbm-with-simple-features)

## Méthodologie d'entraînement du modèle :

Ce n’est pas un modèle qui va être entrainé, mais plusieurs et dans différentes configurations afin de choisir celui qui permet le mieux de répondre à la problématique donnée.

Chaque essai sera comparé aux autres sur une base de donnée test.

Ces essais pouvant être long à réaliser (temps machine), l’entrainement se fera sur un jeu de donnée réduis.

Pour le modèle qui sera le plus pertinent, nous rechercherons ensuite les meilleurs hyper paramètres permettant d’optimiser la prédiction.

### TRAIN / TEST :

Le fichier TEST ne contient pas la variable cible, il ne sera donc pas utilisé pour mesurer la qualité de l’entraiment du modèle.

Nous allons donc utiliser le fichier TRAIN après préparation du jeu de donnée (une ligne correspondant à un client)

La cible étant fortement déséquilibrée, les données TRAIN vont être séparées en 2 data set :

Data set d’entrainement 🡪 train set

Data set de test 🡪 test set

Cette séparation sera réalisée en utilisant le paramètre « Stratify « de la fonction train\_test\_split [[1]](#footnote-1)de Sklearn

train, test = train\_test\_split(df\_data\_train, test\_size=0.2, stratify=df\_data\_train['TARGET'])

Ce paramètre permet de conserver après découpage un ratio cible identique pour les 2 jeux de données.

### Gestion du volume des données :

## Le volume des données étant important, j’ai décidé de travailler pour les tests de modèle sur un échantillon aléatoire pour le train set de 10% .

Cependant la totalité du jeu de donnée sera utilisé pour le test set, et nous utiliseront également la totalité du jeu de donnée pour l’optimisation des hyper paramètres du modèle sectionné.

StratifiedShuffleSplit(n\_splits=1, test\_size=0.9, random\_state=0)

### Gestion du déséquilibre :

Comme nous le verrons dans la section suivante, le choix de la métrique est important pour bien évaluer un modèle sur des données déséquilibrées, cependant il est également possible d’agir sur les données en entrées ou le modèle afin d’optimiser l’apprentissage.

#### Sur échantillonnage :

RandomOverSampler()

La fonction [[2]](#footnote-2)ajoute des échantillons aléatoires de la classe minoritaire afin d’obtenir un data set équilibré.

Le nombre total d’observation du data set augmente

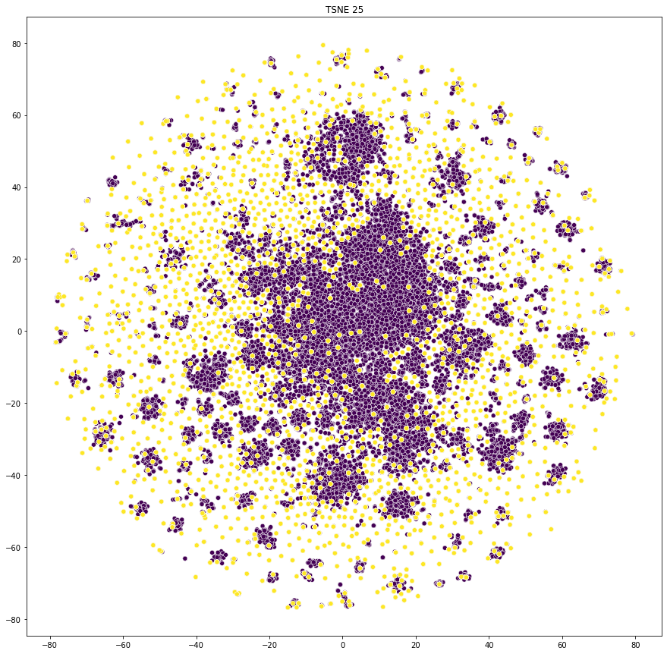


Figure 1: TSNE du train après RandomOverSampler

#### Sous échantillonnage :

RandomUnderSampler()

La fonction [[3]](#footnote-3)retire des échantillons aléatoires de la classe majoritaire afin d’obtenir un data set équilibré.

Le nombre total d’observation du data set diminue

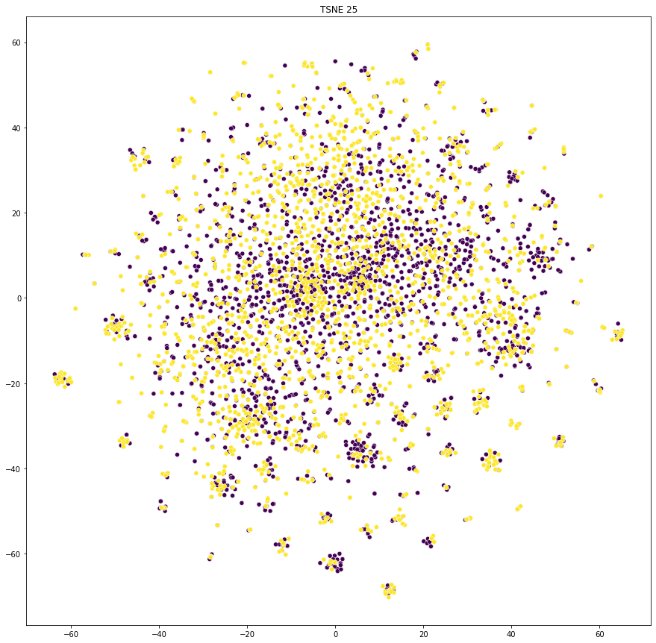


Figure 2Figure 1: TSNE du train après RandomUnderSampler

#### SMOTE : **S**ynthetic **M**inority **O**ver-Sampling **Te**chnique

La fonction[[4]](#footnote-4) génère de nouveaux échantillons en combinant les données de la classe minoritaire avec celles de leurs voisins proches.

Contrairement à la fonction RandomOverSampler, les nouveaux échantillons ne sont pas des copies d’échantillons, mais de nouveaux échantillons ayant des caractéristiques « proches ».

Le nombre total d’observation du data set augmente.

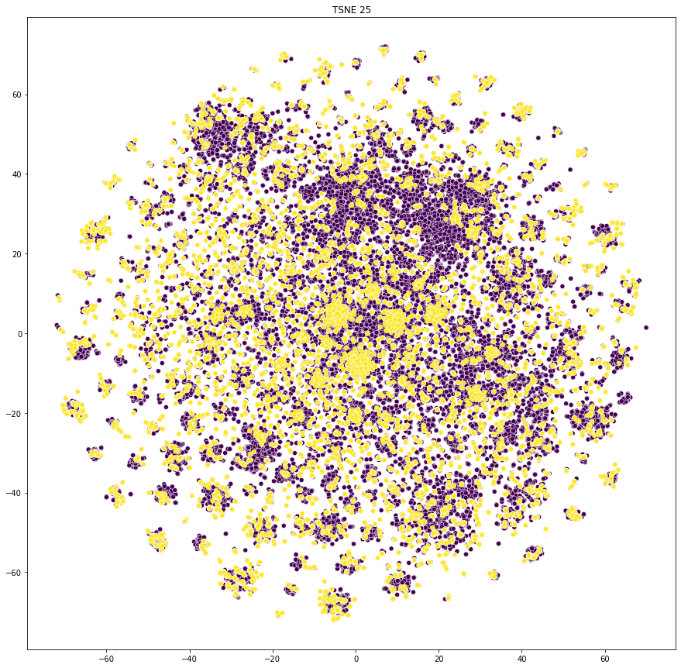


Figure 3 TSNE du train après SMOTE

#### Pondération des classes :

Lors de la phase d’entrainement du modèle, il est possible de spécifier que le jeu de donné est déséquilibré, ceci permet de modifier « artificiellement » la fonction cout en donnant plus ou moins de poids aux prédictions minoritaire.

Ainsi pour LogisticRegression[[5]](#footnote-5) , RandomForestClassifier [[6]](#footnote-6),ou pour LGBMClassifier[[7]](#footnote-7) on retrouve l’hyper paramètre **class\_weight** qui permet de spécifier la gestion du déséquilibre .

### Gestion des valeurs manquantes et mise à l’échelle :

Pour l’utilisation de modèle de classification, il est généralement nécessaire de traiter en amont les valeurs manquantes et de réaliser une mise à l’échelle.

Afin d’éviter une fuite des données ces traitements sont réalisés au sein d’un pipeline[[8]](#footnote-8) contenant également le modèle de classification. Ce pipeline est entrainé sur le train set puis joué sur le test set.

### Modèles testés :

Pour ce problème de classification, J’ai décidé de tester 3 modelés :

* La régression Logistique
* RandomForest Classifier
* LGBM (Light Gradient Boosting Machine)

Ces 3 modèles seront testés avec les paramètre par default sur le jeu de donnée réduis puis avec les 3 méthodes de gestion des données déséquilibré et la gestion de la pondération des classes.

Le modèle avec les meilleurs résultats sera optimisé via la recherche des meilleurs hyper paramètres avec le jeu de donnée total.

1. [sklearn.model\_selection.train\_test\_split — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) [↑](#footnote-ref-1)
2. [RandomOverSampler — Version 0.9.0 (imbalanced-learn.org)](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html) [↑](#footnote-ref-2)
3. [RandomUnderSampler — Version 0.9.0 (imbalanced-learn.org)](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html) [↑](#footnote-ref-3)
4. [2. Over-sampling — Version 0.9.0 (imbalanced-learn.org)](https://imbalanced-learn.org/stable/over_sampling.html#smote-adasyn) [↑](#footnote-ref-4)
5. [sklearn.linear\_model.LogisticRegression — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html) [↑](#footnote-ref-5)
6. [sklearn.ensemble.RandomForestClassifier — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html) [↑](#footnote-ref-6)
7. [lightgbm.LGBMClassifier — LightGBM 3.3.2.99 documentation](https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html) [↑](#footnote-ref-7)
8. [sklearn.pipeline.Pipeline — scikit-learn 1.0.2 documentation](https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html) [↑](#footnote-ref-8)