UNIVERSITÉ DE
SHERBROOKE

IFT712
Projet de session

Classification de feuilles

Étudiants
Josselin DUBOIS (dubj0701)
Antoine GUENARD (guea0702)
Yann JOURDIN (jouy0701)

14 Avril 2022

Table des matières

1 Présentation du projet

1.1 Mise en contexte

Ce travail a pour objectif de tester six méthodes différentes de classification supervisée sur un jeu de données. Ce travail s'effectue en s'appuyant sur la bibliothèque scikit-learn, qui met à disposition une collection de méthodes de classification.

À travers ce travail, on cherche à mettre en oeuvre une démarche scientifique nous permettant de rechercher et trouver les meilleurs paramètres pour nos modèles. Pour s'assurer du bien-fondé de nos estimations, on prendra soin d'effectuer des validations croisées sur nos résultats.

1.2 La base de données

On choisi de travailler sur la base de données Kaggle Leaf Classification (Classification de feuilles). Cette base de données présente l'avantage d'avoir déjà été nettoyée pour être utilisée.

Cette base de données a extrait 3 caractéristiques d'images de feuilles, à savoir la description de leur forme, un histogramme de leur texture, et un histogramme de leurs marges. Ces trois caractéristiques sont représentées par des vecteurs à 64 dimensions. On a donc un total de **192 caractéristiques**.

Le but est d'associer chaque feuille à l'une des **99 espèces**. On dispose pour cela de 16 exemples par catégorie, pour un total de **1 584 échantillons**.

La base de données est de plus séparée en deux jeux de données : un jeu d'entraînement supervisé de **990 échantillons** (10 de chaque espèce, voir figure 3) et un jeu de test non supervisé de **594 échantillons** (6 de chaque espèce).

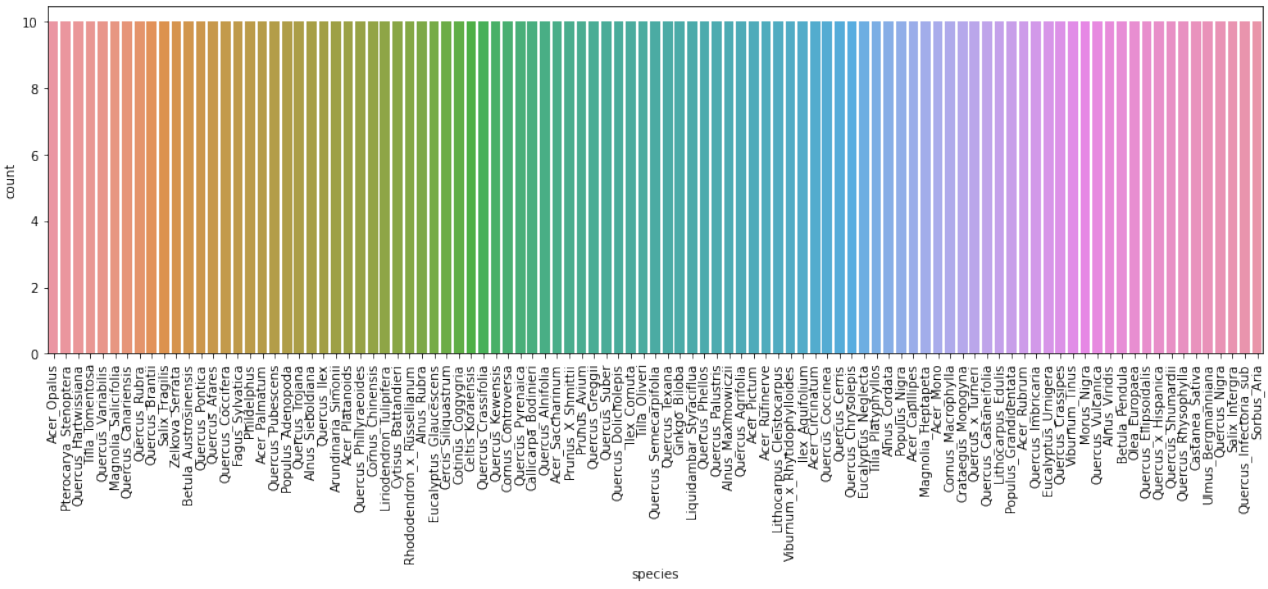

FIGURE 1 – *Répartition des espèces dans le jeu d'entraînement*

2 Travail effectué

2.1 Structure du projet

Notre projet s'organise naturellement autour d'un *Classifieur*. Ce dernier a besoin de pouvoir entraîner un modèle sur des données, puis de pouvoir prédire les espèces de nouvelles données, ainsi que d'évaluer les performances sur un jeu de données. Finalement, on demande au classifieur de pouvoir rechercher les meilleurs hyperparamètres pour nos données. On a ensuite des spécifications pour chacune des méthodes utilisées

Pour la recherche d'hyperparamètres (qui se fait sur une grille de paramètres), on a besoin d'un conteneur contenant nos paramètres et leurs domaines de recherche, permettant d'obtenir cette grille.

Finalement, on utilise un *Gestionnaire de données* pour pouvoir stocker nos jeux de données, et pour pouvoir séparer nos données en jeux d'entraînement et de validation, pour le bien des validations croisées que nous allons effectuer.

Notre architecture se résume ainsi sur le diagramme de classe de la figure 2.

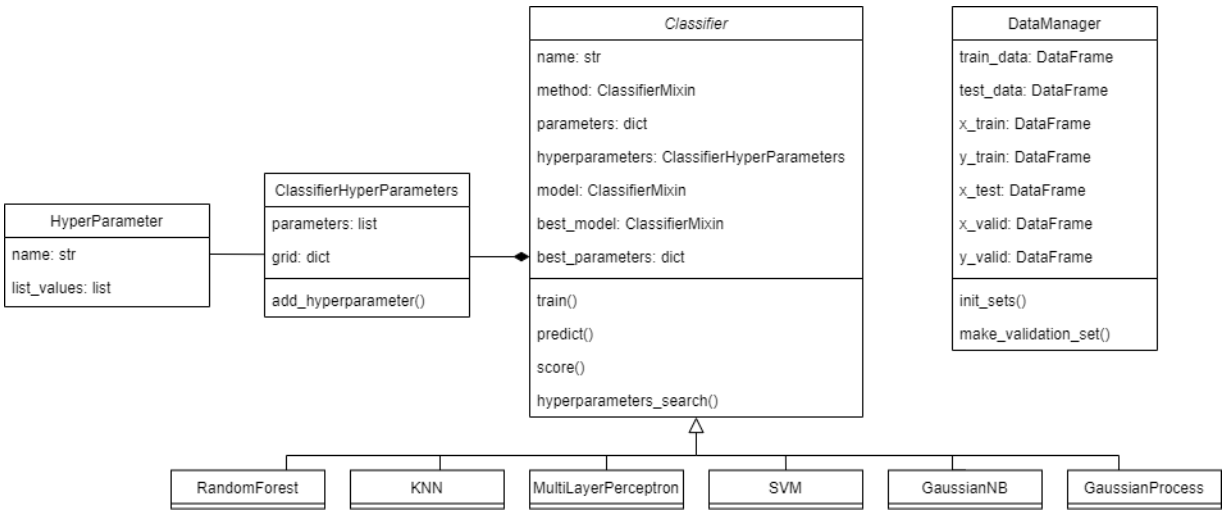

FIGURE 2 – *Diagramme de classes du projet*

2.2 Mesure des performances

On mesure la performance de nos modèles en mesurant leur précision, c'est à dire la proportion de données bien classées parmi les données prédites par le modèle. On n'affecte aucun poids particulier à aucune classe ou aucune donnée. On utilise pour cela la méthode de score *sklearn* des classifieurs.

2.3 Recherche d'hyperparamètres

Pour chaque modèle, on se doit de faire une recherche d'hyperparamètres. On utilise pour cela une grille d'hyperparamètres (le produit cartésien des valeurs à tester pour chaque hyperparamètre) avec laquelle on va entraîner le modèle pour chacune des combinaisons. On applique une validation croisée basée sur la précision (tout comme la mesure de la performance) pour valider la meilleure

combinaison. Tout cela se fait grâce à la classe GridSearchCV de *sklearn*.

Pour former la grille d'hyperparamètres, on choisit à la main les intervalles de valeurs pour chaque hyperparamètre.

3 Modèles utilisés

3.1 K Plus Proches Voisins (K Nearest Neighbors)

La première méthode qu'on a choisi d'utiliser est sûrement la plus classique : les K plus proches voisins. Cette méthode consiste à évaluer la classe d'une donnée en prenant la classe la plus représentée parmi ses k « plus proches » voisins pour une mesure choisie.

La méthode dispose de plusieurs hyperparamètres :

k	Le nombre de voisins à prendre en compte.
Algorithme (et ses paramètres propres)	L'algorithme utilisé pour la recherche des plus proches voisins (parmi BallTree, KDTree, et la force brute).
Métrique (et ses paramètres propres)	La métrique utilisée pour mesurer la distance entre deux points.
Poids	La mesure d'importance des voisins en fonction de leur distance au point à classer.

3.2 Perceptron Multi-Couches (Multi Layer Perceptron)

On implémente aussi un réseau neuronal simple, le perceptron multi-couches. Il consiste en plusieurs couches pleinement connectées de perceptrons, permettant de faire des transformations non linéaires de notre espace de points.

La méthode dispose de plusieurs hyperparamètres :

Nombre et taille des couches	Le nombre de couches cachées et nombre de neurones par couches.
Fonction d'activation	La fonction d'activation des perceptrons, parmi sigmoïde, tanh et ReLU.
Solveur (et ses paramètres propres)	Le solveur utilisé pour la rétropropagation, parmi lbfgs, sgd (et son taux d'apprentissage et moment) et adam (et ses taux de décomposition des vecteurs de moments) .

3.3 Forêt d'arbres décisionnels (Random Forest)

On utilise ensuite la forêt d'arbres décisionnels, une combinaison de modèles. Cette méthode consiste en la combinaison de plusieurs arbres décisionnels afin d'approcher au mieux la solution.

La méthode dispose de plusieurs hyperparamètres :

Nombre d'arbres	Le nombre d'arbres dans la forêt.
Critère de qualité	La fonction qui mesure la qualité d'une séparation, parmi l'impureté de Gini et l'entropie.

3.4 Machine à Vecteur de Support (Vector Support Machine)

Le prochain classifieur utilise les machines à vecteur de support avec différents noyaux, qui permettent de changer la représentation des points en les plongeant dans un espace dans lequel on espère qu'ils soient linéairement séparables (modulo une marge).

Encore une fois, le modèle dispose de différents hyperparamètres :

Noyau (et ses paramètres propres)	La fonction de noyau pour la régression, parmi linéaire, polynomial (et son degré), fonction de base radiale et sigmoïde.
Terme de pénalisation	Le paramètre de la régularisation ℓ_2 pour nos paramètres.
Forme de la fonction de décision	Parmi « one-vs-one » et « one-vs-rest ».

3.5 Classification naïve bayésienne (Gaussian Naive Bayes)

La classification naïve bayésienne effectue une classification en supposant des hypothèses très fortes concernant l'indépendance des caractéristiques de nos données. C'est un modèle très simple qui n'a donc pas d'hyperparamètres.

3.6 Processus Gaussiens (Gaussian Process)

Finalement, on implémente une classification par processus gaussiens, qui se base sur l'approximation de Laplace. Celle-ci n'a qu'un hyperparamètre notable.

Forme de la fonction de décision	Parmi « one-vs-one » et « one-vs-rest ».

4 Présentation des résultats

4.1 Performances des modèles

On examine les performances de nos modèles suite aux recherches d'hyperparamètres effectuées. On calcule une première fois le score (comme expliqué en section 2) sur le jeu utilisé pour entraîner le modèle (résultats affichés en bleu), puis sur un jeu de données de validation qui n'a pas été utilisé pour l'entraînement (résultats affichés en orange). Évidemment, les scores de validation sont inférieurs aux scores d'entraînement.

De manière étonnante, nos résultats sont exceptionnellement bons pour presque tous nos modèles (excepté la classification naïve bayésienne et les processus gaussiens), avec des performances supérieures à 90% sur les jeux de validation.

On en déduit ainsi qu'on n'a ni sur-apprentissage, ni sous-apprentissage sur ces 4 modèles. D'ailleurs, aucun de ces 4 modèles ne semble vraiment se démarquer.

La classification naïve bayésienne présente une forte tendance à sur-apprendre. En effet, elle ne fait aucune erreur sur le jeu d'entraînement mais a un score très faible (inférieur à 50%).

Enfin, les processus gaussiens présentent une forte tendance à sous-apprendre. En effet, les erreurs d'entraînement de de validation sont très élevées (un score inférieur à 50% dans les deux cas).

Les résultats de ces deux derniers modèles nous indiquent d'ailleurs que notre jeu de données ne se prête pas aux analyses et hypothèses gaussiennes.

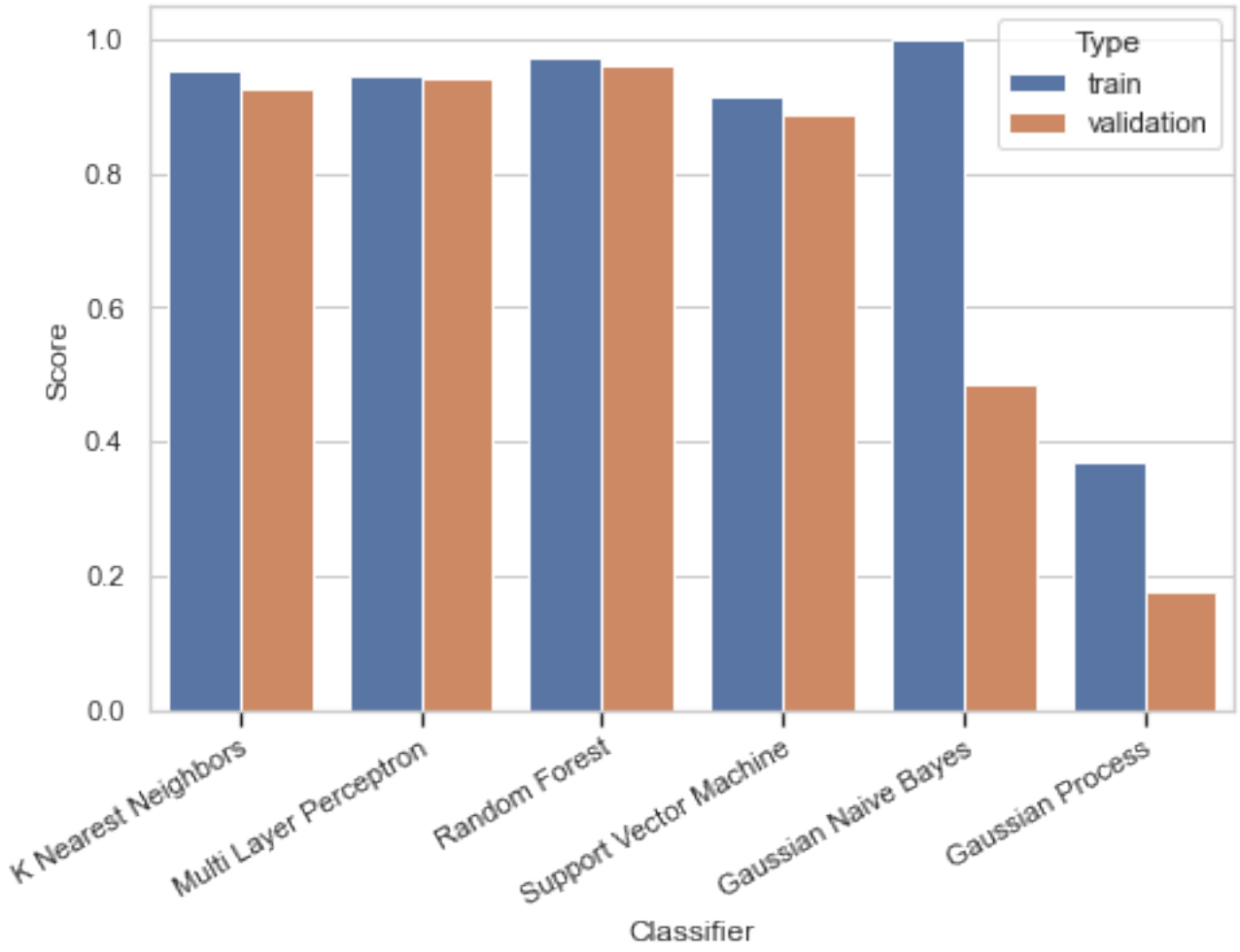

FIGURE 3 – *Performances des modèles*

4.2 Hyperparamètres conservés

On présente ici les grilles d'hyperparamètres choisies pour l'analyse, ainsi que les paramètres finaux validés par la recherche.

K plus proches voisins

Paramètre	Intervalle de valeurs	Valeur conservée
k	[1, 15]	1
Algorithme (et ses paramètres propres)	{ Ball Tree (nombre de feuilles : [10, 100]), KDTree, Brute Force }	Ball Tree (nombre de feuilles : 10)
Métrique (et ses paramètres propres)	{ $\|\cdot\|_1$, $\|\cdot\|_2$ }	$\|\cdot\|_1$
Poids	{ Uniforme, Inverse de la distance }	Uniforme

On peut noter que le k = 1 est très étonnant : il suffit de regarder le voisin dont on est le plus proche pour estimer au mieux sa classe.

Perceptron Multi Couches

Paramètre	Intervalle de valeurs	Valeur conservée
Nombre et taille des couches	1x100, 1x50, 2x10, 2x25	1x50
Fonction d'activation	{ Identité, Logistique, Tanh, ReLU }	Logistique
Solveur (et ses paramètres propres)	{ lbfgs (alpha : [10^{-6}, 10^{-1}]), sgd (taux d'apprentissage : { constant, adaptatif }, moment : [0, 1]), adam (β : [0.5, 1[^{2}) }	lbfgs (alpha = 10-3)

Forêt d'arbres décisionnels

Paramètre	Intervalle de valeurs	Valeur conservée
Nombre d'arbres	{ 10, 50, 100 }	100
Critère de qualité	{ Impureté de Gini, Entropie }	Impureté de Gini

Machines à vecteur de support

Paramètre	Intervalle de valeurs	Valeur conservée
Noyau (et ses paramètres propres)	{ Linéaire, Polynomial (degré : [3, 15]), rbf, Sigmoïde }	rbf
Terme de pénalisation	[0.1, 2]	2
Forme de la fonction de décision	{ one-vs-one, one-vs-rest }	one-vs-one

Processus gaussiens

Paramètre	Intervalle de valeurs	Valeur conservée
Forme de la fonction de décision	{ one-vs-one, one-vs-rest }	one-vs-rest

5 Conclusion

En résumé, on a pu implémenter 6 méthodes différentes pour la classification des feuilles, dont quatre se sont révélées efficaces. En revanche, deux autres méthodes étaient inadaptés à notre problème.

Ce projet nous a permis de constater que le choix de la méthode est une étape qui demande de bien s'adapter au jeu de données, et qui demande donc une grande analyse en elle-même. Certains modèles ne peuvent être adaptés pour un jeu de données, et les différents paramètres à choisir sont le résultat de longs tests.

Liste des figures