Machine Learning

Challenge 1 : Classification de pompes selon leur état de fonctionnement

Julie Courgibet
11/01/2020

Partie I : Nettoyage et préparation des données

Tout d'abord, on peut générer un rapport qui analyse les données du set (pandas_profiling).

On peut ainsi plus aisément voir quelles données sont corrélées à la variable d'intérêt (status_group). Surtout, on observe le nombre de données manquantes (en quelle proportion) et surtout les données aberrantes.

On peut remplacer les données manquantes (NA) par une valeur moyenne (ou la médiane). Ici, j'ai fait le choix de la moyenne.

On peut aussi traiter les données (cas des données de géolocalisation et de gps_height).

On peut garder l'information que si la donnée était manquante en créant des colonnes : si valeur manquante, on attribue la valeur 1 et sinon la valeur 0. On ne perd pas d'information.

La fonction scaling (les données numériques sont mises sur une certaine échelle) améliore aussi les performances du modèle.

Pour les valeurs non numériques, on peut soit utiliser le label encoder, soit la fonction get_dummies de pandas. Il s'agit ici d'un choix important car dans un cas on multiplie le nombre de colonnes par le nombre de valeurs (le temps d'entrainement explose mais les performances du modèle sont meilleures) versus on garde une colonne (temps d'entrainement plus rapide).

Aussi pour les valeurs rares (par exemple apparaissant moins de 20 fois), on peut les associer à la valeur 'Other' (scores plus faibles).

Partie II : Les modèles

C'est un problème de classification avec des classes qui ne sont pas réparties de façon homogène (imbalanced). Différents modèles ont été testés pour améliorer les scores (sans succès).

Avec la librairie imblearn, on peut soit créer de la donnée (over-sampling) pour la/les classe(s) déficitaire(s), soit supprimer de la donnée (under-sampling) pour la/les classe(s) majoritaire(s).

On peut aussi utiliser des modèles linéaires (régression logistique) qui ont donnés des résultats relativement insatisfaisants.

Enfin, les modèles les plus intéressants font partie de la catégorie ensembliste : random forest, XGBoost, AdaBoost. D'autres modèles sont encore possibles (https://scikit-learn.org/stable/modules/ensemble.html) mais n'ont pas été testés.

Le meilleur modèle reste XGBoost.

Partie III : Les paramètres

Pour chaque modèle, il y a différents paramètres. Pour trouver les meilleurs paramètres, on peut utiliser la fonction aléatoire RandomizedSearchCV. Globalement, plus le nombre d'estimateurs est important, plus le temps d'entrainement est long et meilleur est le modèle. Le paramètre max_depth influence aussi les résultats. D'autres paramètres tels gamma et min_child_weight influencent les résultats.

On constate une nette amélioration des scores en jouant avec les paramètres.

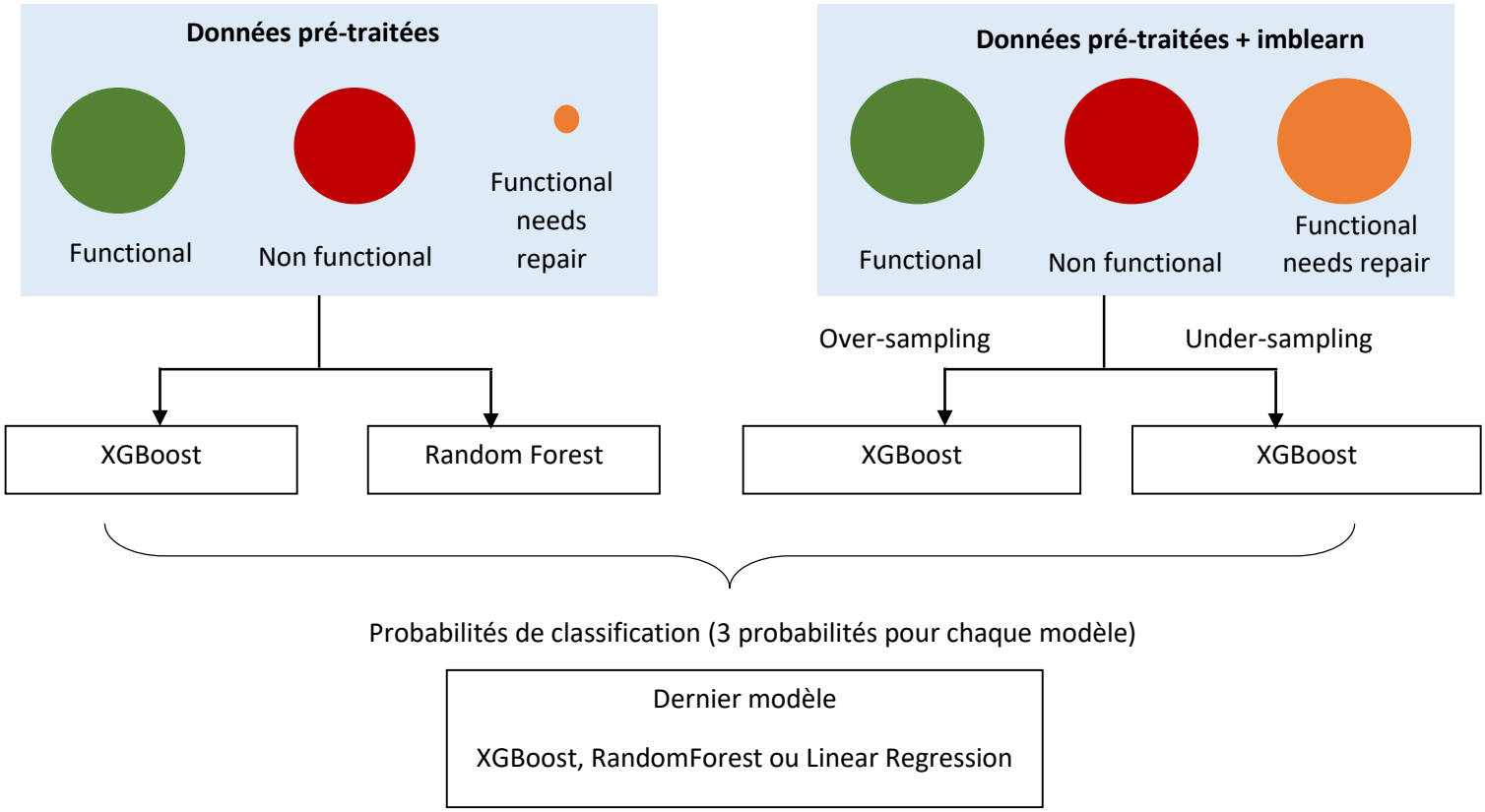
Ces résultats sont confirmés avec la cross-validation, qui nous permet d'évaluer les performances du modèle sur nos données d'entrainement.

Partie IV : Tests et idées

Ici, je présente les différents tests et idées d'amélioration (sans réussite).

Tout d'abord, j'ai accédé aux probabilités de classification des modèles plutôt qu'à la classification finale (predict_proba). J'ai trouvé ce résultat plus parlant.

J'ai aussi voulu créer une sorte de supra-modèle qui se basait sur les prédictions d'autres modèles (comme un XGBoost, un random forest et des modèles basés sur des données over-sampled et under-sampled). Les performances sont restées plus faibles que sur le modèle XGBoost initial.



Pour le dernier modèle, la question du jeu d'entrainement de données s'est posée : faut-il partager les données en 2 jeux d'entrainement (train 1 et train 2) pour l'entrainement des 4 premiers modèles (avec train 1). On utilise ces 4 modèles sur les données train 2 qu'on utilise pour l'entrainement du dernier modèle. On a donc un affaiblissement du modèle (moins de données). Cependant, il me semble peu pertinent d'utiliser les mêmes données pour le dernier modèle (puisqu'on utilise alors les mêmes données pour entrainer et générer des prédictions).

Enfin, j'ai aussi voulu générer des modèles à deux classes soit en supprimant les données d'une des classes (functional versus non functional, functional versus functional needs repair et non functional versus functional needs repair). Soit aussi en associant une classe à une autre (donc sans supprimer de données). Cela semble peu pertinent car déjà pris en compte avec les modèles ensemblistes.