**ITF-712 Techniques d’apprentissage**

Rapport Projet de session

6 algorithmes de classifications

Leaf-Classification

Thibault Languillat – 21204590

18 avril 2022

# Introduction

L’objectif de ce devoir de session est de démontrer l’application de 6 algorithmes de classifications sur un dataset spécifique. Le dataset provenant de Kaggle proposé, leaf-classification, a été utilisé. Les détails de ce dataset seront expliqués dans la section exploration des données. 6 algorithmes ont été évalué et optimisé pour cet exercice :

* Régression logistique
* Arbre de décision
* Multi-layer Perceptron
* Random Forest
* Support Vector Machine
* KNeighbors

Ces algorithmes ont été optimisés dans la mesure de possible des ressources disponibles pour l’exploration des hyperparamètres. Les algorithmes utilisés seront détaillés dans la section algorithmes.

Les résultats des différents algorithmes seront présentés avec l’impact que les différents pré-traitements appliqués sur les données ont pu avoir.

Le code source est hébergé dans le repo suivant :

<https://github.com/ThibLang/ITF712-Classification-Project>

# Exploration des données

## Exploration initiale

Le dataset est composé de 1593 entrées, divisées en 2 sections. La section d’entraînement contient 990 éléments et la section test 594 éléments.

Figure 1- Répartition des données

La différence entre les 2 sections est que la section test ne contient pas les labels, les résultats de tests ne sont obtenus qu’en soumettant un fichier de probabilités à la compétition. Dans le cadre d’une utilisation locale, cette section est inutilisable.

Le dataset contient 192 attributs, répartit équitablement dans 3 sections :

* Margin
* Shape
* Texture

Le dataset contient 99 classes. Dans le dataset d’entraînement, chaque classe à 10 éléments.

## Séparation entraînement/test

Avant d’aller plus loin dans l’analyse des données, il a été pris pour acquis que le dataset serait utilisé localement, et donc que la section test de cette compétition serait inutilisable. Pour permettre de valider le bon fonctionnement des algorithmes de classifications, nous avons séparés les données d’entraînement en 2 sections, 80% pour la section d’entraînement et 20% pour la section de test. Cette séparation a été faite en respectant la répartition de classes de l’ensemble du dataset. Les données ont aussi été mélangées avant la séparation. C’est la méthode StratifiedShuffleSplit de Sklearn qui a été utilisé, comme on peut le voir sur la figure ci-dessous.

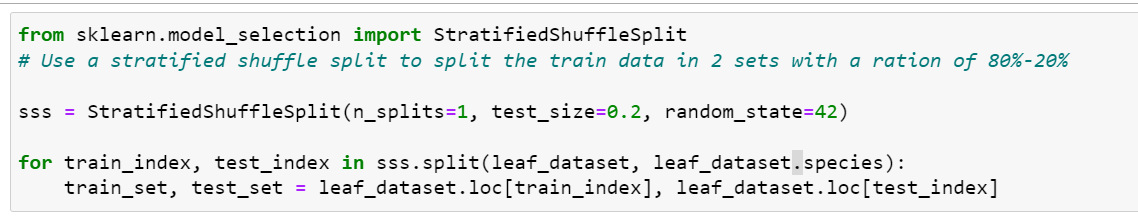


Figure 2- Séparation des données

Pour le reste du document, le dataset d’entrainement et de test feront références à ces sous datasets du dataset d’entraînement initial.

## Exploration approfondie

Le reste de l’analyse des données a été effectué sur le dataset d’entraînement uniquement. Cela permet d’éviter le transfert d’information du dataset de test vers le dataset d’entrainement. L’ensemble des figures de cette sections sont tirés du notebook InitialDataExploration dans le dossier notebook.

Dans la figure ci-dessous, la répartition des valeurs de chacune des caractéristiques a été tracé. On peut remarquer que toutes les caractéristiques ont des répartitions semblables, centrés sur la gauche. Avec ces histogrammes, ainsi que la description du dataset on voit que les données ont une plage similaire, allant de 0 à environ 0.8. Les plages de valeurs sont similaires pour les caractéristiques d’un même groupe (Margin, Shape, Texture), mais pas entre les groupes eux-mêmes. On observe aussi plusieurs histogrammes ne contenant que des données très proche de 0.

À partir de ces informations, on peut envisager plusieurs transformations. Tout d’abord, il serait idéal que la plage de valeur soit la même pour l’ensemble des caractéristiques du dataset. Ensuite, des histogramme « tail-heavy » peut être le symptôme de données aberrantes, agrandissant artificiellement la queue de l’histogramme. Il pourrait donc être intéressant de procéder à une élimination des données aberrantes.

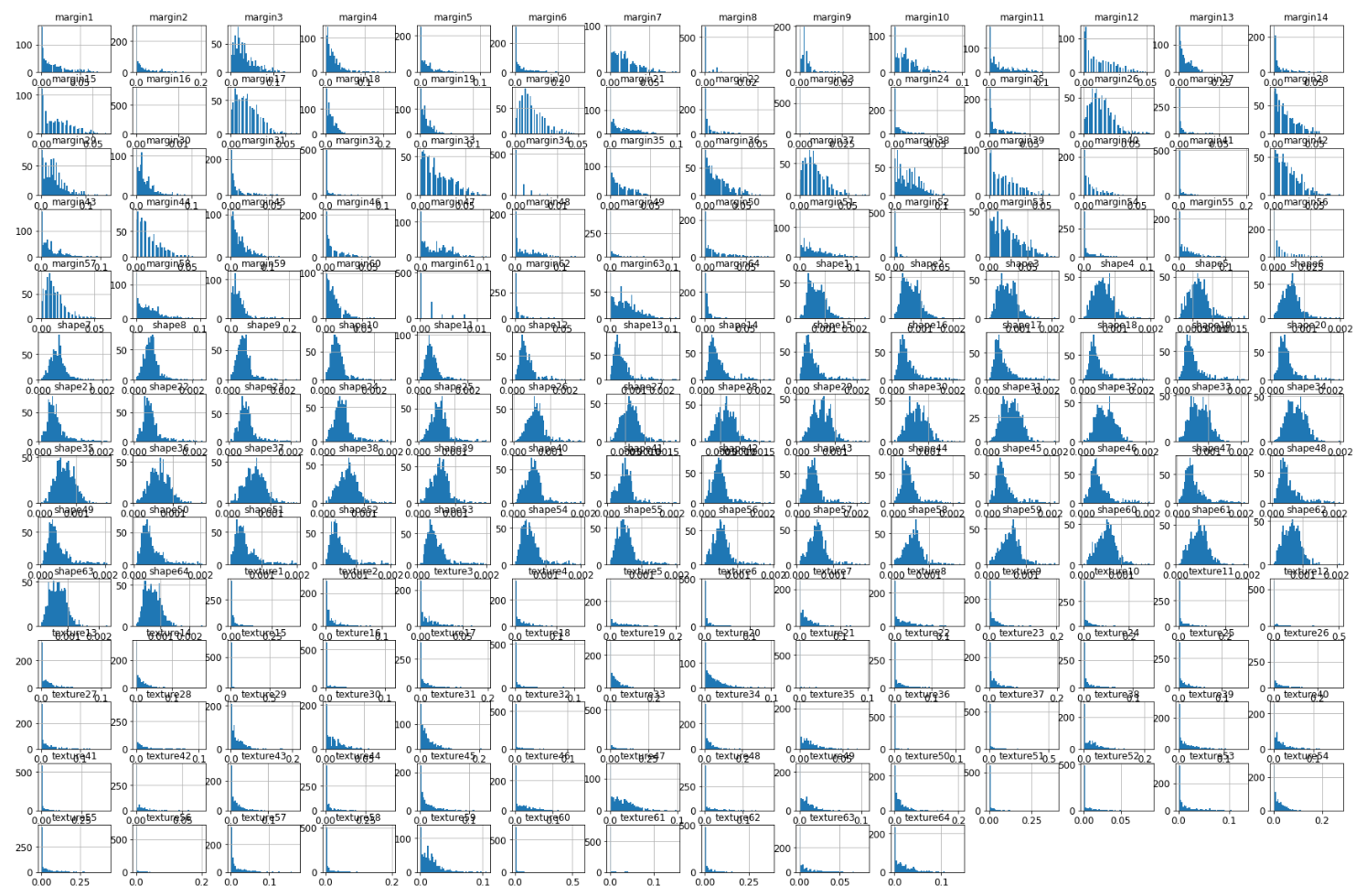


Figure 3- Histograme de chaques caractéristiques

La matrice de corrélation, figure ci-dessous, montre aussi des informations intéressantes. Plus le coefficient de corrélation est proche de 1 ou -1, plus les variables sont corrélées. C’est-à-dire qu’un changement de valeur de l’une, va impacter de la même façon la seconde variable. La fonction corr() de pandas permet de calculer le coefficient de corrélation linaire entre chacune des variables. Il pourrait y avoir d’autres corrélations plus complexes, mais nous ne les avons pas recherchées. On peut remarquer qu’il y a beaucoup de corrélations entre les caractéristiques d’un même groupe, que c’est particulièrement présent dans le groupe Shape. On peut envisager de fusionner les caractéristiques qui ont des coefficients de corrélation très élevé. Cela permettrait sans doute d’alléger l’entraînement tout en gardant le même niveau de résultat.

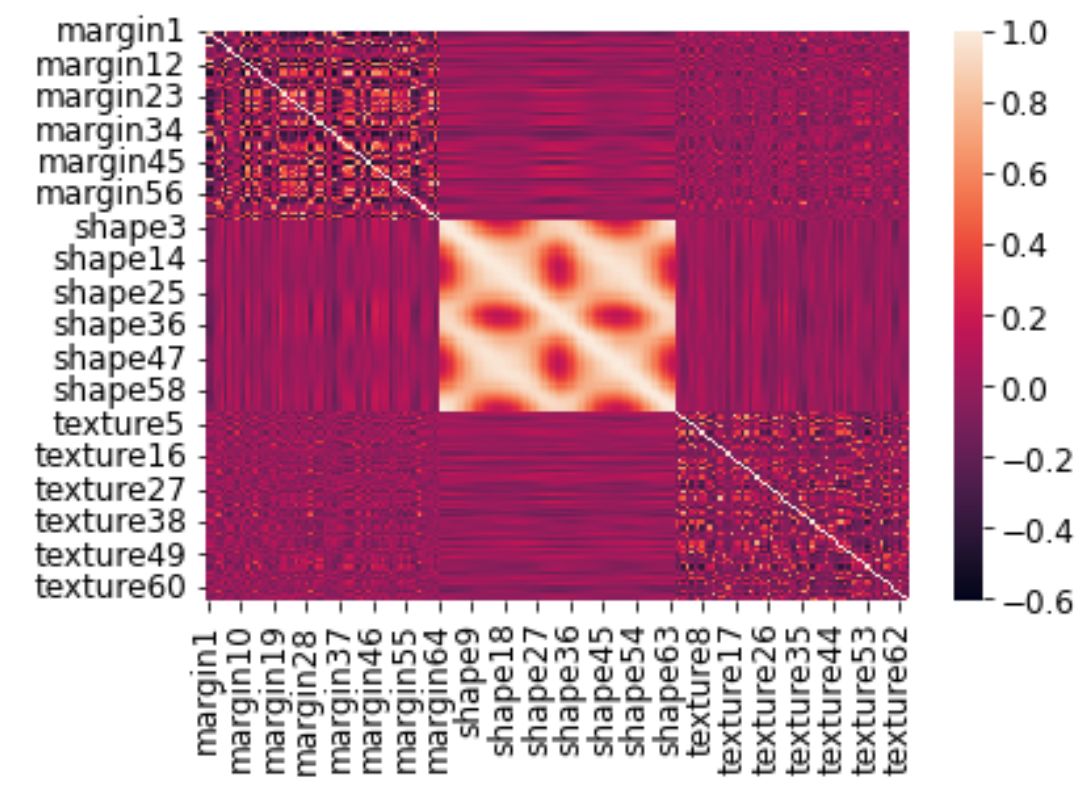


Figure 4-Matrice de corrélation

Les 3 histogrammes suivant on pour objectif de visualiser les répartitions des valeurs dans chacun des groupes. Comme ils n’ont pas la même plage de valeur, c’est plus lisible de les visualiser séparément. On remarque que les groupe Margin et le groupe Texture contiennent plusieurs données s’éloignant de la moyenne. Par contre, les données du groupe Shape semblent mieux concentrées. Il y aurait donc quelques données anormales à éliminer du dataset mais la très grande majorité des données sont valables. On remarque que le dataset ne contient aucune valeur aberrantes, c-à-d ridiculement grande par rapport aux autres, ni aucune donnée manquante.

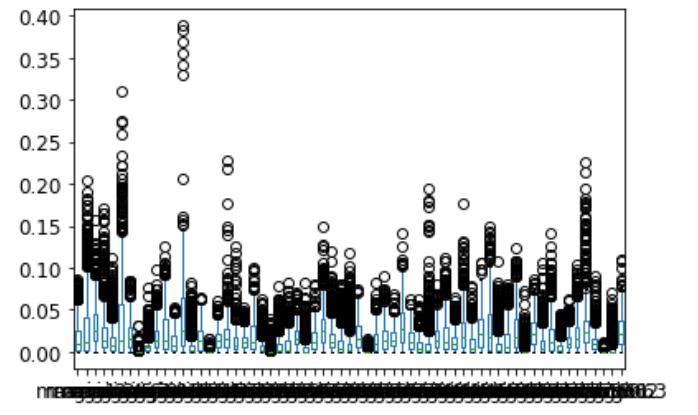


Figure 5-Groupe Margin

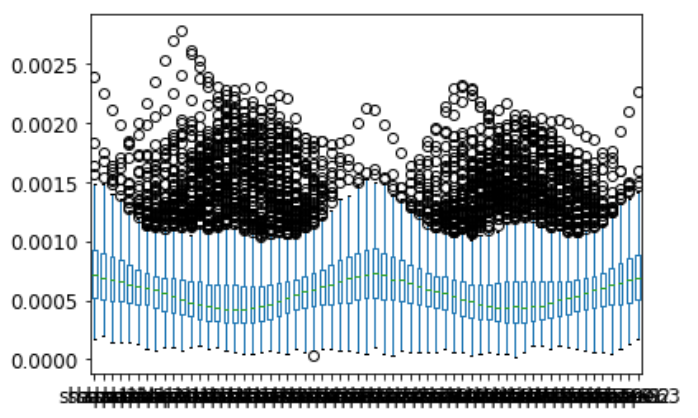


Figure 6-Groupe Shape

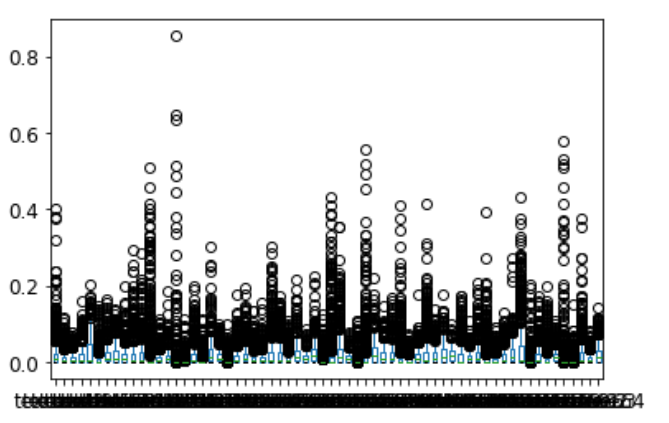


Figure 7-Groupe Texture

### Transformation des données

La première transformation sur les données qui a été effectué a été de convertir les espèces, qui contenaient des strings, en chiffre. Pour cela, la classe LabelEncoder de skelarn.

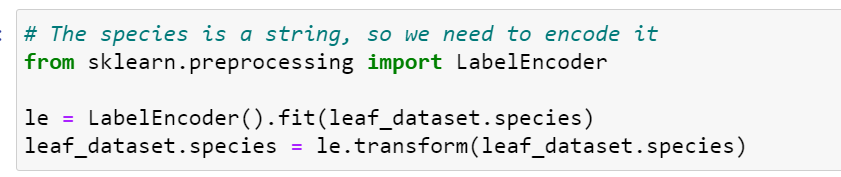


Figure 8- Encodage des espèces

La seconde transformation a été d’uniformiser la plage de valeurs des différentes caractéristiques. Plusieurs techniques s’offrent à nous. Dans notre cas, nous avons privilégié le StandardScaler, qui permet de centrer les données autour de 0 avec une déviation standard de 1. Cette modification est appliquée à chacune des caractéristiques de façon indépendante.

L’ensemble du pipeline de transformation se retrouve dans le fichier LeafClassificationPipeline.py qui permet d’automatiser la création des données.

Enfin, l’entraînement des différents algorithmes a été effectués avec la technique de cross-validation. Dans ce cas-ci, c’est la classe StratifiedKFold qui a été utilisé pour la séparation des données en fold. Et nous avons utilisés 5 fold. Le manque de données par rapport au nombre de classe ne permet pas d’augmenter facilement le nombre de fold, car on se retrouve rapidement avec un seul exemple de classe par fold.

Il est facile de modifier les paramètres de générations du dataset. Tous les paramètres sont dans le fichier yaml data\_cfg.yaml. Après avoir modifié un paramètre, exécuté make\_dataset.py pour générer le nouveau dataset.

# Algorithmes

L’entraînement et l’optimisation des algorithmes présenté dans le fichier 4. Training. Celui-ci repose sur les différentes classes de Classifier qui ont été créés pour faciliter la gestion des résultats et de l’entraînement.

Tous les algorithmes ont été raffiné avec un RandomSearchCV de manière itérative. Les paramètres utilisés dans le RandomSearchCV ont été présélectionné par rapport à leur impact potentiel sur les résultats. La plage de valeur des paramètres a été sélectionné itérativement pour permettre au RandomSearchCV de ne pas sélectionner une solution sur les limites des paramètres. Enfin, certains paramètres ont dû être limité à cause de limite computationnelle.

L’utilisation d’un RandomSearchCV permet de rapidement obtenir une solution viable pour chacun des algorithmes, ce n’est toutefois pas une solution parfaite. L’utilisation d’un algorithme génétique serait préférable pour raffiner les résultats, mais sklearn n’en contient pas de base. Des tests ont été effectué avec sklearn-genetic-opt mais le besoin computationnel était trop élevé pour obtenir des résultats convenables. Enfin, des tests ont étés fait avec la classe expérimentale HalvinRandomSearchCV. Toutefois, sont implémentation est problématiques. En effet, celle-ci s’arrête dès qu’on ensemble d’hyperparamètres n’est pas compatible ensemble (ex : certaines loss avec certains solver), alors que RandomSearchCV et GridSearchCV ne font qu’ignorer le résultat.

Le RandomSearchCV a utilisé le score Neg\_Log\_Loss pour l’optimisation car Log Loss est la métrique utilisée dans la compétition Kaggle.

## Logistic Regression

La régression logistique a été la plus simple à raffiné et c’est aussi la technique qui nous a donné les meilleurs résultats.

Nous avons testé l’ensemble des 5 solvers disponible, chacun avec les pénalités respectives :

* Newton-cg – l2
* Lbfgs – l2
* Libliinear – [l1, l2]
* Sag – l2
* Saga – [elasticnet, l1, l2]

Le coefficient de régularisation suivait la distribution loguniform entre 1e-5 et 1000.

La solution fournit par le RandomSearchCV est :

* C=390.02
* Penalty=L2
* Solver=lbfgs

## SVM

Les 4 noyaux disponibles ont été testé, chacun avec leurs paramètres respectifs :

* Linear
* Sigmoid
* Poly
* Rbf

Les différents noyaux donnent des résultats similaires, mais notre RandomSearchCV a sélectionné le noyau polynomial de degrés 1.

## Random Forest

Cet algorithme a été le plus compliqué à raffiner à cause du temps d’entraînement. En effet, le RandomSearchCV sélectionne toujours une solution qui se situe sur les limites du nombre d’estimateur, et du maximum de profondeur. Les meilleurs résultats obtenus sont donc limités par le temps de calcul et non par les données. Toutefois, le temps d’entraînement et d’inférence sont largement supérieurs aux autres algorithmes, il serait difficile d’en justifier son choix étant donné que d’autres algorithmes ont de meilleurs résultats beaucoup plus rapidement.

La meilleure solution obtenue a 1000 estimateurs, une profondeur maximale de 10.

## Multi-layer Perceptron

Cet algorithme a été le second plus complexe a entraîné. En effet, la grande liberté au niveau de l’architecture des couches pose problème pour faire une recherche d’hyperparamètres automatique. Nous nous sommes concentrés sur des architectures simples, à 1 ou 2 couches, ne contenant pas plus de 500 neurones chaque.

Nous avons ignoré le solver sgd car dans de nombreux cas, la solution était divergence et générait des valeurs infinies. En restreignant le taux d’apprentissage, nous obtenions des solutions, mais inférieures aux solutions fournit par les 2 autres solver.

## KNeighbors

Pour l’algorithme de classification KNeighbors, sa rapidité a facilité l’entraînement et a permis d’évaluer les paramètres suivants :

* Voisins
* Poids
* Algorithme
* Taille des feuilles
* P

Toutefois, très rapidement, le RandomSearchCV a fourni des résultats à l’intérieur des limites de la recherche. Des tests ont été effectué pour valider que ce n’était pas un minimum local.

## Decision Tree

Comme pour l’algorithme précédent, l’algorithme Decision Tree est a rapidement convergé à l’intérieur des limites du RandomSearchCV. Les résultats de cet algorithme semblent aberrants. Bien que le score utilisé pour évaluer l’algorithme s’améliore, les métriques de précision, justesse et de rappel diminuent avec l’entraînement.

# Résultats

Pour analyser les résultats, nous avons utilisés plusieurs métriques :

* Score f1
* Justesse
* Rappel
* Précision
* Log\_loss

## Résultat avec StandardScaller

Ces résultats ont été obtenu avec StandardScaller appliqué sur les données. Les noms avec “\_o” correspondent aux algorithmes entraînés. Tandis que les noms sans ce suffix, correspondent aux versions de base des algorithmes.

La figure ci-dessous montre les résultats sur le dataset d’entraînement. On remarque que c’est l’algorithme le plus « simple » qui semble fonctionner le mieux. Le MLP obtient aussi de très bonne performance, dans sa version entraînée et de base.

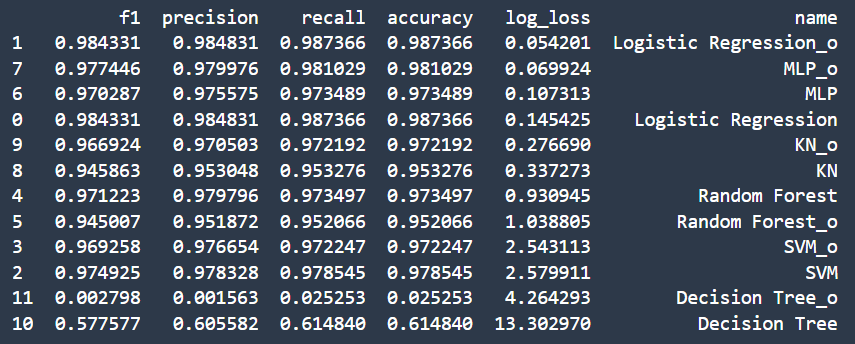


Figure 9- Résultats sur le dataset d'entrainement

L’ordre des résultats reste le même mais les résultats en eu même change légèrement sur le dataset de validation. On remarque quand même que la log\_loss est inférieur sur le dataset de test que sur le dataset d’entraînement. C’est qui est particulier. On suppose que des données aberrantes sont présentes dans le dataset d’entraînement, ce qui aurait pour effet de diminuer les résultats. On peut remarquer que les algorithmes généralisent bien, malgré le peu de données.

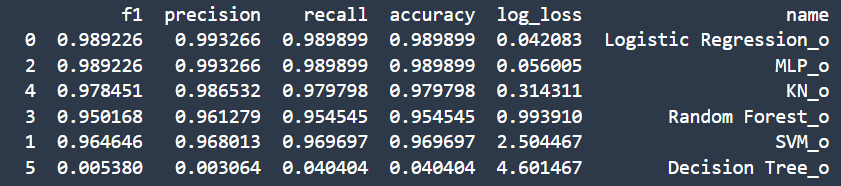


Figure 10- Résultats sur le dataset de test

## Résultat avec MinMaxScaller

Les résultats suivants ont été obtenus avec le MinMaxScaller. On remarque que les résultats sont inférieurs à ceux obtenus avec le StandardScaller. Cela laisse supposé que certaines données extrêmes perturbent l’entraînement, car la technique MinMax est très susceptibles aux données extrêmes.

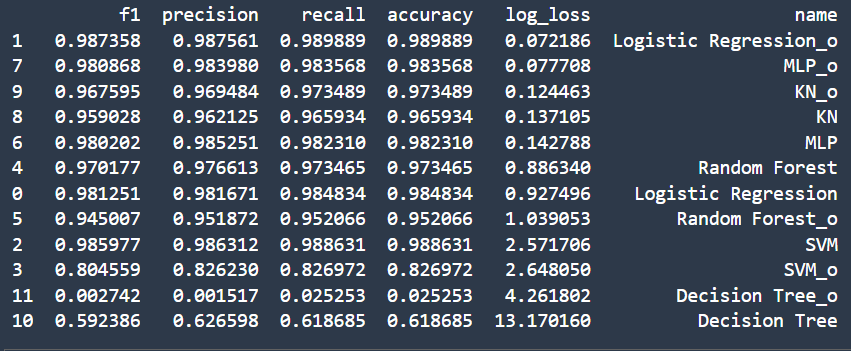


Figure 11- Résultats sur le dataset d'entraînement

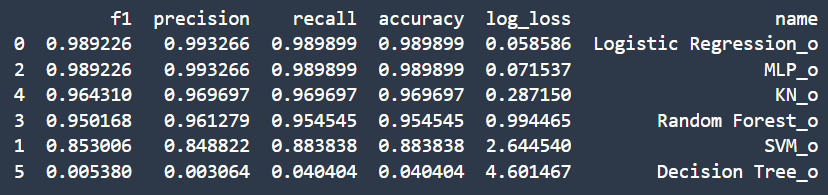


Figure 12- Résultats sur le dataset de test

## Résultat avec NormalisationScaller

## Comparaison par rapport à la compétition

Les meilleurs résultats obtiennent un des scores d’un facteur 10 meilleur à ce que nous obtenons. Toutefois, le score obtenu en soumettant nos algorithmes a toujours été supérieur à ceux que l’on a calculé localement. Cette différence provient en partie de séparation supplémentaire de données que l’on a fait pour travailler localement.

La majorité des meilleurs résultats utilisent des réseaux de neurones. Dans nos choix, nous avons limités le champ d’action du MLP pour des raisons computationnelles, ce qui limite ses performances. De plus, l’implémentation des MLP ne se fait pas avec sklearn, mais avec Keras majoritairement, qui est une librairie beaucoup plus adéquate pour implémenter des solutions à base de réseaux de neurones.

Les différents scaller montrent que la préparation des données à un impacte sur les résultats. Il serait intéressant de voir l’impacte de technique de suppression des données aberrantes sur nos résultats.