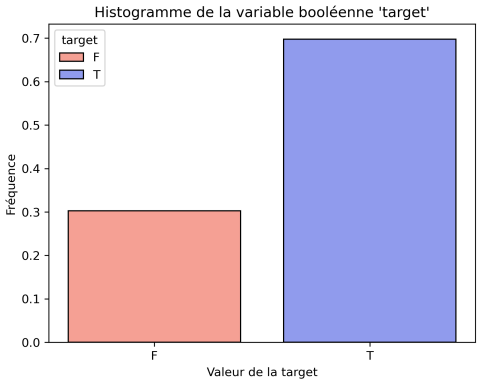
Projet Apprentissage Supervisé

Table of contents

1 Introduction

La variable d'interêt est une variable booléenne synthétique (elle ne représente pas une caractéristique empiriquement observable).

Elle présente un déséquilibre de classes selon 70% de *true* et 30% de *false*.



Notre dataset principal, *learn_dataset*, renseigne 50044 individus, il ne contient pas de valeurs manquantes.

Les autres datasets sont de deux types différents:

- Ceux donnant des détails supplémentaires sur certains individus en fonction de leur statut, ils contiennent des valeurs manquantes. La variable clé est *Unique_id*.

- Ceux donnant des renseignements supplémentaires non directement reliés aux individus par la clé *Unique_id*. Par exemple, des caractéristiques des communes ou bien des détails supplémentaires sur les nomenclatures utilisées par l'INSEE.

Nous présentons tout d'abord un modèle avec tous les individus présents dans *dataset_learn* (modèle *all*), auxquels nous ajoutons les informations des datasets suivants:
- *learn_dataset_sport*
- *city_adm*
- *departments*
- *city_pop*

Ce modèle est en quelque sorte notre *baseline*. Notre objectif étant de faire mieux que cette *baseline* avec les modèles plus élaborés.
Nous avons essayé plusieurs approches différentes (nous en détaillerons certains éléments en fin de rapport) pour aboutir à une utilisation de trois modèles.

Stratégie retenue :
Parmi ces trois modèles différents, le premier concerne les individus en emploi (modèle *emploi*), le deuxième concerne ceux à la retraite (modèle *retired*), le troisième concerne le reliquat des individus non pris en compte par les deux modèles précédents (modèle *nini*, pour ni en emploi ni à la retraite).

- Utilisation des datasets

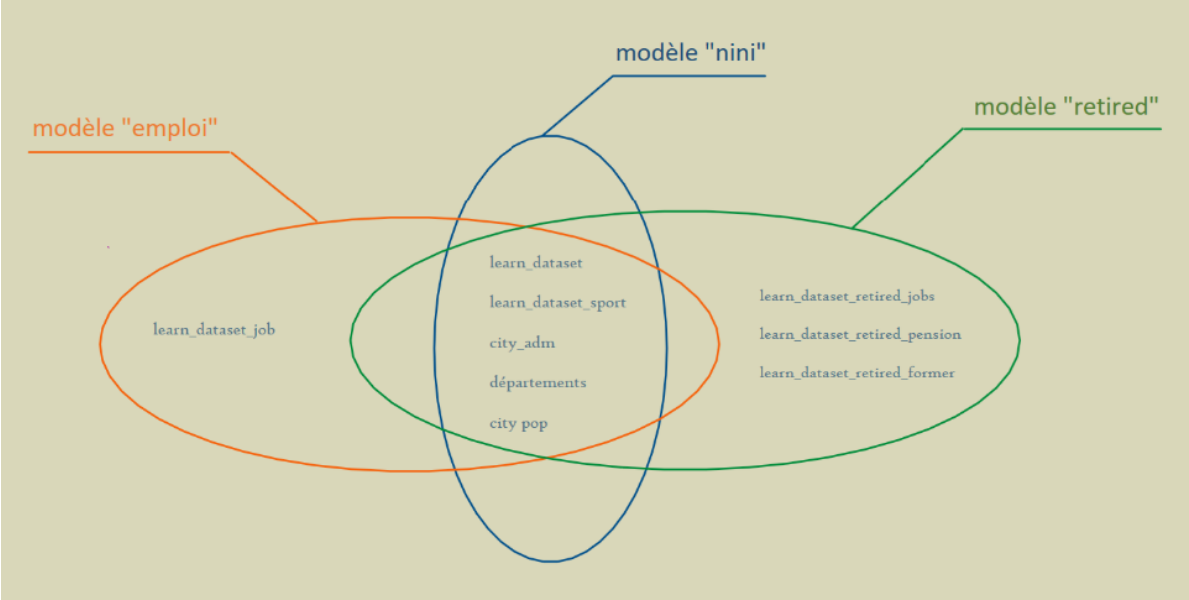


Diagramme de l'utilisation des datasets en fonction du modèle

- Répartition des individus

Le modèle *emploi* concerne les individus renseignés dans *learn_dataset_job*. Le modèle *retired* concerne les individus qui sont renseignés dans *learn_dataset_retired_jobs* et *learn_dataset_retired_pension*, nous y ajoutons les informations de *learn_dataset_retired_former*. Notons que *learn_dataset_retired_jobs* et *learn_dataset_retired_pension* concernent les mêmes individus et que tous ces individus sont présents dans *learn_dataset_retired_former* (autrement dit la population de *learn_dataset_retired_former* inclut celle de *learn_dataset_retired_jobs*) .

Pour chaque modèle, nous effectuons un Random Forest et un XGBoost. Nous avons constaté qu'à chaque fois le XGB obtient de meilleurs résultats et en un temps plus court, ce qui explique que les raffinements et la recherche d'hyperparamètres seront plus poussés sur les algorithmes de XGB tout au long du projet.

2 Détails des résultats des modèles

2.1 Remarque concernant tous les modèles

Dans un premier temps, nous avons mis en place un modèle initial et simple, “de travail”, sans recherche spécifique d’optimisation des hyperparamètres. Ce modèle a permis de vérifier l’absence d’erreurs au sein du pipeline.

Ensuite, nous avons lancé des modèles avec une grille d’hyperparamètres relativement complète, en utilisant des environnements tels que Kaggle ou Colab pour profiter de leurs ressources de calcul. Les paramètres sélectionnés à l’issue de cette étape ont été intégrés dans notre notebook.

Dans la plupart des cas, nous avons poursuivi par un affinage manuel des hyperparamètres directement au sein du notebook pour ajuster les performances finales des modèles.

Exemple de grilles utilisées sur Kaggle et Colab:

Grille RF	Grille XGB
‘classifier___n_estimators’: [50, 100, 300] ‘classifier___max_depth’: [10, 30, 50] ‘classifier___min_samples_split’: [2, 10, 20] ‘classifier___min_samples_leaf’: [1, 2, 5] ‘classifier___max_features’: [0.3, 0.5, ‘sqrt’] ‘classifier___class_weight’: [None, ‘balanced’, ‘balanced_subsample’]	‘n_estimators’: [50, 100, 200] ‘max_depth’: [3, 5, 7] ‘learning_rate’: [0.01, 0.1, 0.2] ‘subsample’: [0.6, 0.8, 1.0] ‘colsample_bytree’: [0.6, 0.8, 1.0] ‘gamma’: [0, 1, 5] ‘lambda’: [1, 5, 10] ‘alpha’: [0, 1, 5]

2.2 Notre *baseline:* modèle *all*

Nous commençons par concevoir un pipeline de préparation des données, prenant en entrée tous les dataframes nécessaires et retournant les ensembles suivants : *X_train*, *X_val, y_train* et *y_val*. Ce pipeline est commun aux modèles RF et XGB.

Le pipeline réalise les jointures nécessaires entre le dataframe principal et les dataframes requis pour le modèle all. La variable cible est transformée en booléen.

Remarque: nous ne transformons pas la variable *sex* en booléen, nous avons essayé et le résultat est très légérement meilleur en la laissant en catégorielle (la différence est néanmoins négligeable).

Le numéro de la région (*REG*) est transformé en catégorielle.

Nous supprimons les colonnes *Unique_id*, *Insee*, *Nom de la commune et Nom du département; Unique_id* n'ayant pas d'interêt statistique*, le* nom du département étant redondant avec le numéro du département. Concernant le nom de la commune (et le code Insee affairant), on pourrait éventuellement conserver la variable (dans la population proposée, certains individus résident dans la même commune) mais au prix d'un nombre de modalités très élevé, nous avons choisi de l'écarter.

Les seules valeurs manquantes concernent la variable *club* (87% de valeurs manquantes).

Nous avons choisi pour le split *train/val* un rapport 80/20, le nombre d'observations total permettant de limiter la proportion d'observations mises de coté pour l'ensemble de validation. Nous utilisons une *cross-validation* sur 5 *folds*.

Nous stratifions sur la variable cible.

Ces étapes seront reprises dans les autres pipeline de préparation des données.

2.2.1 Random Forest *all*

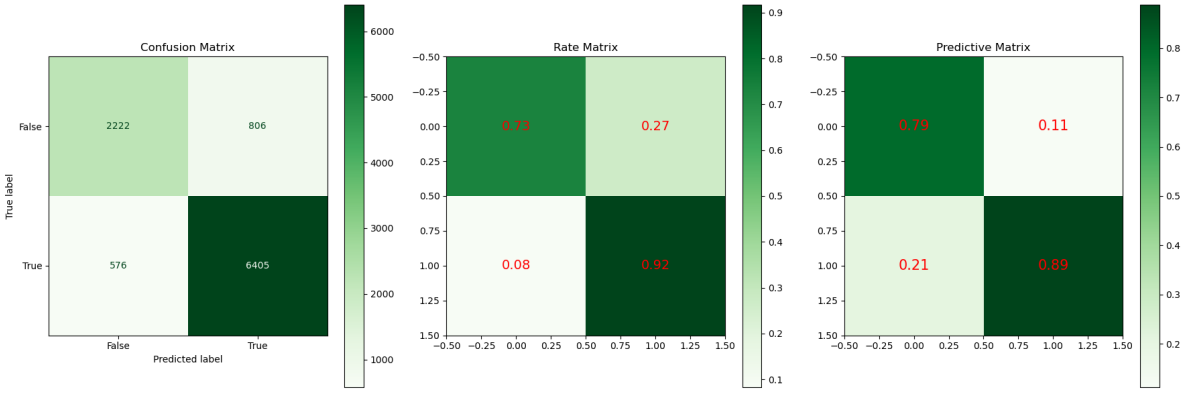
Nous ne faisons pas d'imputation sur les valeurs manquantes de la variable *club*, les lignes non renseignées sont rassemblées dans une colonne après One Hot Encoding (OHE).

Nous utilisons *roc_auc* comme métrique de scoring à optimiser. Nous avons hésité avec le F1 score mais les essais ont montré que les résultats étaient quasi-identiques. Nous surveillons surtout les matrices de confusion pour guider nos choix d'hyperparamètres.

Le nombre de variables après OHE est de 283 pour 40035 observations dans X_train (après le *split*).

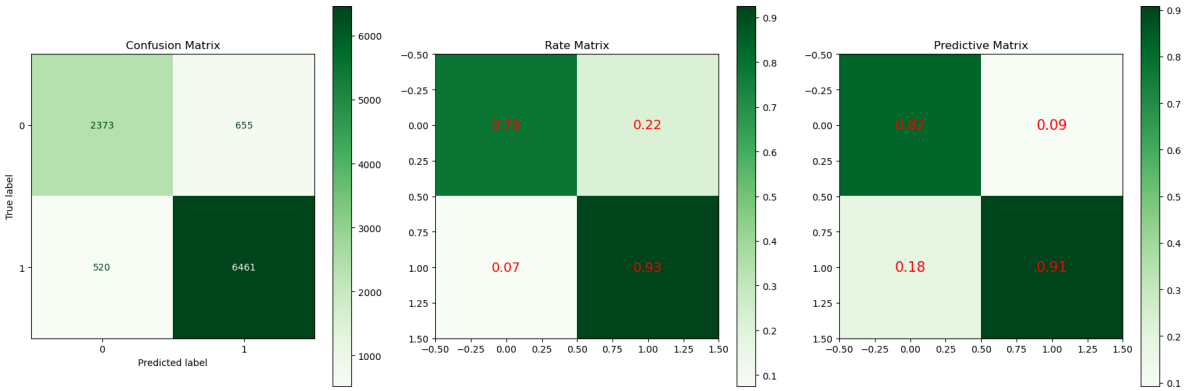
Hyperparamètres retenus, métriques et matrices de confusion:

param_grid	métriques
'classifier___n_estimators': [100]	train roc_auc: 0.9281
'classifier___max_depth': [30]	val roc_auc: 0.8257
'classifier___min_samples_split': [10]	train accuracy: 0.8604
'classifier___min_samples_leaf': [2]	Val accuracy: 0.8619
'classifier___max_features': [0.3]	
'classifier___class_weight': [None]	



2.2.2 XGBoost *all*

param_grid	métriques
'classifier___n_estimators': [300]	train roc_auc: 0.9488
'classifier___max_depth': [10]	val roc_auc: 0.8546
'classifier___learning_rate': [0.1]	train accuracy: 0.8838
'classifier___subsample': [0.8]	val accuracy: 0.8826
'classifier___colsample_bytree': [0.8]	
'classifier___gamma': [0.5]	
'classifier___lambda': [1]	
'classifier___alpha': [0]	



Comparé au RF, le modèle XGB améliore les scores sur toutes les métriques.

Notre *baseline* situe la performance à dépasser à environ 88% (accuracy sur l'ensemble de validation).

Note: Pour la suite du rapport, et afin de ne pas le surcharger inutilement, nous présenterons uniquement les résultats des modèles XGB. Ceux concernant les modèles RF sont présentés dans le *notebook* associé à ce rapport.

2.3 Modèle *emploi*

En complément des étapes déjà décrites pour le modèle *all*, nous avons uniquement ajouté une jointure avec le dataset *learn_dataset_job* et en ne retenant que les individus renseignés dans ce dernier.

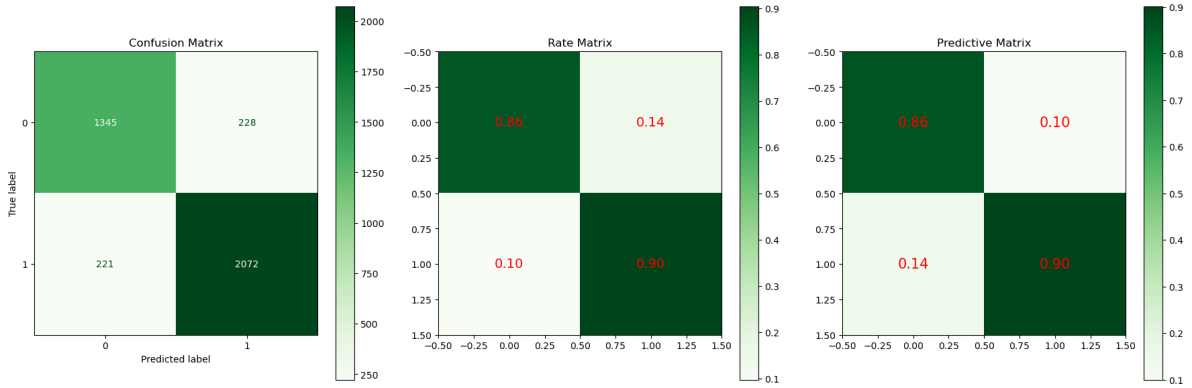
Nous avons également testé des jointures avec le dataset *learn_dataset_emp* (dont la population inclut celle de *learn_dataset_job*). Bien que l'accuracy en cross-validation ait légèrement augmenté (d'environ 0.01), l'accuracy sur l'ensemble de validation a diminué, passant en deçà de celle obtenue sans *learn_dataset_emp*. Afin de limiter le risque de surapprentissage, ce dataset a finalement été exclu du pipeline.

Par ailleurs, nous avons géré les valeurs manquantes de la variable numérique *Working_hours*. Dans un premier temps, nous avons utilisé la méthode *IterativeImputer* de scikit-learn, qui repose sur une approche itérative par régression pour prédire les valeurs manquantes.

Nous avons ensuite comparé les résultats obtenus avec cette méthode à ceux issus de la méthode d'imputation native intégrée à XGB. Les performances des deux approches sont très proches, mais la méthode native s'est légèrement démarquée, notamment quant à la vitesse d'exécution. Nous avons donc choisi de retenir cette dernière.

Le nombre de variables après OHE est de 829 pour 15463 observations dans X_train.

param_grid	métriques
'classifier___n_estimators': [1000]	train roc_auc: 0.9531
'classifier___max_depth': [10,]	val roc_auc: 0.8793
'classifier___learning_rate': [0.05]	train accuracy: 0.8811
'classifier___subsample': [0.9]	val accuracy: 0.8839
'classifier___colsample_bytree': [0.7]	
'classifier___gamma': [0.4]	
'classifier___lambda': [0.5]	
'classifier___alpha': [0.1]	



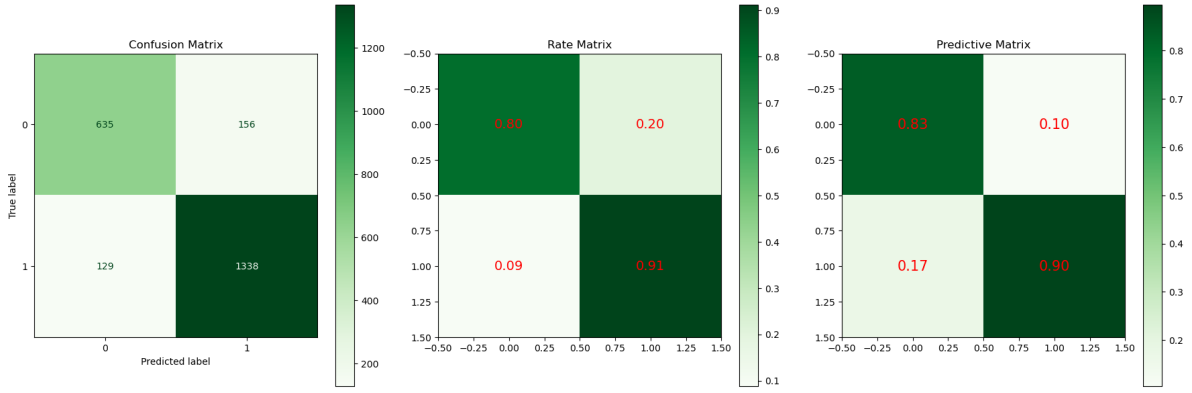
Avec cette approche, nous n'améliorons que marginalement notre *baseline.*

Par contre, l'équilbre entre recall et spécificité est nettement amélioré.

2.4 Modèle *retired*

Le nombre de variables après OHE est de 897 pour 9028 observations dans X_train.

param_grid	métriques
'classifier___n_estimators': [10000]	train roc_auc: 0.9522
'classifier___max_depth': [10]	val roc_auc: 0.8574
'classifier___learning_rate': [0.04]	train accuracy: 0.8804
'classifier___subsample': [0.9]	val accuracy: 0.8738
'classifier___colsample_bytree': [0.8]	
'classifier___gamma': [0.5]	
'classifier___lambda': [1]	
'classifier___alpha': [0.1]	

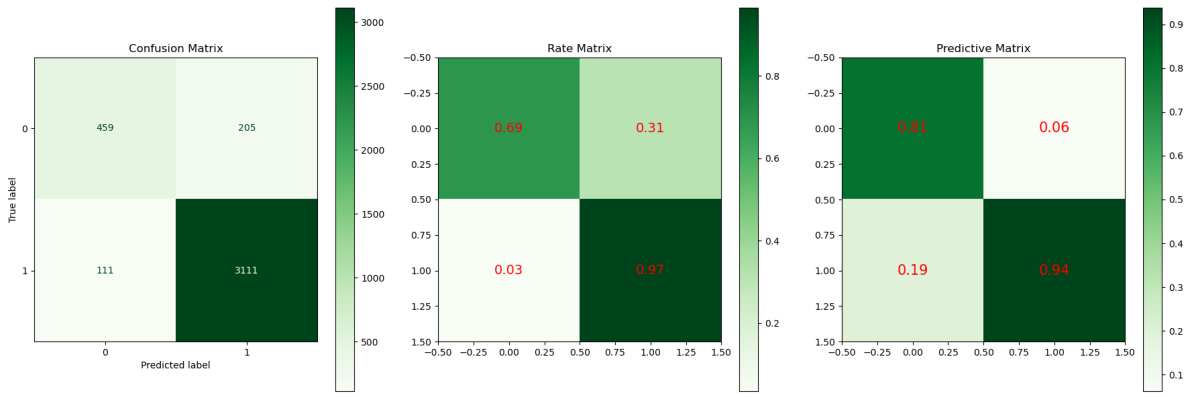


Les résultats obtenus sont légèrement inférieurs à ceux de la *baseline*. Cependant, la sous-population ciblée par le modèle *retired* est la plus petite des trois sous-populations, et, malgré des performances moindres, les résultats restent comparables à ceux de la *baseline*.

2.5 Modèle *nini*

Le nombre de variables après OHE est de 277 pour 15543 observations dans X_train.

param_grid	métriques
'classifier___n_estimators': [1000]	train roc_auc: 0.9516
'classifier___max_depth': [10]	val roc_auc: 0.8278
'classifier___learning_rate': [0.1]	train accuracy: 0.9110
'classifier___subsample': [0.9]	val accuracy: 0.9197
'classifier___colsample_bytree': [0.5]	
'classifier___gamma': [0.6]	
'classifier___lambda': [0.1]	
'classifier___alpha': [0.1]	



C'est sur cette sous-population que nous observons le gain le plus marqué par rapport à la *baseline*. Le recall est très élevé (97 %), mais cela se fait au détriment de la spécificité (69 %). Néanmoins, l'accuracy globale est améliorée.

3 Prédictions

Le dataset *test_dataset* contient 50 043 observations pour lesquelles nous devons prédire la variable cible.

En appliquant les mêmes transformations que celles réalisées sur le dataset train, nous avons généré les prédictions à partir des trois modèles XGB retenus.

L'accuracy que nous obtenons est estimée à 89,55 %. (Ce score a été calculé par somme pondérée des résultats des trois modèles retenus.)

4 Autres stratégies envisagées

4.1 Modèle *nini* alternatif

Nous avons testé une alternative à la stratégie exposée plus haut concernant les individus du modèle *nini* : récupérer les données de ces individus à partir d'un modèle entraîné sur l'ensemble de la population.

Le choix revient ici à comparer deux approches: un modèle plus spécialisé (entraîné uniquement sur les individus *nini*) ou un modèle moins spécialisé mais plus exhaustif en termes des relations que l'algorithme peut détecter.

Un autre critère de sélection entre ces deux possibilités est la taille de l'échantillon des individus concernés: dans le cas où les individus "réellement" *nini* seraient peu nombreux, il serait pertinent de privilégier l'extraction des résultats depuis un modèle construit sur l'ensemble de la population.

Bien que les résultats du modèle alternatif soient légèrement inférieurs à ceux obtenus avec le modèle nini "classique" présenté ici, ils restent néanmoins comparables.

4.2 Données géographiques

En utilisant les données du dataset *city_loc*, nous avons affiché, via Plotly, toutes les positions géographiques des individus sur une carte de France (en distinguant les observations selon la variable d'intérêt). Aucun schéma ou pattern significatif n'a émergé.

Dans une autre tentative d'exploiter les positions géographiques, nous avons ajouté une variable représentant la distance à la préfecture (*distance_pref*) pour chaque individu. Nous avons ensuite entraîné un modèle en utilisant uniquement cette nouvelle variable, mais les résultats se sont révélés très faibles.

Lorsque cette variable a été intégrée dans le modèle *all*, nous avons observé une légère baisse de performance par rapport à la version sans cette variable.

Par conséquent, nous ne l'avons pas retenu dans le modèle final.

4.3 Features engineering plus poussé

Nous avons essayé plusieurs approches différentes comme:

- Rassembler des modalités qui paraissent très proches entre elles au sein de certaines variables, notamment les modalités concernant le niveau de formation.
- Réunir les modalités des variables en appliquant une analyse par décile en fonction de la variable cible, puis extraire les modalités les plus discriminantes pour la cible et éliminer les autres.
- Sélectionner les variables les plus importantes sur un modèle saturé. Nous avons utilisé les *Shap Values* sur le modèle RFF all et la fonction importance de XGB sur le modèle XGB *emploi*. Cette dernière approche semble plus prometteuse et nous permet de nous rapprocher rapidement des performances du modèle saturé.
- Utiliser des algorithmes de réduction de dimensions (UMAP sur *learn_dataset_job* dans notre cas) pour essayer des modèles avec moins de variables. La variable cible n'était pas suffisamment corrélée avec les dimensions pour poursuivre dans cette voie.

Ces différentes approches donnent des résultats intéressants (hormis la réduction de dimension). Cependant, elles sont difficiles à intégrer à un pipeline et notre stratégie retenue (i.e.: en conservant la plupart des variables) ne montre pas de tendance à l'overfitting sur la métrique accuracy. Nous n'avons donc pas retenu ces approches en définitive.

5 Conclusion

Notre *baseline* initiale n'est battue que de peu par notre stratégie à trois modèles. Cela montre qu'un modèle simple, sans pre-processing avancé, obtient des résultats difficiles à augmenter par la suite. (Gardons à l'esprit que cette remarque concerne les données et le cahier des charges de ce projet et qu'elle pourrait ne pas s'appliquer à d'autres contextes.)

Autre constat: la recherche poussée d'hyperparamètres n'améliore les performances que de manière margniale.