|  |
| --- |
| **PROJET 7 :** **NOTE MÉTHODOLOGIQUE** |

1. **MÉTHODOLOGIE D’ENTRAÎNEMENT**

Le jeu de données initial a été nettoyé des lignes et colonnes pour lesquelles le taux de remplissage était inférieur à 80%.

Dans le jeu de données initial (train), nous avons séparé :

* X : matrice des variables
* y : vecteur des cibles

Nous avons ensuite séparé en deux :

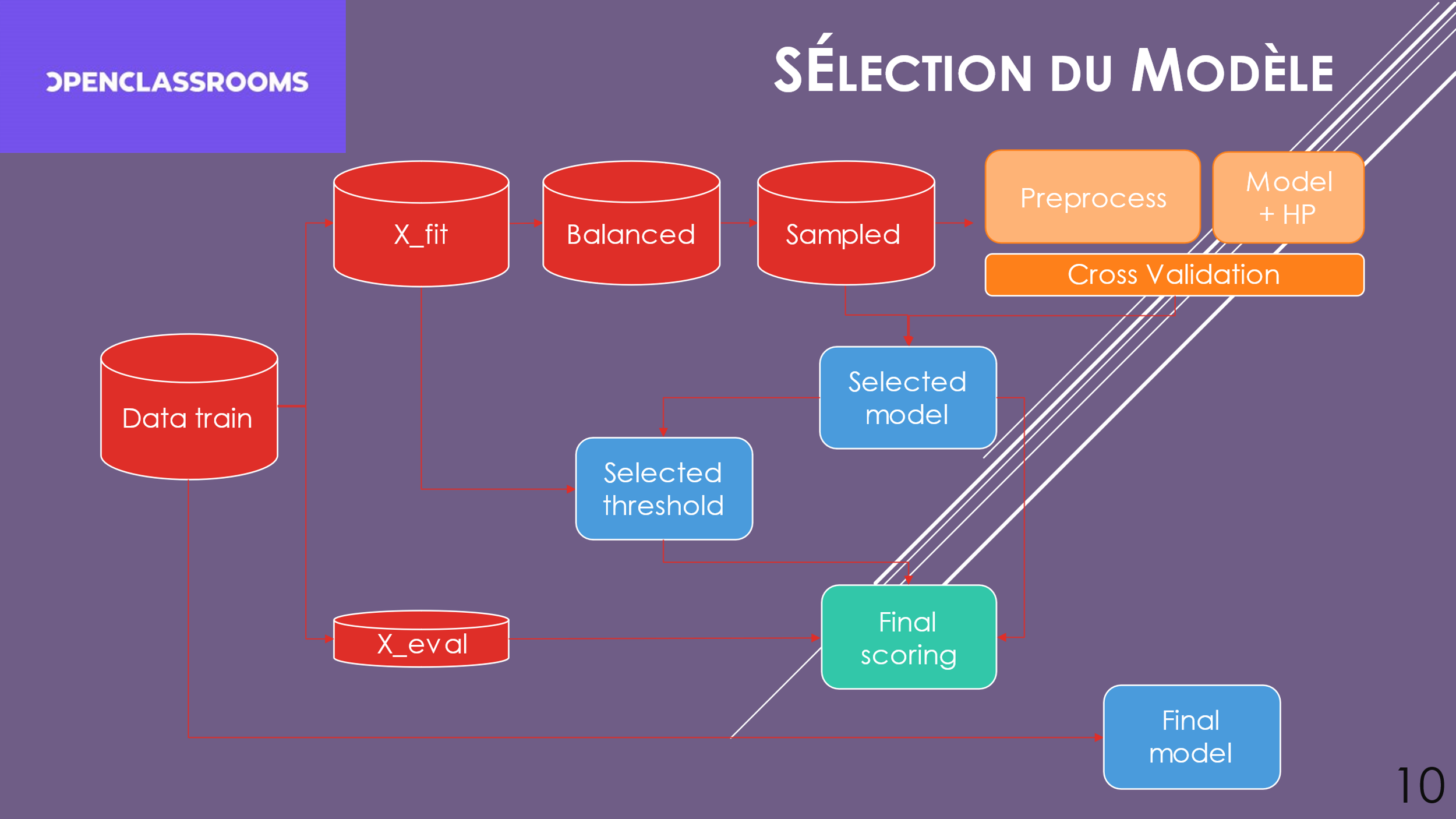
* X\_fit, y\_fit : jeu de données servant à la sélection du modèle et des hyperparamètres
* X\_eval, y\_eval : jeu de données servant à l’évaluation finale du modèle

Les valeurs manquantes résiduelles ont été imputées « par la moyenne » (moyennes calculées sur les données X\_fit, pour éviter la fuite des données).

Pour faciliter l’apprentissage du modèle, nous avons procédé à l’équilibrage des classes par sous-échantillonnage (*downsampling*) de la classe majoritaire.

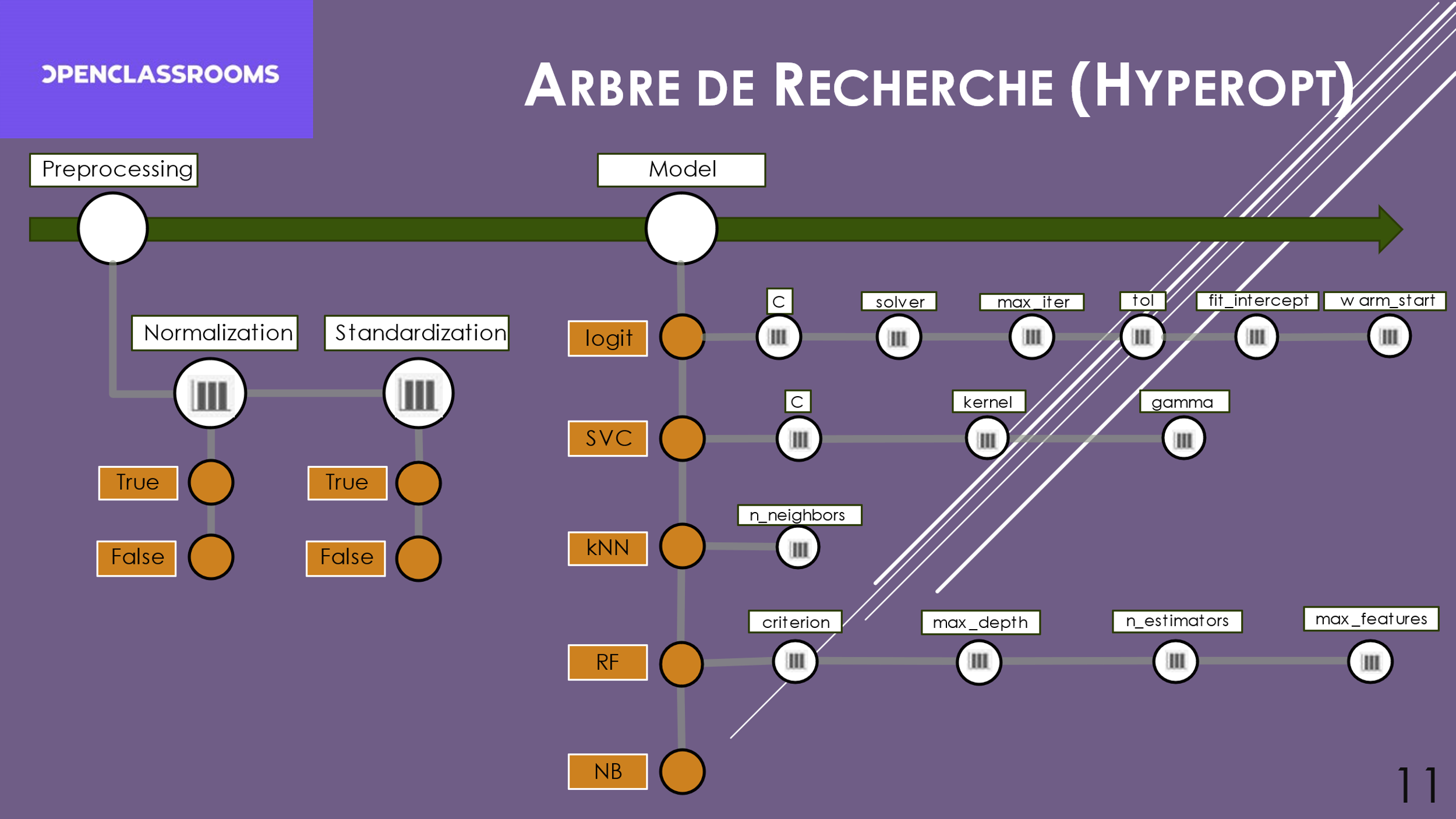
Nous avons prélevé aléatoirement un échantillon de 5000 demandes de crédit dans le jeu (X\_fit, y\_fit) pour accélérer la recherche d’optimisation des modèles et hyperparamètres.

Les autres étapes de prétraitement (normalisation, standardisation), la nature du modèle et les hyperparamètres ont été sélectionnés en utilisant l’algorithme d’optimisation discrète *Tree of Parzen Estimators* (TPE) implémentée dans la librairie Hyperopt.



Les modèles de classification faisant partie de l’espace de recherche testé sont ceux de la bibliothèque scikit-learn. Nous indiquons aussi les distributions statistiques spécifiées pour les hyperparamètres de chaque modèle.

* Bayésien naïf (sklearn.naive\_bayes.GaussianNB)
* SVM (sklearn.svm.SVC)
* Forêt aléatoire(sklearn.ensemble.forest.RandomForestClassifier)
* Régression logistique(sklearn.linear\_model.LogisticRegression)
* kNN (sklearn.neighbors.KNeighborsClassifier)



1. **FONCTION COÛT, ALGORITHME D’OPTIMISATION, MÉTRIQUE D’ÉVALUATION**
2. **Fonctions de pertes**

Les fonctions de pertes utilisées par chacun des modèles listés ci-dessus sont celles implémentées sous scikit-learn.

Certains modèles scikit-learn permettent de modifier la fonction de perte en modifiant un paramètre d’entrée. Lorsque cela était le cas, nous l’avons spécifié comme hyperparamètre dans l’espace de recherche, par exemple pour la forêt aléatoire :

'criterion': hp.choice('rf\_criterion', ["gini", "entropy"])

D’une façon générale, la sélection du modèle est faite en se basant sur l’évaluation, par validation croisée à 3 plis, à l’aide de la métrique personnalisée décrite ci-dessous.

1. **Algorithmes d’optimisation**

Les algorithmes d’optimisation utilisées par chacun des modèles listés ci-dessus sont ceux implémentés sous scikit-learn.

Certains modèles scikit-learn permettent de modifier l’algorithme d’optimisation en modifiant un paramètre d’entrée. Lorsque cela était le cas, nous l’avons spécifié comme hyperparamètre dans l’espace de recherche, par exemple pour la régression logistique :

'solver' : hp.choice('logit\_solver', ['newton-cg', 'lbfgs', 'liblinear']),

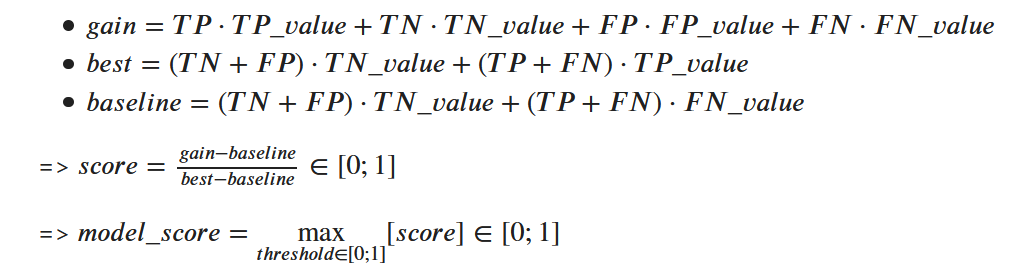
1. **Métrique d’évaluation**

Les classes cibles du jeu de données initial sont très déséquilibrées (plus de 90% des crédits sont remboursés sans défaut). Cela rend la métrique *accuracy* peu pertinente.

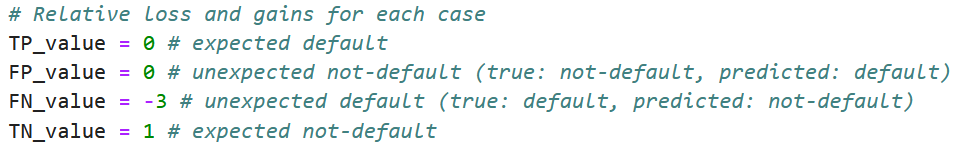
De plus, on peut supposer que l’impact pour l’entreprise d’un faux négatif et ceux d’un faux positif ne sont pas les mêmes, ce qui rend une *AUC* peu pertinente.

* FN, erreur de 1ère espèce :  client à qui l’on refuse un prêt, mais qui l’aurait honoré
* FP, erreur de 2ème espèce : client à qui l’on accorde un prêt, mais qui fait défaut

Nous avons donc construit une métrique d’évaluation « métier » qui a servi à la sélection des modèles et des hyperparamètres lors de l’exécution de l’algorithme TPE (hyperopt).



Cette métrique se base sur 4 paramètres TP\_value, TN\_value, FN\_value et FP\_value qui sont respectivement les coûts associés aux vrais positifs (TP), vrais négatifs (TN), faux négatifs (FN) et faux positifs (FP). Voici les valeurs de ces paramètres que nous avons utilisés pour la construction du modèle. Ces valeurs peuvent facilement être modifiés pour tenir compte de la « réalité métier ».



La valeur de la perte ou de gain pour l’entreprise est fonction du seuil de probabilité de défaut à partir duquel l’entreprise refusera le prêt. C’est donc un hyperparamètre qui est également à optimiser, dans une optique d’aide à la décision, et c’est pourquoi le score du modèle correspond à un maximum lorsque le seuil varie entre 0 et 100%.

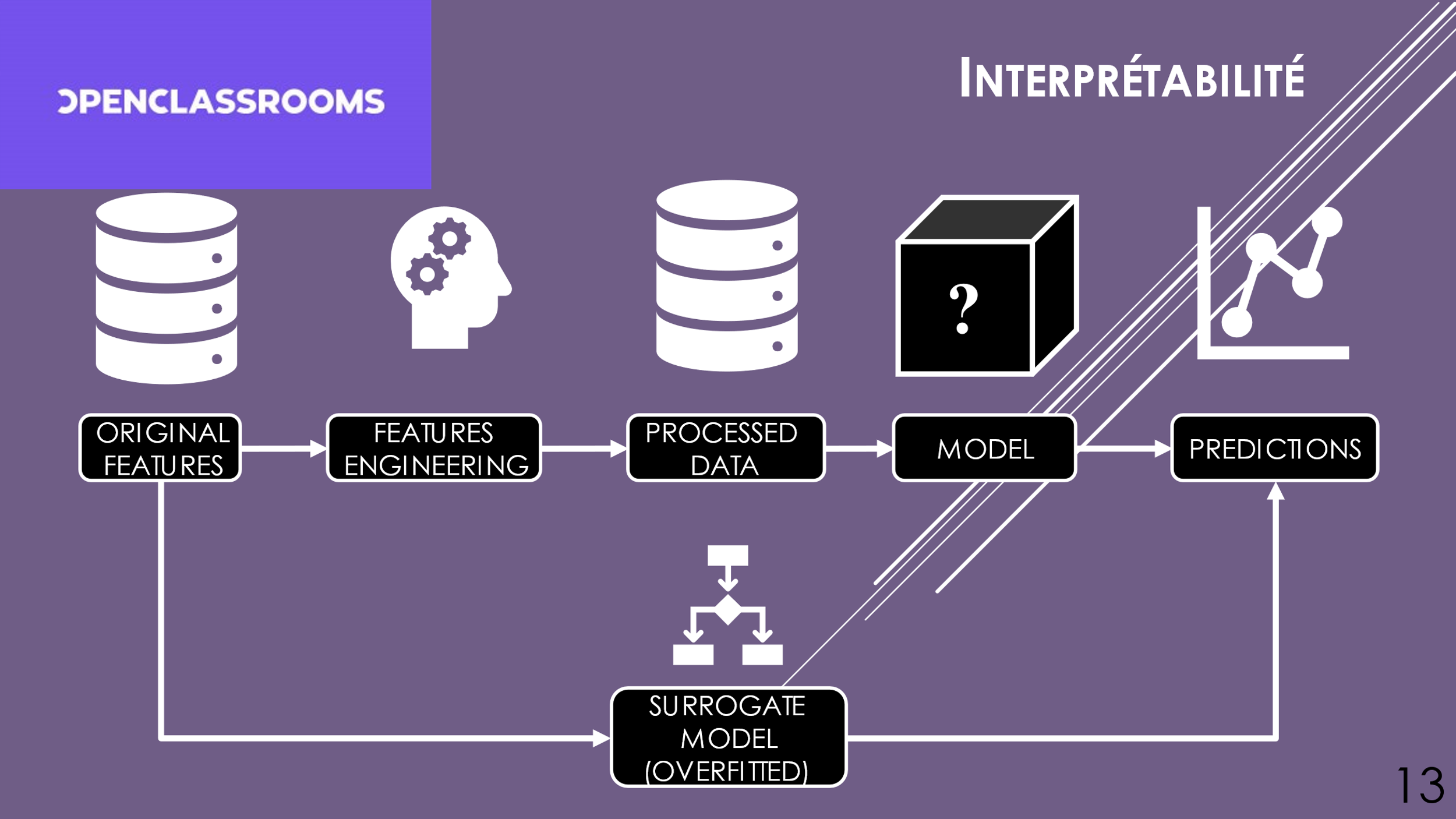
Enfin, cette métrique a été normalisée afin que :

* 0 : représente le score obtenu par un modèle naïf (prédiction systématique de la classe majoritaire N : pas de défaut de paiement).
* 1 : représente le score maximal, obtenu lorsque toutes les prédictions sont correctes.

1. **INTERPRÉTABILITÉ DU MODÈLE**

Le modèle étant sélectionné sur la seule base de ses performances, il n’est pas certain qu’il s’agisse d’un modèle interprétable.

Pour répondre au besoin d’interprétabilité, nous avons donc privilégié des méthodes « indépendantes du modèle » (model-agnostic). Nous avons utilisé la méthode du « surrogate model » (modèle de substitution) qui consiste à entraîner un modèle interprétable sur les prédictions du modèle à interpréter en forçant le sur-apprentissage. Ce modèle (arbre) est alors interprétable : globalement, avec les « features importances », et localement en utilisant la librairie treeinterpreter.



1. **COMPLEXITÉ ALGORITHMIQUE DES MODÈLES**

|  |  |  |
| --- | --- | --- |
| **Modèle** | **Complexité de l’entraînement** | **Complexité de la prédiction** |
| Bayésien naïf | O(nd) | O(cd) |
| SVM | O(n⁴) | ? |
| Forêt aléatoire | O ( nb\_trees m n log n ) | O(height) |
| Régression logistique | linéaire | linéaire |
| kNN | O(1) | O(nd+kn) or O(ndk) |

* N: number of training examples
* d: dimensionality
* c: number of classes
* nb\_trees: number of trees in the forest
* m: max\_features of tree
* height: max height of trees in the forest
* k: number of kneighbors

1. **LIMITES & AMÉLIORATIONS**
2. Modèles et hyperparamètres
3. Un premier axe d’amélioration est d’intégrer dans l’espace de recherche d’autres familles de modèles et notamment :

* Boosting : XGBoost, LightGBM, CatBoost…
* Réseaux neuronaux : fully-connected avec Keras/TensorFlow

1. Un second axe d’amélioration est d’intégrer dans l’espace de recherche d’autres hyperparamètres (par exemple tester l’influence de class\_weight sur la capacité de la régression logistique à mieux détecter la classe « default »). On peut aussi essayer d’intégrer davantage d’étapes de pré-traitement (pour, par exemple comparer les méthodes d’équilibrage des classes, ou les méthodes d’imputation).
2. Un troisième axe d’amélioration est de simplement continuer à itérer l’algorithme TPE (hyperopt), ce qui nécessite du temps. Remarquons que, dans l’implémentation utilisée, les 20 premières itérations sont totalement aléatoires.
3. Interprétabilité
4. La méthode du « surrogate model » permet d’interpréter le modèle de substitution, pas réellement le modèle original. Pour les individus faisant partie du jeu de données d’entraînement, cela ne pose pas de soucis, mais cela peut en poser pour les individus qui ne font pas partie du jeu d’entraînement. Le « surrogate model » peut alors donner des prévisions différentes du modèle initial. Une solution, coûteuse en temps de calcul, serait d’ajouter la nouvelle prédiction au jeu d’entraînement sur lequel entraîner le surrogate model.