Rapport de projet

IFT 870 - Forage de données

Classification des feuilles

Hiver 2024

Étudiants

	Courriel	CIP
Taoufik ASSABAR	asst1001@usherbrooke.ca	asst1001
Nada BELLARI	beln1812@usherbrooke.ca	beln1812
Anne-Sophia LIM	lima1401@usherbrooke.ca	lima1401
Yanis PERRIN	pery2002@usherbrooke.ca	pery2002

Sommaire

Partie 1 - Données

1 Description du sujet, contexte général et problématique

Le développement de l'informatique ainsi que l'expansion rapide d'internet et des réseaux sociaux a provoqué une augmentation significative du nombre de données à disposition des entreprises. À tel point que la science des données est devenue un enjeu majeur au centre de leurs stratégies. Son apport leur permet de tirer des informations à partir des données recueillies et d'orienter la stratégie de l'entreprise afin de faciliter la prise de décision. Cependant, son application ne se limite pas au monde de l'entreprise. L'étude des données est aussi un enjeu majeur de la médecine, de la bioinformatique et peut même s'appliquer à des sujets en lien avec la biodiversité. En étudiant les données, des avancées majeures peuvent être apportées et ainsi aider ces secteurs à se développer.

C'est dans ce contexte que nous nous penchons aujourd'hui sur des techniques d'apprentissage qui nous permettent de créer des modèles qui peuvent généraliser des schémas à partir d'ensembles de données. Dans le cadre de notre projet IFT 870, nous nous pencherons sur une base de données provenant de Kaggle : la classification des feuilles d'arbres. La problématique du projet est ici la différenciation des espèces végétales. **La science des données et le machine learning peuvent-ils nous aider dans la tâche de différenciation des espèces végétales ?** L'objectif étant d'effectuer des prédictions à l'aide d'algorithmes de machine learning afin de prédire l'espèce de la plante. Les feuilles, de par leur volume, leur prévalence et leurs caractéristiques uniques, constituent un moyen efficace de différenciation des espèces végétales. C'est pour cela que nous effectuerons les prédictions à partir des caractéristiques recueillis des feuilles qui composent les plantes. Les prédictions se feront à l'aide de différentes méthodes de classifications, toutes mises en œuvre grâce à la bibliothèque bien établie scikit-learn.

2 Importance du sujet et motivations

On estime qu'il existe près d'un demi-million d'espèces de plantes dans le monde. La classification des espèces a toujours été problématique et entraîne souvent des identifications en double. La réponse à cette problématique permettra, à terme, de régler d'autres préoccupations touchant de près ou de loin à la classification des espèces végétales.

Par exemple, bien que les incendies de forêt au Canada et le projet de classification des feuilles sur Kaggle ne soient pas directement liés, il existe des domaines où ils pourraient se rejoindre. Les modèles développés pour classer les feuilles pourraient être utilisés dans des systèmes de surveillance des forêts pour détecter les signes de stress ou de maladie des arbres, ce qui pourrait potentiellement aider à identifier les zones à risque accru d'incendie et permettrait aux agriculteurs et aux spécialistes de la santé des plantes d'intervenir rapidement pour prévenir la propagation des maladies. De plus, une meilleure identification des espèces végétales dans les forêts pourrait contribuer à la gestion et à la conservation des écosystèmes forestiers, ce qui pourrait indirectement avoir un impact sur la prévention des incendies. Enfin, en analysant les caractéristiques des feuilles des plantes en relation avec leur rendement ou leur résilience aux conditions environnementales, le machine learning peut aider à optimiser les pratiques agricoles pour une production plus efficace et durable.

De nombreuses motivations entourent donc notre sujet, il serait peut être intéressant de jeter un œil à ce qu'il a déjà été fait pour tenter de répondre à notre problématique.

3 Historique des travaux et développements

Historiquement, de nombreux travaux en lien avec notre sujet ont déjà été effectués et nombreuses sont les approches explorées par les chercheurs.

[1] Étude par Keivani et al. (2020): Dans l'étude "Automated Analysis of Leaf Shape, Texture, and Color Features for Plant Classification", Mohammad Keivani, Jalil Mazloum, Ezatollah Sedaghatfar, et Mohammad Bagher Tavakoli ont exploré des méthodes avancées pour la classification automatique des plantes. Leur recherche s'est concentrée sur l'analyse automatisée des caractéristiques telles que la forme, la texture et la couleur des feuilles pour identifier les espèces végétales. Ils ont utilisé des techniques de traitement d'image pour extraire ces caractéristiques et ont appliqué des méthodes de machine learning pour classer les feuilles comme KNN, SVM ou encore les arbres de décisions (explication détaillé de ces techniques dans le pdf *explication.pdf*). Cette approche multidimensionnelle a permis d'améliorer la précision de la classification des plantes en exploitant divers aspects morphologiques des feuilles. Cette approche est la plus similaire à celle que nous aborderons dans notre projet et aussi la plus en lien avec le cours d'IFT 870 (algorithmes que nous avons eu l'occasion d'étudier et méthode efficace pour un ensemble de jeu de données réduit).

[2] Étude par Cope et al. (2012): L'article "Plant species identification using digital morphometrics: A review" par James S. Cope et al. se penche sur l'identification des espèces végétales en utilisant des morphométries numériques. Cette recherche a exploré l'utilisation de descripteurs de forme, de texture et de couleur pour classifier les feuilles d'arbres. Les techniques employées incluent l'analyse en composantes principales (PCA) pour réduire la dimensionnalité des données et améliorer la classification. Leur travail a mis en avant l'efficacité des méthodes de vision par ordinateur combinées à des techniques statistiques pour la classification des feuilles.

[3] Étude par Kumar et al. (2012): Dans l'étude "Leafsnap: A Computer Vision System for Automatic Plant Species Identification", Neeraj Kumar et al. ont développé Leafsnap, un système de vision par ordinateur qui identifie automatiquement les espèces végétales à partir de photos de leurs feuilles. Leafsnap utilise un algorithme de correspondance de caractéristiques basé sur les veines des feuilles et la forme générale pour réaliser la classification. Cette approche est innovante car elle combine la photographie numérique et un traitement d'image sophistiqué pour fournir une identification précise et automatisée des espèces d'arbres.

4 Description des données

Avant de se lancer dans la mise en place de nos techniques d'apprentissage, il est crucial de comprendre le jeu de données que nous manipulons.
Celui-ci est disponible à l'adresse suivante : www.kaggle.com/c/leaf-classification.

Le jeu de données présente de nombreuses caractéristiques associées aux feuilles telle que leur forme, leur marge ou encore leur texture associé au type d'espèce de la plante (Acer Opalus, Pterocarya Stenoptera…). Ces différentes caractéristiques peuvent être utilisées pour différencier les espèces de plantes. Notre objectif ici est donc de classifier les espèces de plantes en fonction des caractéristiques rencontrées, afin de prédire à partir des caractéristiques de la feuille l'appartenance à une espèce, notre donnée cible est donc l'espèce de la plante, on parle d'apprentissage supervisé (la cible est connue).

Le jeu de données est sous format csv et présente à l'origine 990 observations pour 193 attributs. Nous dénombrons 192 attributs numériques et 1 attribut de type objet (la cible). La donnée cible, l'espèce, est ici décliné en 99 classes distinctes qui comportent chacune 10 observations *(cf Figure 1)*. Le jeu de données est donc bien balancé (pas de classe plus présente que d'autres).

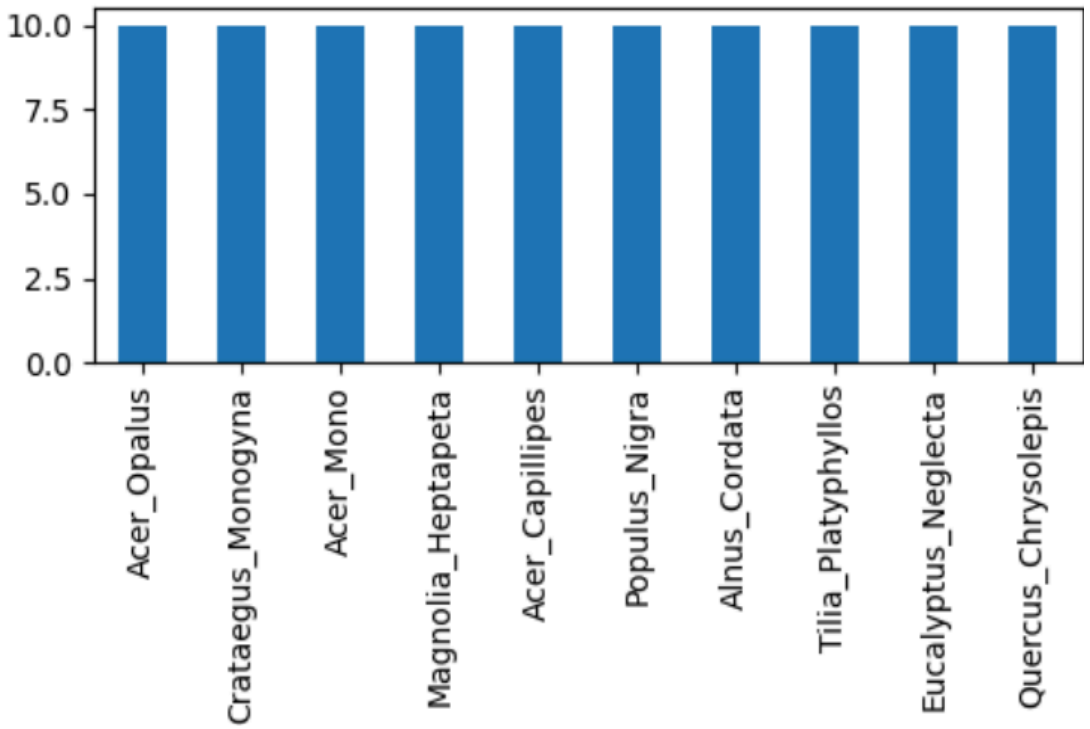

Figure 1 : Exemple du nombre d'observations par classe de la variable espèce (voir notebook pour voir l'entièreté des 99 classes)

	id	species	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	...	texture55	texture56	texture57	texture58
0	1	Acer_Opalus	0.007812	0.023438	0.023438	0.003906	0.011719	0.009766	0.027344	0.0	...	0.007812	0.000000	0.002930	0.002930
1	2	Pterocarya_Stenoptera	0.005859	0.000000	0.031250	0.015625	0.025391	0.001953	0.019531	0.0	...	0.000977	0.000000	0.000000	0.000977
2	3	Quercus_Hartwissiana	0.005859	0.009766	0.019531	0.007812	0.003906	0.005859	0.068359	0.0	...	0.154300	0.000000	0.005859	0.000977
3	5	Tilia_Tomentosa	0.000000	0.003906	0.023438	0.005859	0.021484	0.019531	0.023438	0.0	...	0.000000	0.000977	0.000000	0.000000
4	6	Quercus_Variabilis	0.005859	0.003906	0.048828	0.009766	0.013672	0.015625	0.005859	0.0	...	0.096680	0.000000	0.021484	0.000000
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
985	1575	Magnolia_Salicifolia	0.060547	0.119140	0.007812	0.003906	0.000000	0.148440	0.017578	0.0	...	0.242190	0.000000	0.034180	0.000000
986	1578	Acer_Pictum	0.001953	0.003906	0.021484	0.107420	0.001953	0.000000	0.000000	0.0	...	0.170900	0.000000	0.018555	0.000000
987	1581	Alnus_Maximowiczii	0.001953	0.003906	0.000000	0.021484	0.078125	0.003906	0.007812	0.0	...	0.004883	0.000977	0.004883	0.027344
988	1582	Quercus_Rubra	0.000000	0.000000	0.046875	0.056641	0.009766	0.000000	0.000000	0.0	...	0.083008	0.030273	0.000977	0.002930
989	1584	Quercus_Afares	0.023438	0.019531	0.031250	0.015625	0.005859	0.019531	0.035156	0.0	...	0.000000	0.000000	0.002930	0.000000

990 rows × 194 columns

Figure 2 : Aperçu des données à l'origine

En ce qui concerne notre analyse plus poussée du jeu de données (analyse des valeurs manquantes, données aberrantes, données dupliquées, statistiques descriptives…etc) celle-ci est présente dans la partie 2.

5 Quelques notions importantes

Avant de continuer et de passer au prétraitement de nos données, il serait intéressant que nous définissions quelques notions de l'apprentissage automatique qui pourront servir à mieux comprendre nos méthodes et analyses.

Le sur-apprentissage:

Le sur-apprentissage survient lorsqu'un modèle d'apprentissage automatique performe très bien sur les données d'entraînement mais à des performances basses sur l'ensemble de test. Cela est dû au fait que le modèle apprend non seulement les modèles généraux des données, mais aussi des bruits ou des détails spécifiques des données d'entraînement qui ne se généralisent pas bien à de nouvelles données. En d'autres termes, si l'on compare ce problème à la vie réelle, c'est comme si pour un examen de mathématiques vous avez appris à résoudre les exercices particuliers que le professeur vous a montrés, mais que vous n'avez pas vraiment compris les principes sous-jacents qui pourraient être appliqués à de nouveaux problèmes. Vous avez trop ajusté votre compréhension en fonction des exemples spécifiques que vous avez vus, cela pourrait être difficile si vous êtes face à des exercices que vous n'avez jamais vus.

La régularisation :

Les termes de régularisation sont justement là pour empêcher les modèles de sur-apprendre sur les données. Sans entrer dans les détails, plus le paramètre de régularisation du modèle est élevé, plus le modèle aura tendance à moins sûr apprendre. D'autres techniques sont utilisées pour éviter le sur-apprentissage. Parmi eux figure la validation croisée que nous allons expliquer dans le projet.

6 Organisation du projet

6.1 Arborescence et diagramme de classe

Avant d'entamer l'analyse plus approfondie du jeu de données, voici comment est organisé le projet :

```
|   explication.pdf
|   ProjetIFT870_H2024_v1.pdf
|   requirements.txt                    <-- bibliothèque nécessaire à la visualisation du projet
|
├───data                                <-- données brutes
|       test.csv
|       train.csv
|
└───src                                 <-- code source du projet
    |   projet_ift870.ipynb             <-- notebook associé au projet
    |
    ├───analyzer                        <-- script pour visualiser et analyser les données/résultats
    |       Analyzer.py
    |
    ├───classifier                      <-- ensemble des modèles
    |       Classifier.py
    |       DecisionTreeModel.py
    |       KNNModel.py
    |       LogisticRegressionModel.py
    |       MLPModel.py
    |       RandomForestModel.py
    |       SVMModel.py
    |
    └───data_manager                    <-- script pour gérer le chargement ainsi que le prétraitement des données
            Loader.py
            Preprocessor.py
```

Figure 3 : Diagramme des classes du projet

L'utilisation de l'ensemble des classes est présente dans le notebook du projet.

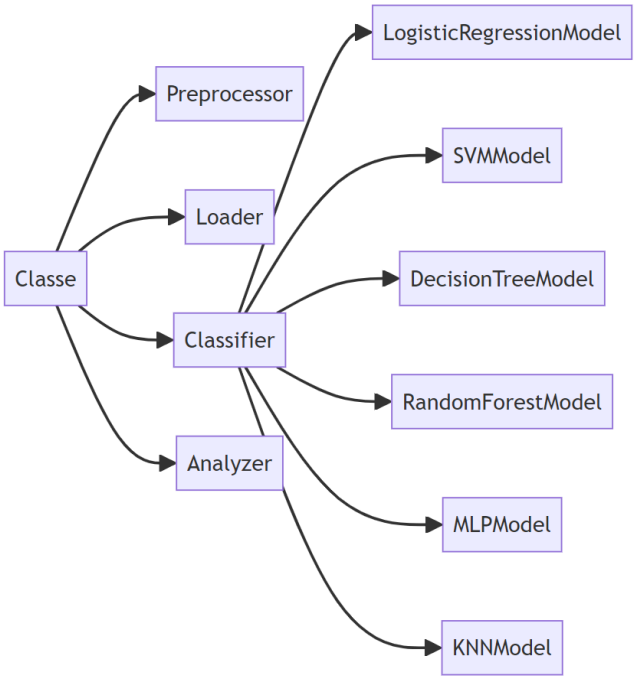

Figure 4 : Diagramme des classes du projet

Partie 2 - Algorithmes

L'utilisation de modèles de machine learning implique la décomposition du jeu de données en un ensemble d'apprentissage et un ensemble de test. Cette décomposition nous permet d'entraîner le modèle sur l'ensemble d'apprentissage, puis de tester le modèle sur un ensemble de test composé de données jamais vu par le modèle. Cela nous permet de tester la compétence du modèle à bien généraliser et à tester son aptitude à ne pas sur apprendre sur les données d'apprentissage.

1 Importation du jeu de données

Pour importer les données d'entraînement ("train.csv") et de test ("test.csv"), nous créons une classe Loader() qui présente 3 méthodes principales : load(), get_trainset() et get_testset().

load() : Cette méthode permet de charger les données à partir du chemin spécifié.
get_trainset() : Cette méthode nous retourne les données d'apprentissage.
get_testset() : Cette méthode nous retourne les données de test.

Ici, les données dites de test représentent un ensemble réservé à la soumission des résultats sur Kaggle. Notre objectif n'étant pas de faire des soumissions à partir du jeu de test dans Kaggle, notre projet se concentre majoritairement sur les données d'apprentissage ("train.csv") que nous décomposons par la suite en ensemble d'apprentissage et de test.

2 Analyse et prétraitement

Une fois le jeu de données importé à l'aide de la classe Loader(), nous pouvons à présent passer à une analyse plus poussée de notre jeu de données. Il est impératif de comprendre ce que nous manipulons. Pour ce faire, nous créons une classe Analyzer() qui nous suivra tout au long du projet afin d'analyser dans un premier temps les données mais aussi dans un second temps nos résultats. Cette classe présente 19 fonctions distinctes que nous présenterons au fil du rapport.

2.1 Données aberrantes

Dans un premier temps, il est important de savoir s' il présente des données manquantes, dupliquées et/ou aberrantes dans notre jeu de données. L'appel aux fonctions number_na() et number_duplicated() d'Analyzer() nous indique qu'aucune donnée manquante/dupliquée n'est observée dans le jeu de donnée (même en supprimant la colonne "id" qui aurait pu apporter la distinction des observations et donc peut être caché des données dupliquées). Cette information semble à première vue positive, cependant poussons l'analyse du jeu de données à l'aide de la fonction statistics().

	id	margin1	margin2	margin3	margin4	margin5	margin6	margin7	margin8	margin9	...	texture55	texture56
count	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	990.000000	...	990.000000	990.000000
mean	799.595960	0.017412	0.028539	0.031988	0.023280	0.014264	0.038579	0.019202	0.001083	0.007167	...	0.036501	0.005024
std	452.477568	0.019739	0.038855	0.025847	0.028411	0.018390	0.052030	0.017511	0.002743	0.008933	...	0.063403	0.019321
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	...	0.000000	0.000000
25%	415.250000	0.001953	0.001953	0.013672	0.005859	0.001953	0.000000	0.005859	0.000000	0.001953	...	0.000000	0.000000
50%	802.500000	0.009766	0.011719	0.025391	0.013672	0.007812	0.015625	0.015625	0.000000	0.005859	...	0.004883	0.000000
75%	1195.500000	0.025391	0.041016	0.044922	0.029297	0.017578	0.056153	0.029297	0.000000	0.007812	...	0.043701	0.000000
max	1584.000000	0.087891	0.205080	0.156250	0.169920	0.111330	0.310550	0.091797	0.031250	0.076172	...	0.429690	0.202150

8 rows × 193 columns

Figure 5 : Statistiques avancée du jeu données

Si l'on observe attentivement les données maximales observées par rapport à la donnée moyenne de chaque caractéristique, on remarque que certaines données semble très loin de la données moyenne, ce qui pourrait nous indiquer la présence de données aberrantes, c'est à dire une observation qui est significativement différente des autres observations dans un ensemble de données. Cependant, il serait préférable d'appuyer notre hypothèse via une analyse visuelle qui complétera notre simple analyse statistique. Pour cela, nous faisons appel à la fonction boxplot() et histogram() d'Analyzer() :

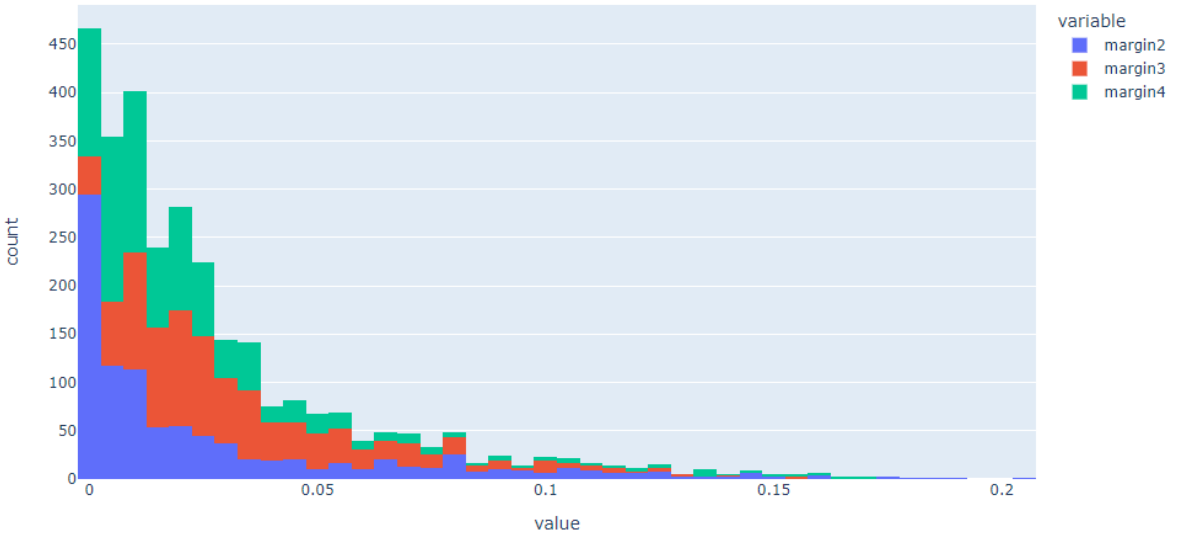

Figure 6 : Histogramme du comptage des différentes valeurs des variables "margin2", "margin3" et "margin4".

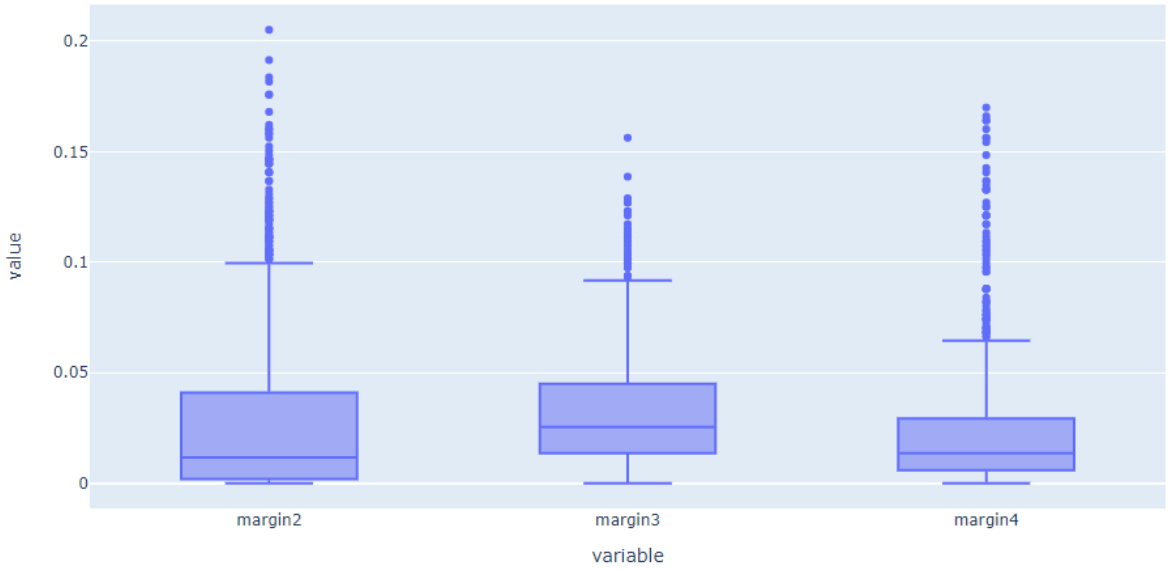

Figure 7 : Boxplot des différentes valeurs des variables “margin2”, “margin3” et “margin4”.

Si on observe attentivement l’histogramme et le boxplot ci-dessus, on remarque que certaines valeurs sont en effet bien loin de l’ensemble des valeurs observées pour les différentes caractéristiques, notre hypothèse est donc vérifiée. La présence de données aberrante n’est pas une bonne nouvelle car elles ont le potentiel de fausser les résultats d'une analyse statistique ou d'un modèle, car elles peuvent influencer les mesures centrales (moyenne, médiane) et les estimations de dispersion (écart-type, plage interquartile). C’est donc pour cela que nous passons à l’étape de remplacement des données aberrantes.

2.2 Écart interquartile et KNN imputer

Afin de supprimer les données aberrantes, nous créons une classe Preprocessor(), cette classe présente 10 fonctions qui nous aideront à nettoyer le jeu de données. La fonction replace_outliers_na() est une fonction que nous avons créée afin de remplacer les valeurs aberrantes par des données np.nan. Ce remplacement nous permettra par la suite d’appliquer un algorithme d’imputation de données manquantes par l’algorithme des plus proches voisins.

La fonction replace_outliers_na() utilise l’écart inter-quartile afin de définir une donnée aberrante pour ensuite la remplacer par np.nan. Ainsi, une donnée est dite aberrante si sa valeur ne se trouve pas dans l’intervalle suivant :

[Q1 - threshold * IQR, Q3 + threshold * IQR]

où Q1 et Q3 représentent respectivement le 1er et 3e quartile, IQR l’écart interquartile (Q3-Q1) et threshold un seuil spécifié, dans notre cas il est conseillé d’initialiser le seuil à 1.5.

Une fois la donnée aberrante remplacée, il pourrait être intéressant de regarder le pourcentage de données manquantes par caractéristiques.

```
Percentage of nan value by column :

species        0.000000
margin1        4.141414
margin2        7.878788
margin3        4.141414
margin4        7.474747
                ...
texture60    100.000000
texture61    100.000000
texture62     10.404040
texture63      7.676768
texture64      4.343434
```

Figure 8 : Pourcentage de données manquantes par caractéristiques

On remarque que certaines caractéristiques présentent énormément de données manquantes, certaines caractéristiques présentaient uniquement la valeur 0 et donc une variance de 0, ce qui ne nous intéresse absolument pas pour la suite de notre projet. Nous décidons donc de supprimer du jeu de données les caractéristiques qui présentent plus de 50% de données manquantes à l'aide de la fonction drop_column_na().

En ce qui concerne les données manquantes restantes, nous faisons appel à la fonction knn_imputer() qui utilise la méthode KNNImputer de la bibliothèque scikit-learn afin de remplacer les données manquantes par des données déterminées via l'algorithme des plus proches voisins. Cet algorithme fonctionne en trouvant les k voisins les plus proches d'un point donné dans un espace multidimensionnel, et en utilisant ces voisins pour effectuer une prédiction (ici la valeur manquante).

```
Percentage of nan value by column after the replacement :

species      0.0
margin1      0.0
margin2      0.0
margin3      0.0
margin4      0.0
            ...
texture58    0.0
texture59    0.0
texture62    0.0
texture63    0.0
texture64    0.0
```

Figure 8 : Pourcentage de données manquantes après remplacement

On n'oubliera pas d'encoder la variable "species" qui représente notre variable cible car celle-ci est uniquement composée de données textuelles. La fonction encoding() d'Analyzer() utilise la méthode LabelEncoder() de scikit-learn afin d'associer une valeur numérique propre à chacune des variables. Cela nous permettra ainsi de pouvoir appliquer nos algorithmes de machine learning car ils ne prennent pas en entrée de données textuelles.

2.3 Mélange et séparation des données

La dernière étape de notre prétraitement consiste à mélanger les données afin d'éviter les biais liés à l'ordre des données via la méthode shuffle() de la classe preprocessor. Nous séparons les données en base d'apprentissage et de test à l'aide de la fonction split_dataset() de Preprocessor() qui sépare la variable cible des autres variables à l'aide de la méthode train_test_split() de scikit-learn. Nous aurons donc deux bases de données contenant chacune l'ensemble des caractéristiques d'un côté et la variable cible de l'autre. Nous priorisons 70% des données à la base d'apprentissage afin de laisser la majorité des données et donc avoir le maximum de cas appris par le modèle, nous laissons 30% à la base de test afin de tester le modèle et mesurer sa précision sur des données jamais vues, ce qui conclut notre prétraitement des données.

3 Algorithme grid search et validation croisée

Une fois les données prétraitées, nous disposons d'un ensemble de données d'apprentissage et un ensemble de données de test. En ce qui concerne l'ensemble des données de test, nous n'y touchons plus, celui-ci est réservé seulement pour tester les modèles sur des données jamais vues. À présent, nous utiliserons l'ensemble des données d'apprentissage afin de réaliser la validation croisée et l'algorithme grid search.

3.1 Grid Search

L'algorithme de Grid Search est un algorithme qui est utilisé pour trouver les meilleures hyper paramètres d'un modèle. Comme vue en cours, les hyperparamètres sont des paramètres qui ne sont pas appris directement à partir des données lors de l'entraînement du modèle, mais qui influent sur le processus d'apprentissage. Il serait donc intéressant de chercher les hyper paramètres qui maximisent la précision de nos modèles. C'est pour cela que nous faisons appel à la méthode GridSearchCV() de la bibliothèque scikit-learn. Cette méthode intègre une validation croisée (nous verrons par la suite en détaille ce qu'elle fait) qui entraîne le modèle avec les différents paramètres spécifiés dans la grille de paramètres puis les teste sur un ensemble de validation. L'algorithme retourne ensuite la combinaison d'hyper paramètres qui maximise les performances du modèle. Dans notre cas, nous appliquons l'algorithme de grid search sur l'ensemble d'apprentissage en fonction de la grille d'hyper paramètres spécifiée dans chacune des classes du classifieur. Ainsi, chaque classifieur se verra attribuer la meilleure combinaison d'hyper paramètres.

3.2 Validation croisée

La validation croisée est un point central de notre projet. C'est une méthode statistique qui permet de minimiser les risques de sur-apprentissage ou de biais liés à la division des données. L'objectif de la validation croisée est d'obtenir une estimation fiable des performances d'un modèle sur des données non vues.

Son fonctionnement suit les étapes suivantes :

- Diviser les données d'apprentissage en k-1 sous-ensembles d'apprentissage et 1 sous-ensemble de validation.

- Entraîner le modèle sur les k-1 sous-ensembles d'apprentissage.
- Tester le modèle sur l'ensemble de validation.
- Répéter le cycle k fois en définissant un nouveau sous-ensemble d'apprentissage et de validation différents des fois précédentes

Le schéma ci-dessous explique bien son fonctionnement :

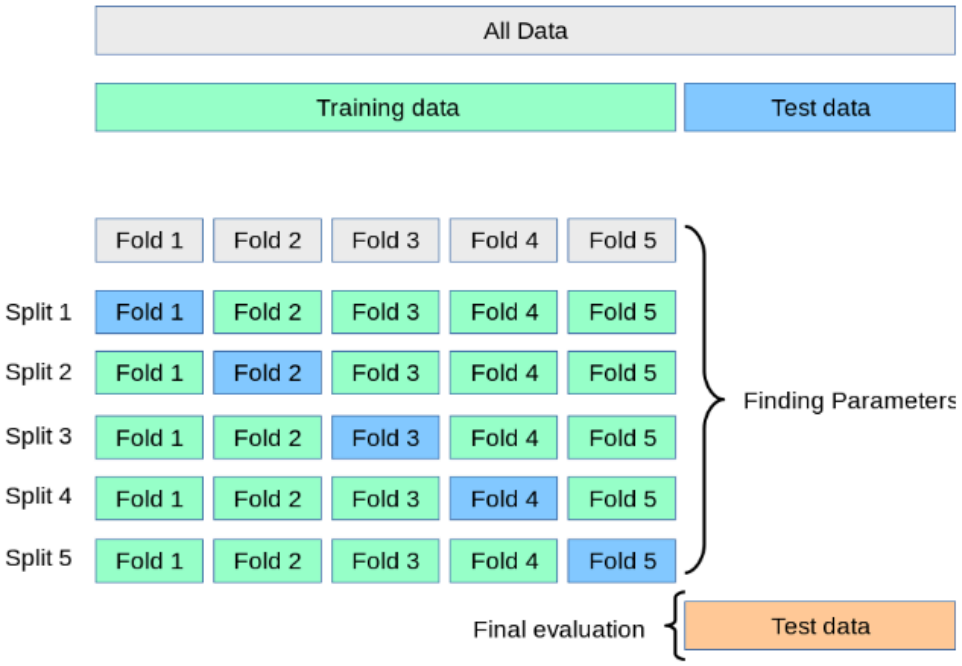

Figure 9 : Fonctionnement de la validation croisée

Ainsi le modèle est d'abord entraîné sur les sous-ensembles d'apprentissage puis tester sur un ensemble de validation contenant des données encore jamais vues par le modèle. Cela permet d'éviter le sur-apprentissage et d'améliorer la généralisation du modèle. Une fois le modèle entraîné et validé, nous pouvons le tester sur l'ensemble des données de test encore jamais vues. L'utilisation de la validation croisée est cruciale, en particulier lorsque les ensembles de données sont limités, ce qui est notre cas avec ce jeu de données.

La classe Classifier() que nous créons nous permet à l'aide des fonctions train(), test() et grid_search() d'effectuer l'apprentissage, le test ainsi que de lancer l'algorithme de grid search via la fonction GridSearchCv() de scikit-learn. En ce qui concerne la validation croisée, nous créons à l'aide de la méthode KFold() de scikit-learn des indices de train et de validation qui nous permettent de séparer l'ensemble d'apprentissage en k-1 sous-ensembles d'apprentissage et un sous-ensemble de validation. Nous pouvons ainsi utiliser les fonctions de la classe Classifier() pour effectuer l'apprentissage et la validation puis le test du modèle.

3.2.1 Standardisation des données

Pour chaque sous-ensemble d'apprentissage, il est intéressant de standardiser les données, c'est-à-dire de centrer et de réduire les données, en les transformant de telle sorte qu'elles aient une moyenne nulle et un écart-type unitaire. Pour cela, nous appelons la fonction standardization() de la classe Preprocessor() sur chaque sous-ensemble d'apprentissage de la validation croisée. Cette fonction utilise la méthode StandardScaler() de scikit-learn. La standardisation permet de stabiliser les calculs numériques, en particulier dans le contexte de l'optimisation des modèles et ainsi améliorer leur performance.

Maintenant que nous avons vu l'algorithme de grid search ainsi que la validation croisée, nous pouvons étudier les différents modèles que nous utiliserons dans ce projet. Chaque modèle que nous verrons hérite de la classe Classifier() et donc de toutes ses fonctions associées.

4 Les 6 modèles retenus

Les 6 modèles que nous avons retenus afin d'effectuer les différentes prédictions sont les suivants :

1. La Régression Logistique
2. Machine à vecteur de support
3. Arbre de décision
4. Forêt aléatoire
5. Perceptron multicouche
6. Classification par algorithme des K plus proches voisins

Nous avons fait le choix de ces modèles car ce sont tous des modèles que nous avons pu étudier au cours de nos études. Il sont pour la plupart très performant, parmi eux figurent les arbres de décisions ainsi que la classification par algorithme des K plus proches voisins, des modèles que nous avons eu l'occasion de voir au cours des différents ateliers, c'est aussi une raison pour laquelle nous les avons choisi même si d'autres algorithmes plus performants existent. Ici le but n'est pas de détailler le fonctionnement de chacune des méthodes, si cela vous tente nous vous laissons jeter à œil à l'explication de chaque méthode que nous avons pu mettre dans le fichier explication.pdf. Nous y détaillons les grandes parties ainsi que les choix d'hyperparamètres à tester pour l'algorithme grid search.

5 Analyse des résultats et démarche scientifique

Une fois l'entraînement des modèles terminé (validation croisée + grid search), nous récupérons les différents scores associés aux modèles. Pour une meilleure visualisation, nous avons intégré des fonctions à la classe Analyzer() qui nous permettront de mieux visualiser les différents résultats.

5.1 Score validation croisée

Nous faisons appel à la méthode plot_scores() qui nous permet d'analyser les scores d'apprentissage, de validation et de test de nos modèles à travers les 5 divisions effectuées lors de la validation croisée. Ici le score est la précision du modèle (accuracy_score).

5.1.1 Score d'apprentissage, validation et de test

	Classifier	Mean Train Score	Mean Validation Score	Mean Test Score	Mean F1 Score
0	LogisticRegressionModel	1.0	0.977364	0.971865	0.970684
1	SVMModel	1.0	0.971337	0.977370	0.976547
2	DecisionTreeModel	1.0	0.456869	0.519266	0.496381
3	RandomForestModel	1.0	0.909455	0.939450	0.937172
4	MLPModel	1.0	0.945717	0.958410	0.956866
5	KNNModel	1.0	0.948701	0.968807	0.968143

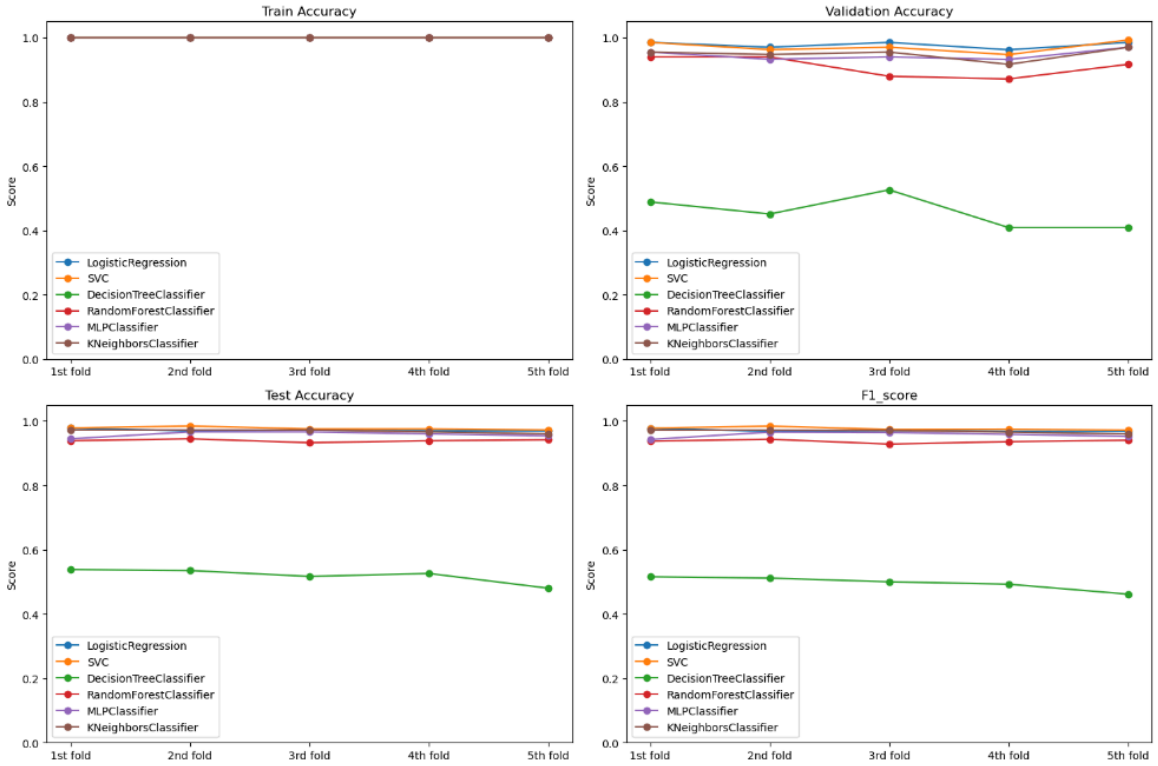

Figure 10 : Score des modèles

L'analyse du score d'apprentissage est claire : tous les modèles sont très efficaces voire trop efficaces. Malgré l'utilisation de la validation croisée, de l'algorithme grid search ainsi que l'ensemble du prétraitement effectué, les modèles restent très complexes voir trop complexes pour le jeu de données que nous disposons. Cependant, si l'on regarde attentivement les autres courbes, on se rend compte que nos modèles généralisent assez bien puisque qu'ils ont de très bons résultats sur l'ensemble de validation ainsi que de test (96.26% d'accuracy en moyenne sur l'ensemble de test en dehors des arbres de décision). Le seul modèle qui a du mal est l'arbre de décision, cela s'explique par le fait que c'est un modèle qui a beaucoup de mal à généraliser et mène très souvent au surapprentissage. Lorsque celui-ci doit effectuer des prédictions sur des données jamais vues, son fonctionnement le mène à ne pas pouvoir prédire efficacement, nous en reparlerons lors de l'analyse des courbes de validations.

5.1.2 Score F1

Une autre analyse intéressante que nous pouvons faire est sur le score F1 de chaque modèle. Le score F1 est une métrique que nous avons eu l'occasion de voir pendant le cours, celle-ci combine les concepts de précision et de rappel pour fournir une évaluation globale de la performance d'un modèle de classification. La précision mesure la proportion de prédictions positives qui sont correctes tandis que le rappel mesure la proportion de véritables positifs qui ont été correctement identifiés. Toutes les deux sont calculées comme suit :

$$\text{Précision} = \frac{\text{Vraies Positives}}{\text{Vraies Positives} + \text{Faux Positifs}}$$

$$\text{Rappel} = \frac{\text{Vraies Positives}}{\text{Vraies Positives} + \text{Faux Négatifs}}$$

$$F1 = 2 \times \frac{\text{Précision} \times \text{Rappel}}{\text{Précision} + \text{Rappel}}$$

Figure 11 : Formule de la précision, du recall et du score f1

Celui-ci peut être faussé surtout lorsque l'on dispose de données débalancées, ce qui n'est pas notre cas. Si l'on regarde attentivement la courbe, les scores f1 sur l'ensemble de test sont aussi très bons. Cela veut dire que nos modèles de classification ont une performance élevée en termes d'équilibre entre la précision et le rappel, c'est-à-dire qu'ils ont réussi à bien gérer à la fois les vrais positifs (instances correctement prédites comme positives) et les faux négatifs (instances réellement positives qui ont été manquées). Bien sûr le parallèle entre le score f1 et les scores de validation et de test peut être fait vu leur grande ressemblance, à noter que le modèle d'arbre décision a encore une fois du mal.

5.1.3 Score moyen

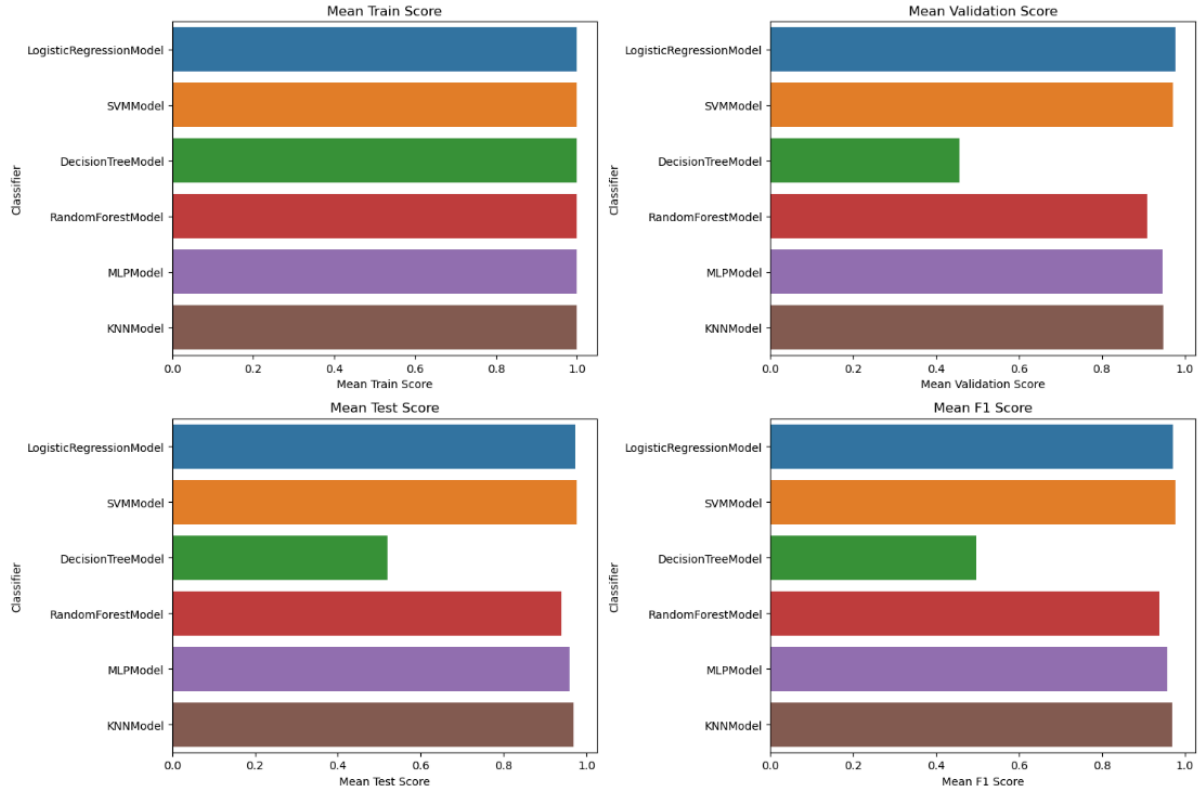

Figure 12 : Barplot représentant les scores moyens des modèles

Les scores moyens de nos modèles rejoignent les analyses effectuées précédemment.

5.2 Courbes (Sanity Checks)

Après avoir analysé les différents scores de nos modèles, il pourrait être intéressant de visualiser les courbes d'apprentissage ainsi que de validation de nos modèles. Cela fait partie des vérifications à faire sur nos modèles (ce que l'on appelle sanity checks).

5.2.1 Courbes d'apprentissage/Validation

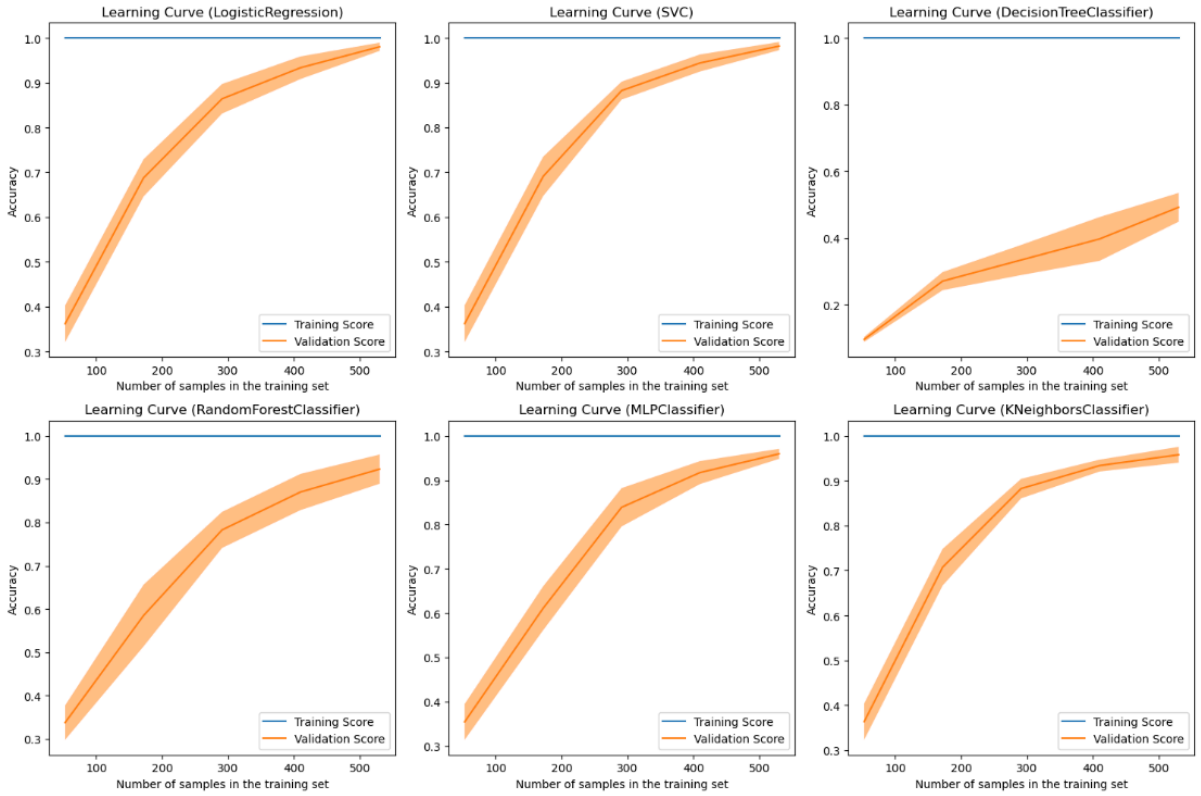

Figure 13 : Courbe d'apprentissage et de validation

Les courbes d'apprentissage et de validation représentent ici les courbes liées au score en fonction du nombre de données dans les sous-ensembles d'apprentissage. On a donc la courbe d'apprentissage comparé à la courbe de validation. La première analyse que nous pouvons effectuer est celle que l'on a fait précédemment : on du sur apprentissage sur l'ensemble de nos modèles. En effet, la courbe d'apprentissage est à 1 sur un petit nombre de données tandis que la courbe de validation est très basse (proche de 0). Pour la plupart des modèles, la courbe de validation est plutôt logique car plus le nombre de données augmente, plus le modèle arrive à prédire. Le sur apprentissage est très visible sur la courbe de l'arbre de décision qui a du mal à augmenter son score de validation.

5.2.2 Courbes d'apprentissage/validation paramètre

Maintenant que nous avons pu visualiser les courbes d'apprentissage et de validation, il serait peut être intéressant de jeter à œil à la courbe d'apprentissage et de validation en fonction de la valeur de certains paramètres de nos modèles notamment sur les termes de régularisation.

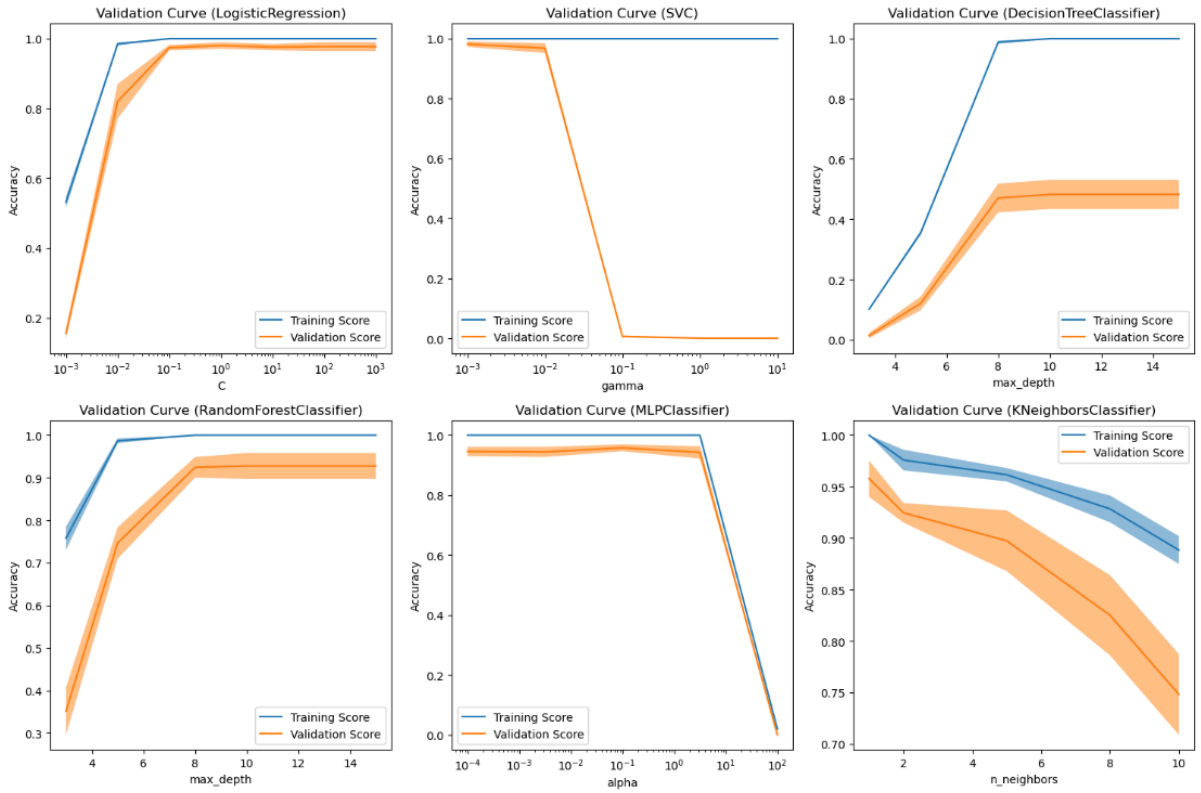

Figure 14 : Courbe d'apprentissage et de validation en fonction de la valeur de certains paramètres

<u>Régression logistique :</u>

Le terme inverse de régularisation "C" est celui que nous avons décidé d'étudier pour la régression logistique. Ce que nous observons c'est que plus "C" est grand plus on a un risque de surapprentissage, si l'on lie nos connaissances cela est absolument logique car "C" est l'inverse de la régularisation, c'est-à-dire que plus "C" est grand plus le terme de régularisation (dans notre cas L2) sera bas et donc mènera à du sur-apprentissage.

<u>SVM :</u>

Pour les machine à vecteurs de support, nous avons choisis le terme gamma. Ce que nous observons c'est que plus gamma est grand plus le sur apprentissage est présent, si l'on s'intéresse à la définition de gamma on remarque qu'il y'a la notion de la considération des points loin de la marge. Plus gamma est petit plus les points loins de la marge sont considérés et donc mènent à baisser le surapprentissage. Une grande valeur de gamma pourrait donc mener a du surapprentissage.

Arbre de décision :

Si on se souvient du fonctionnement de ce modèle, la combinaison des classifieurs stumps permettent de créer l'arbre de décision et donc plus l'arbre de décision combine de classifieur stumps plus celui-ci aura une profondeur élevé et donc mené au sur-apprentissage. C'est ce que l'on remarque avec le paramètre max_depth qui représente la profondeur de l'arbre, plus l'arbre est profond donc plus il y'a de classifieur stumps et donc plus le modèle est susceptible de surapprendre.

Forêts aléatoires :

Les mêmes observations que l'arbre de décision sont remarquées mais bien sûr dans une moindre mesure vu que les forêts aléatoires combinent les arbres de décisions via un vote majoritaire.

K plus proches voisins :

Evidemment, pour cet algorithme nous avons fait varier le nombre de voisins pris en compte. Le nombre de voisins optimal est 1, concrètement cela veut dire que le modèle attribue le poids le plus élevé à l'observation la plus proche lors de la prise de décision. Il faut faire attention car en choisissant un seul voisin, le modèle peut être très sensible au bruit ou aux variations aléatoires dans les données d'entraînement. Si une observation particulière est une aberration ou une exception, elle pourrait avoir un impact disproportionné sur les prédictions du modèle.

Sanity checks :

- On peut donc en conclure que plus la régularisation augmente, plus la perte est forte.
- Surapprentissage sur un petit nombre de donnée
- Visualisation des courbes d'apprentissage et de validation nous donne de nouvelles informations pertinentes sur nos modèles

5.2.3 Courbes ROC

Enfin, nous pouvons aussi vérifier les courbes roc, c'est une métrique que nous avons aussi pu voir dans notre cours. C'est un outil graphique qui nous permet d' évaluer les performances d'un modèle en fonction de différents seuils de classification. Cette courbe enregistre les taux de faux et de vrais positifs en fonction de différents seuils.

5.2.3.1 Courbes Roc pour tous les labels

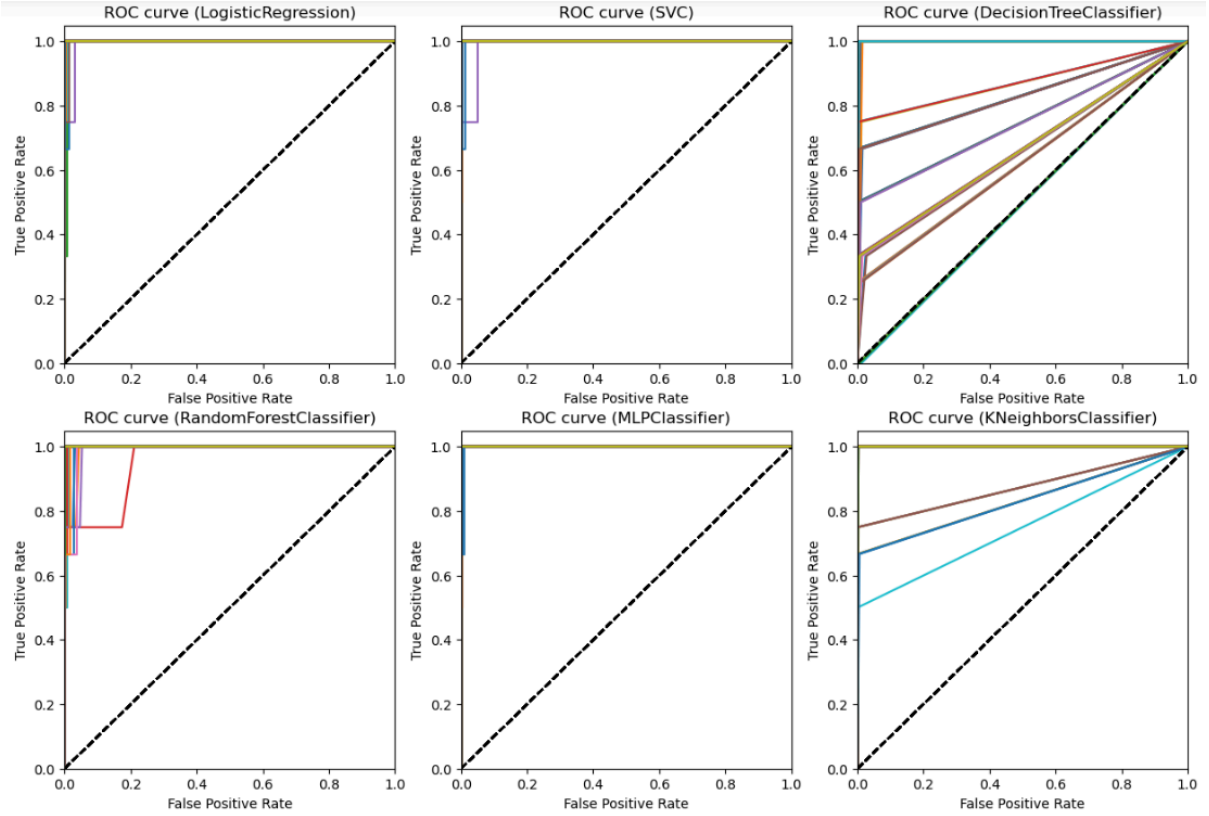

Figure 15 : Courbe roc pour chaque modèle pour toutes les classes de notre variable cible

Nous construisons des courbes roc pour chaque classe de notre variable cible. Les arbres de décisions sont les modèles qui présentent la "moins bonne" courbe. Quelques classes de l'algorithme des plus proches voisins présentes aussi une courbes roc inférieur à 1. Ce que l'on peut affirmer, c'est que nos modèles se rapprochent pour la plupart de la courbe roc idéale. En effet, le taux de faux positifs = au taux de faux négatifs = 0.

5.2.3.1 Courbes Roc moyenne

Il serait aussi intéressant de faire la moyenne des courbes roc.

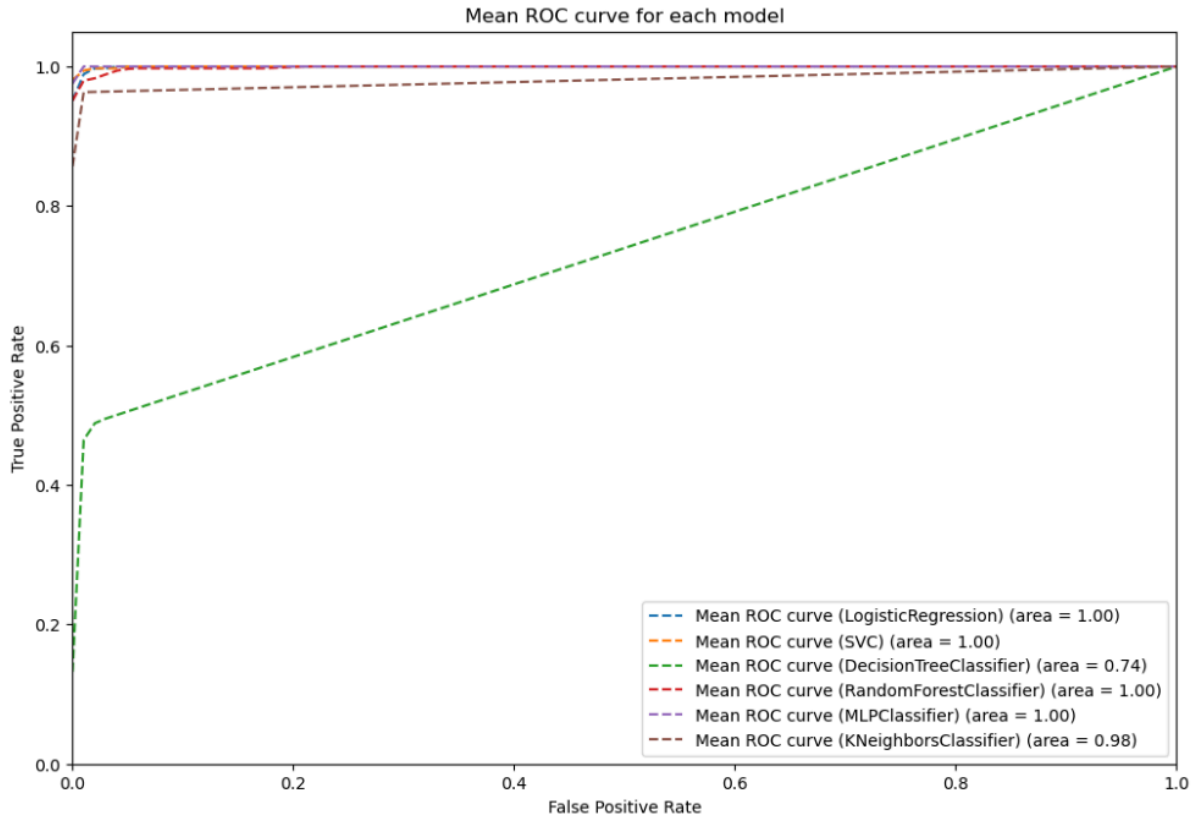

Figure 16 : Courbe roc moyenne des modèles

La courbe moyenne confirme nos observations précédentes, à noter que la valeur moyenne de l'algorithme des k plus proches voisins est bien plus proche des autres modèles que de l'arbre de décision.

Sur l'ensemble de nos analyses, on peut en conclure que le modèle d'arbre de décision est celui qui performe le moins bien, à cause notamment de sa grande faculté à surapprendre. L'impact du terme de régularisation se fait clairement sentir au sein de nos modèles comme le montre les courbes d'apprentissage et de validation. Nous avons le sentiment que malgré nos très bons résultats sur l'ensemble de test que nos modèles sont trop complexes pour un tel jeu de données.

6 Réduction de dimensionnalité

Le nombre d'attributs étant très élevé (183 après prétraitement), il serait peut être intéressant de faire la même chose sur un jeu de données différent qui contient moins de caractéristiques. Un grand nombre d'attributs peut rendre les modèles moins interprétables. Plus il y a d'attribut, plus il devient difficile de comprendre comment chaque attribut contribue à la prédiction finale. Aussi, plus le nombre d'attribut est élevé, plus l'espace des caractéristiques (dimensions) devient grand. Cela signifie que les données deviennent plus dispersées et que le modèle a besoin de plus de données pour généraliser efficacement. Dans notre cas, nous ne disposons pas d'une grande base de données (seulement 990), le choix de réduire le nombre d'attributs est donc une bonne chose en théorie.

6.1 Sélection d'attributs

Nous avons eu l'occasion d'étudier différentes manières de réduire la dimensionnalité de nos données. Dans ce projet, nous utiliserons la méthode d'élimination récursive des attributs dite RFE. C'est une méthode multivariée, c'est-à-dire qu'elle considère toutes les caractéristiques simultanément lors de l'élimination. L'algorithme est le suivant :

- Initialisation du processus : Tous les attributs sont inclus dans le modèle.
- Entraînement du modèle : Un modèle est entraîné sur l'ensemble de données complet.
- Évaluation des attributs : Une fois le modèle entraîné, l'importance de chaque attribut est évaluée.
- Élimination des attributs les moins importants : Les attributs les moins importants sont ensuite éliminés du jeu de données.
- Itération : Les étapes 2 à 4 sont répétées jusqu'à ce qu'un certain critère d'arrêt soit atteint, tel que le nombre de features restantes atteignant un seuil prédéfini ou que la performance du modèle cesse de s'améliorer.
- Retour du jeu de données réduit : Une fois toutes les itérations terminées, le jeu de données réduit est renvoyé, contenant uniquement les features jugés les plus importants par la méthode RFE.

Dans notre cas, nous avons choisi un modèle de forêt d'arbre de décision pour évaluer l'importance de chaque features. Ici on vient utiliser un ensemble d'arbre de décision (comme vu en cours) qui vient déterminer l'importance de chaque caractéristique par la mesure de qualité entré en paramètre de notre méthode. Cela peut être le critère de Gini ou la mesure de l'entropie (cf : atelier 5). Le critère de base de la fonction RandomForestClassifier() est le critère de Gini :

$$Gini(F) = 1 - \sum_{i=1}^{c} p_i^2$$

Figure 17 : Formule du critère de Gini

- F : Attribut de l'ensemble de données
- c : le nombre de classes de l'attribut
- p_i : le nombre d'objets dans la classe i divisé par le nombre total d'objet dans F

En pratique, nous faisons appel à la fonction rfe_selection() de la classe Preprocessor() qui utilise la méthode RFE de scikit learn. Après application de l'algorithme d'élimination, nous avons réduit la dimension des données passant de 183 à 91 attributs, la réduction permet de réduire de plus de moitié les dimensions du jeu de données. Appliquons maintenant le même schéma d'entraînement qu'auparavant mais avec cette fois les 91 attributs choisis par l'algorithme et étudions les résultats obtenus.

6.1.1 Résultats obtenus

6.1.1.1 Score d'apprentissage, validation et de test

	Classifier	Mean Train Score	Mean Validation Score	Mean Test Score	Mean F1 Score
0	LogisticRegressionModel	1.0	0.956254	0.966361	0.964813
1	SVMModel	1.0	0.944201	0.967584	0.966682
2	DecisionTreeModel	1.0	0.494771	0.489908	0.477679
3	RandomForestModel	1.0	0.891445	0.889297	0.885776
4	MLPModel	1.0	0.932114	0.952294	0.952274
5	KNNModel	1.0	0.929152	0.949235	0.947921

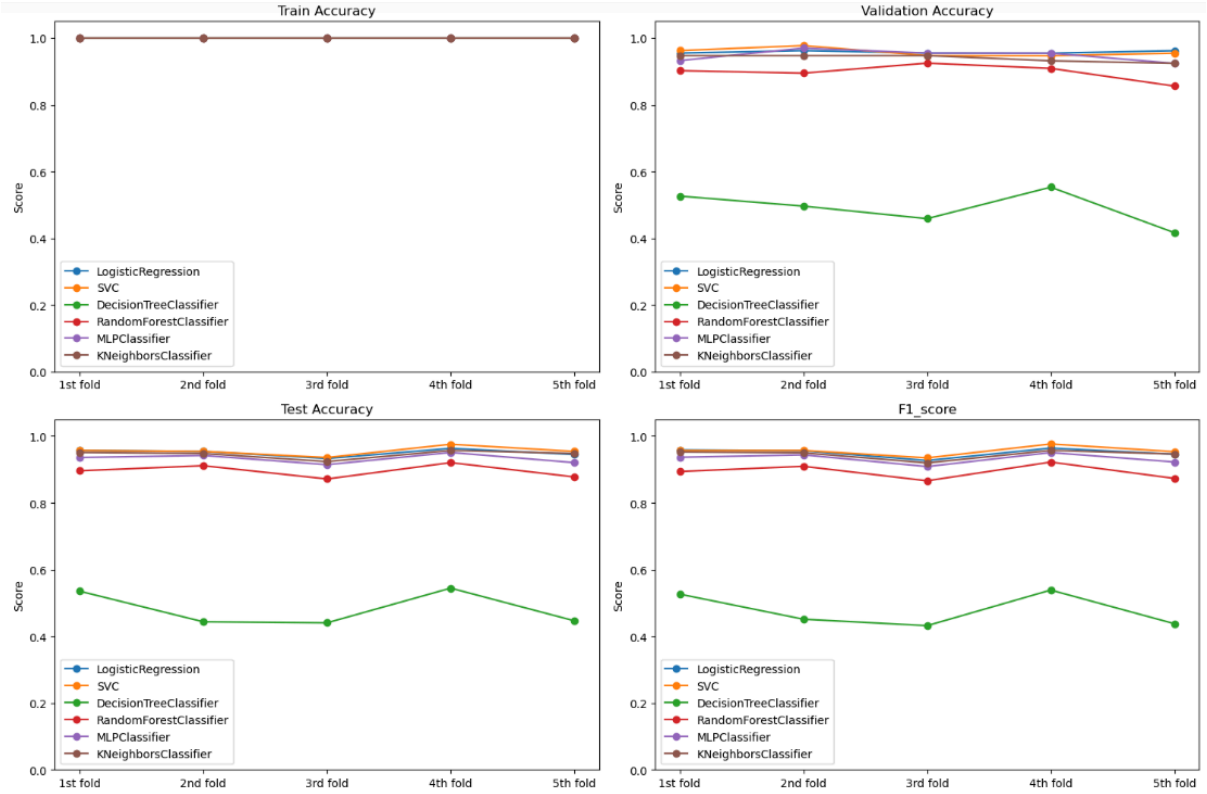

Figure 18 : Score des modèles après sélection d'attributs

6.1.1.2 Score moyen

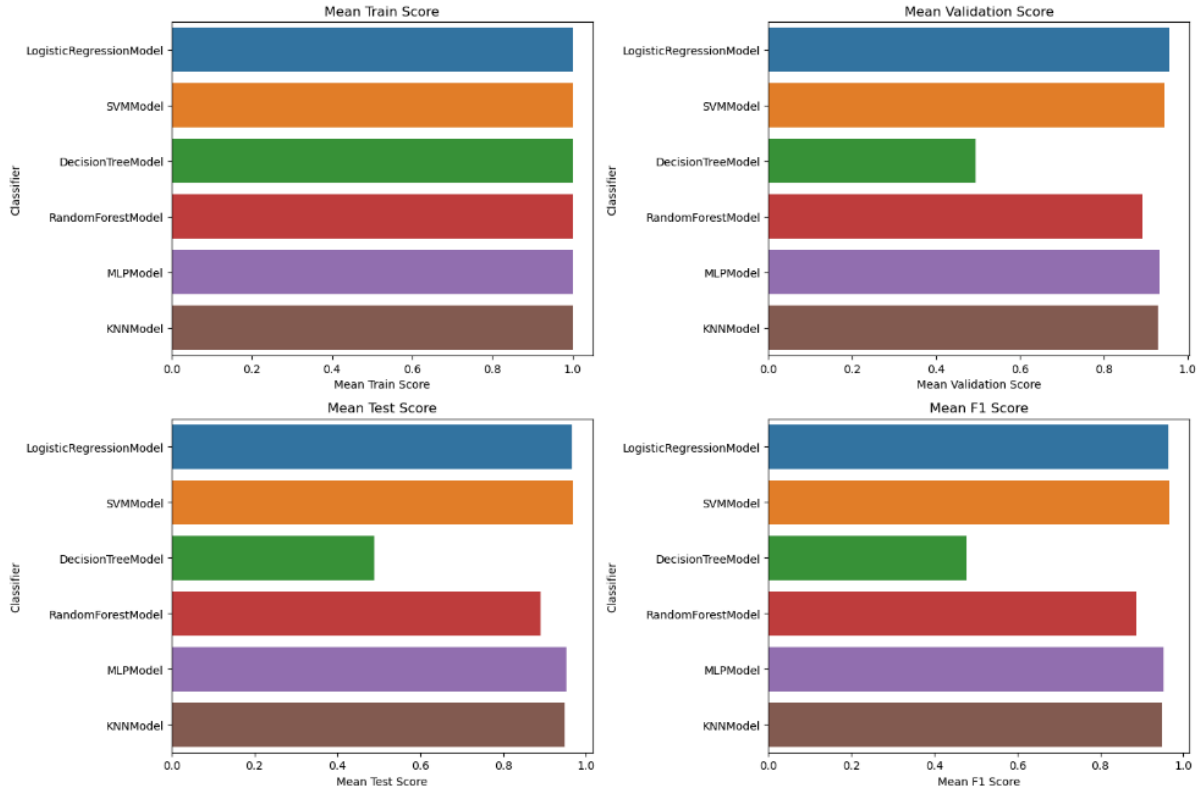

Figure 19 : Score moyen des modèles après sélection d'attributs

6.1.1.3 Courbes d'apprentissage et de validation

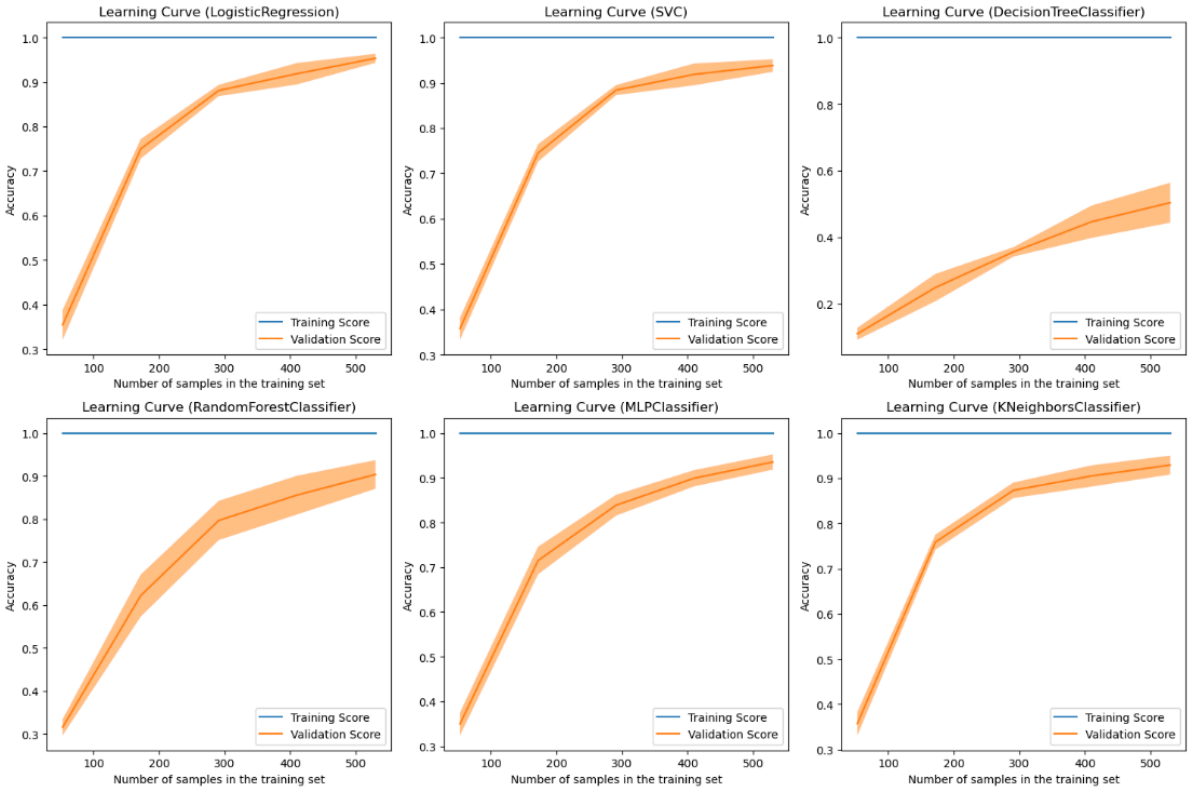

Figure 20 : Courbe d'apprentissage et de validation après sélection d'attributs

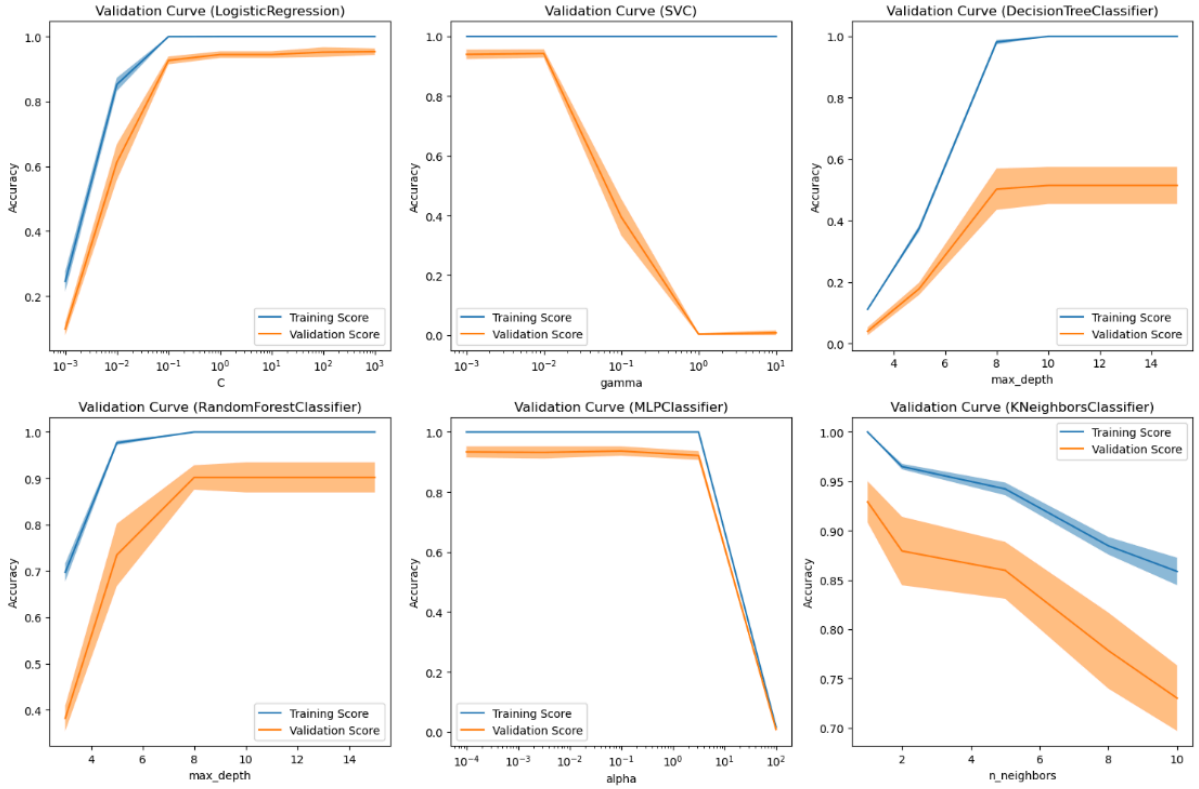

Figure 21 : Courbe d'apprentissage et de validation en fonction de certains paramètres après sélection d'attributs

6.1.1.4 Courbes ROC

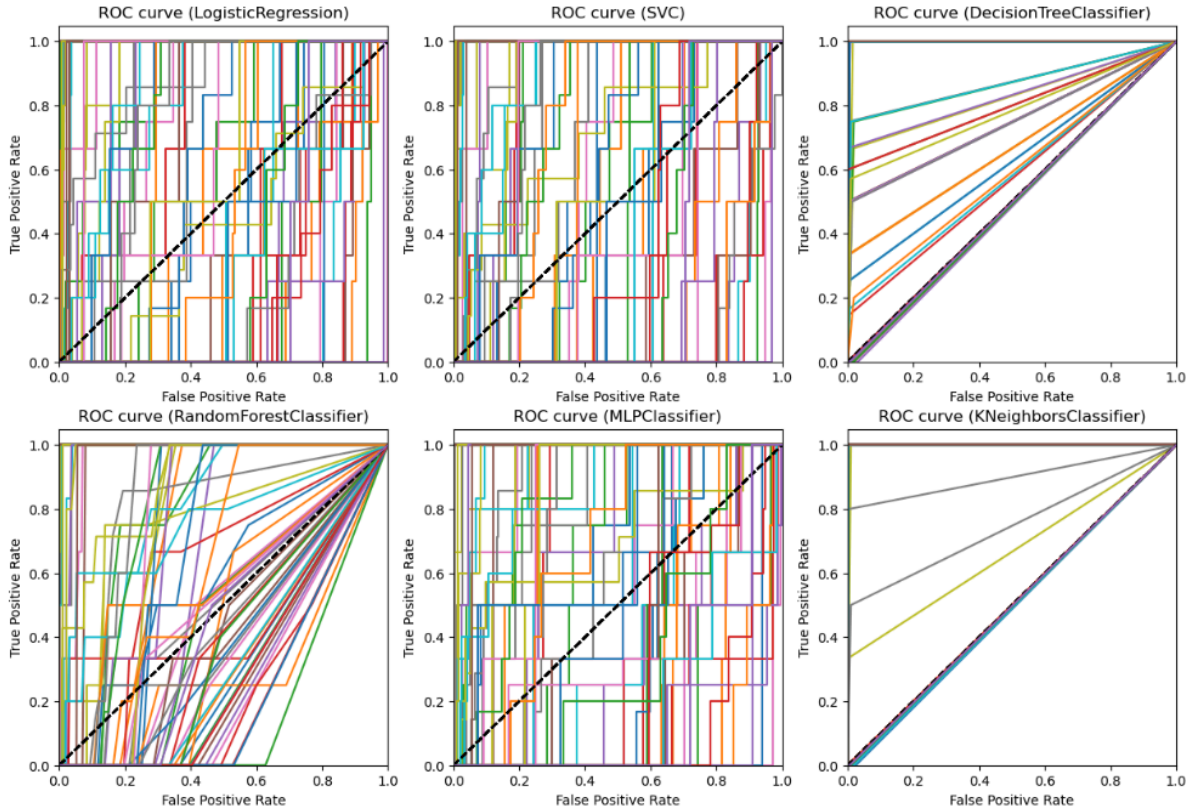

Figure 22 : Courbe roc pour chaque modèle pour toutes les classes de notre variable cible après sélection d'attributs

6.1.1.5 Courbe ROC moyenne

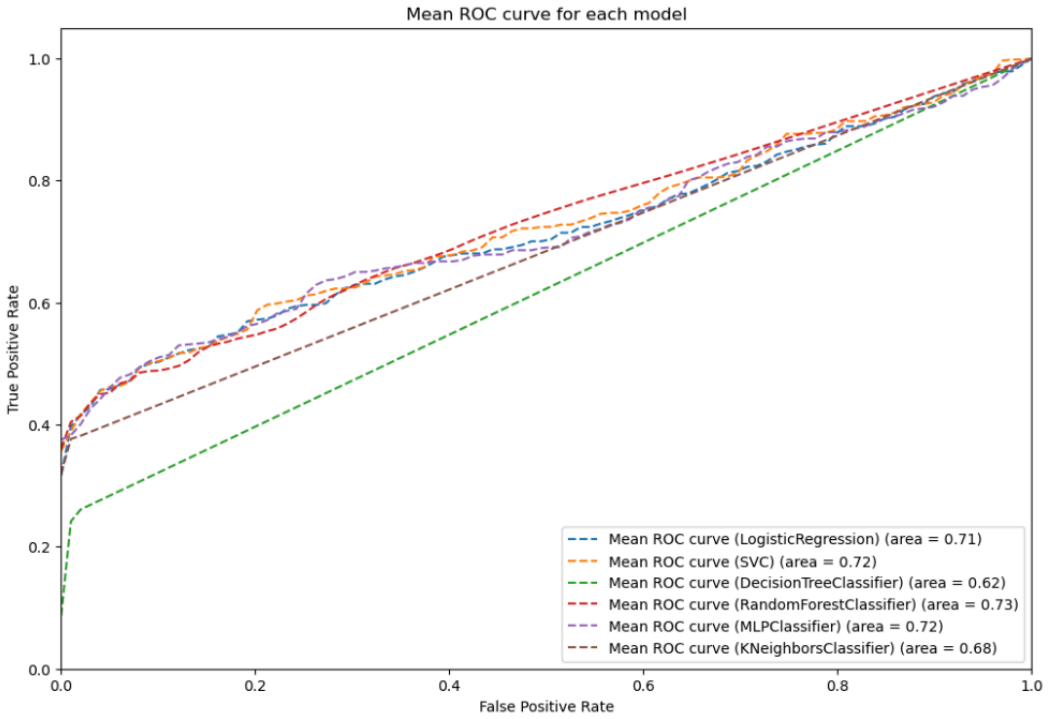

Figure 23 : Courbe roc moyenne pour chaque modèle après sélection d'attributs

6.1.2 Observations

Les scores de nos modèles restent pratiquement identiques à ceux avant la sélection d'attributs (94.4% d'accuracy en moyenne sur l'ensemble de test en dehors des arbres de décision). Les courbes d'apprentissages et de validation sont aussi très identiques ce qui est normal car les comportements des modèles sont restés les mêmes vis à vis des hyperparamètres. Le changement s'effectue sur les courbes ROC où on aperçoit une baisse moyenne sur l'ensemble des modèles. Cela peut s'expliquer par le fait qu'en réduisant les dimensions du jeu de données par 2, une perte d'information se fait ressentir. Cependant, dans l'ensemble nous pouvons affirmer que la réduction de la dimensionnalité par l'élimination récursive des attributs est une méthode efficace qui nous permet de réduire la dimensionnalité de nos données sans grosse perte de précision de nos modèles.

6.2 Analyse en composantes principales

Tout comme la réduction de la dimensionnalité par l'élimination récursive des attributs (RFE), l'analyse en composantes principales permet de réduire la dimensionnalité pour simplifier la complexité des données tout en préservant autant que possible leur structure. L'ACP utilise la matrice de covariance des données pour extraire les composantes principales, ainsi la corrélation entre les variances est une indication de s' il pourrait être intéressant ou non de l'appliquer. Étudions la corrélation de nos 91 attributs obtenus après la sélection d'attributs.

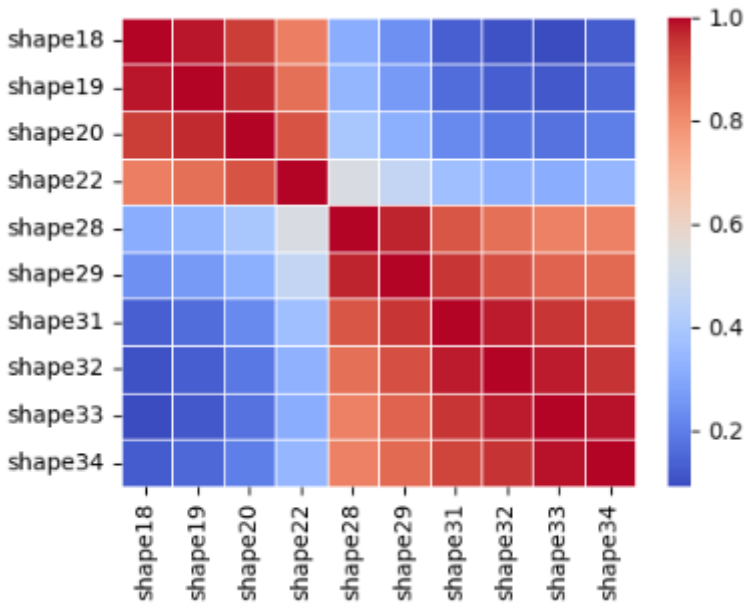

Figure 24 : Exemple d'une partie de la matrice de corrélation (voir notebook pour voir l'entièreté des matrices)

La première observation que nous pouvons faire est que certains attributs sont fortement corrélés entre eux. On peut prendre l'exemple des attributs "shape33" et "shape34" qui sont très fortement corrélés.. Autre exemple, l'attribut "shape33" et "shape18" sont fortement corrélés négativement. Ces observations nous indiquent que certains attributs portent en eux les mêmes informations, il serait donc intéressant d'appliquer une méthode comme ACP afin de réduire la dimensionnalité de nos données.

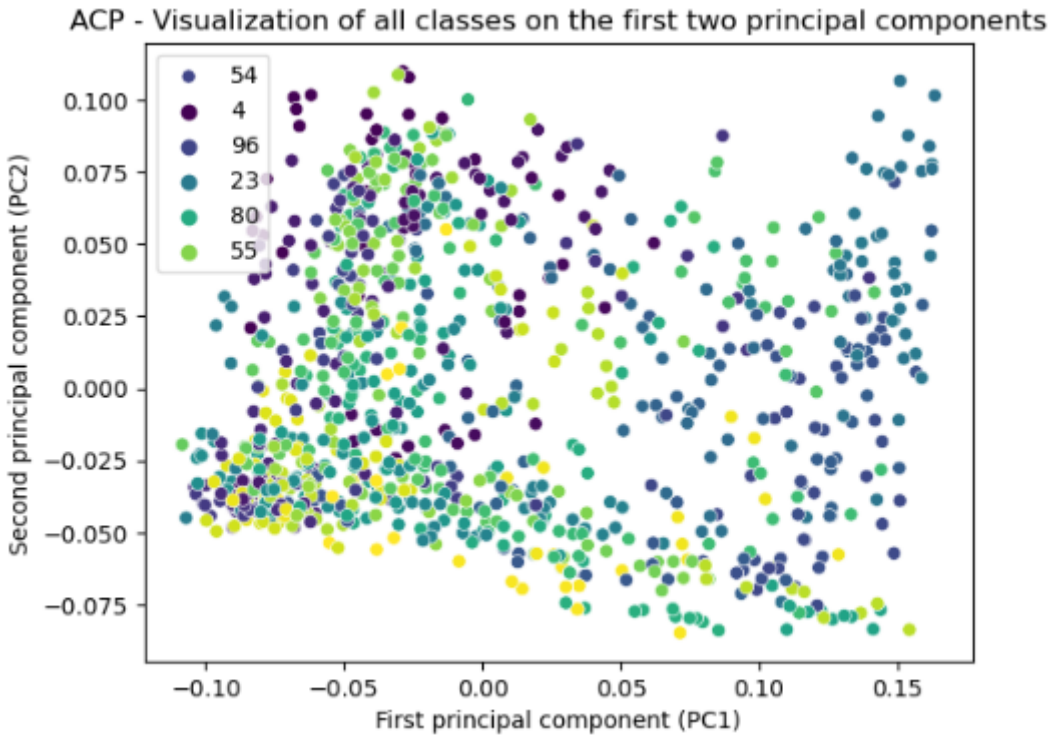

Figure 25 : Visualisation de l'ensemble des classes sur les deux premières composantes principales

L'analyse visuelle des deux composantes principales nous montre que la plupart des points projetés sur ces deux axes se chevauchent, cela nous indique que deux composantes ne suffiront pas à bien séparer les données. Confirmons nos analyses avec l'étude de la variance expliquée par les composantes principales.

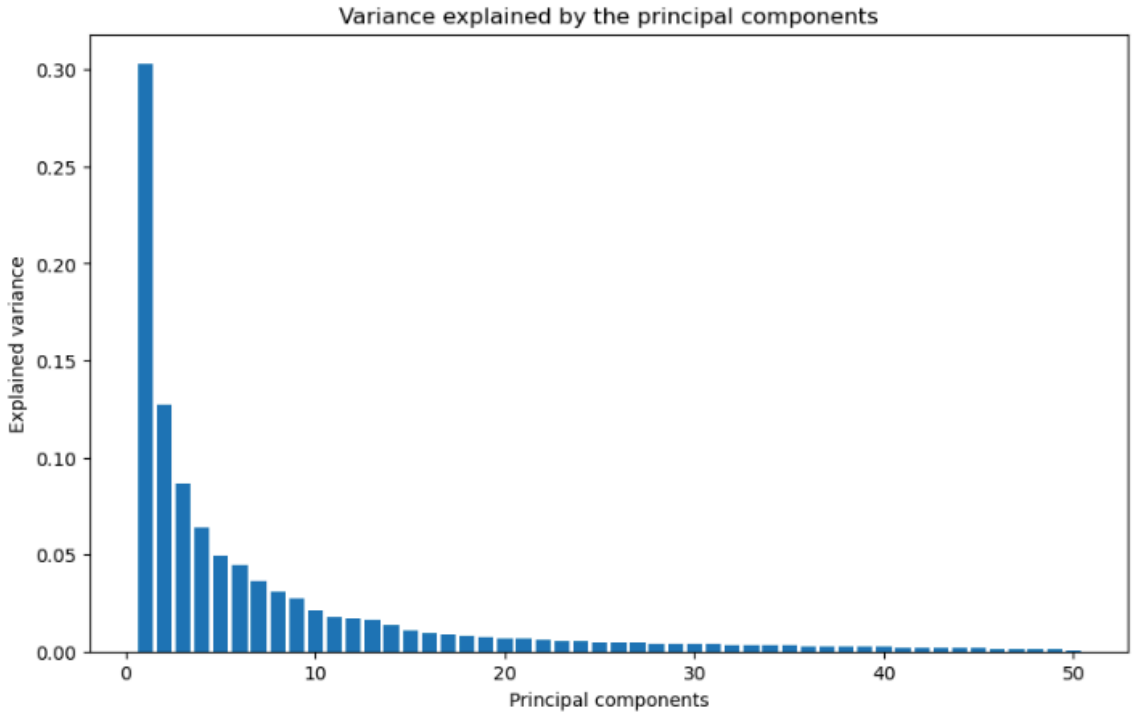

Figure 26 : Variance expliquée par les composantes principales

L'analyse de la variance expliquée par les différentes composantes principales nous informe que l'information est répartie à travers un plus grand nombre de composantes principales que ce que nous avions l'habitude de voir au cours des tp et atelier où l'information était contenue très majoritairement dans les 3 premières composantes. Ici, si on regarde au niveau du ratio expliqué par les 20 premières composantes, on trouve que 90% de la variance est expliquée à travers ces 20 composantes. Il serait donc peut être intéressant de prendre 20 composantes principales, au-delà l'évolution de la variance expliquée semble assez faible pour que l'on puisse ne pas les considérer.

6.2.1 Résultats obtenus

6.2.1.1 Score d'apprentissage, validation et de test

	Classifier	Mean Train Score	Mean Validation Score	Mean Test Score	Mean F1 Score
0	LogisticRegressionModel	0.999623	0.802381	0.803058	0.802573
1	SVMModel	0.995099	0.820563	0.831193	0.830339
2	DecisionTreeModel	1.000000	0.410196	0.380428	0.373446
3	RandomForestModel	1.000000	0.752518	0.774312	0.765936
4	MLPModel	1.000000	0.826532	0.836697	0.836382
5	KNNModel	1.000000	0.767715	0.757187	0.749851

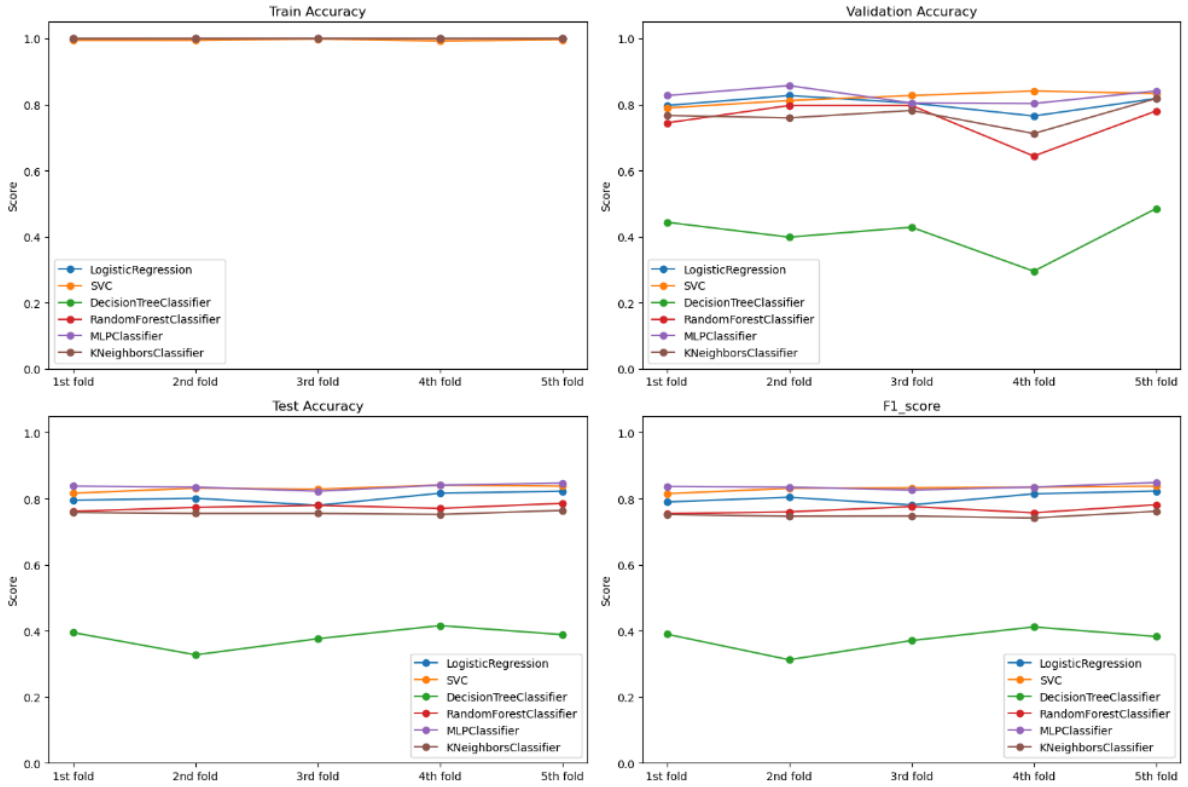

Figure 27 : Score des modèles après l'ACP

6.2.1.2 Score moyen

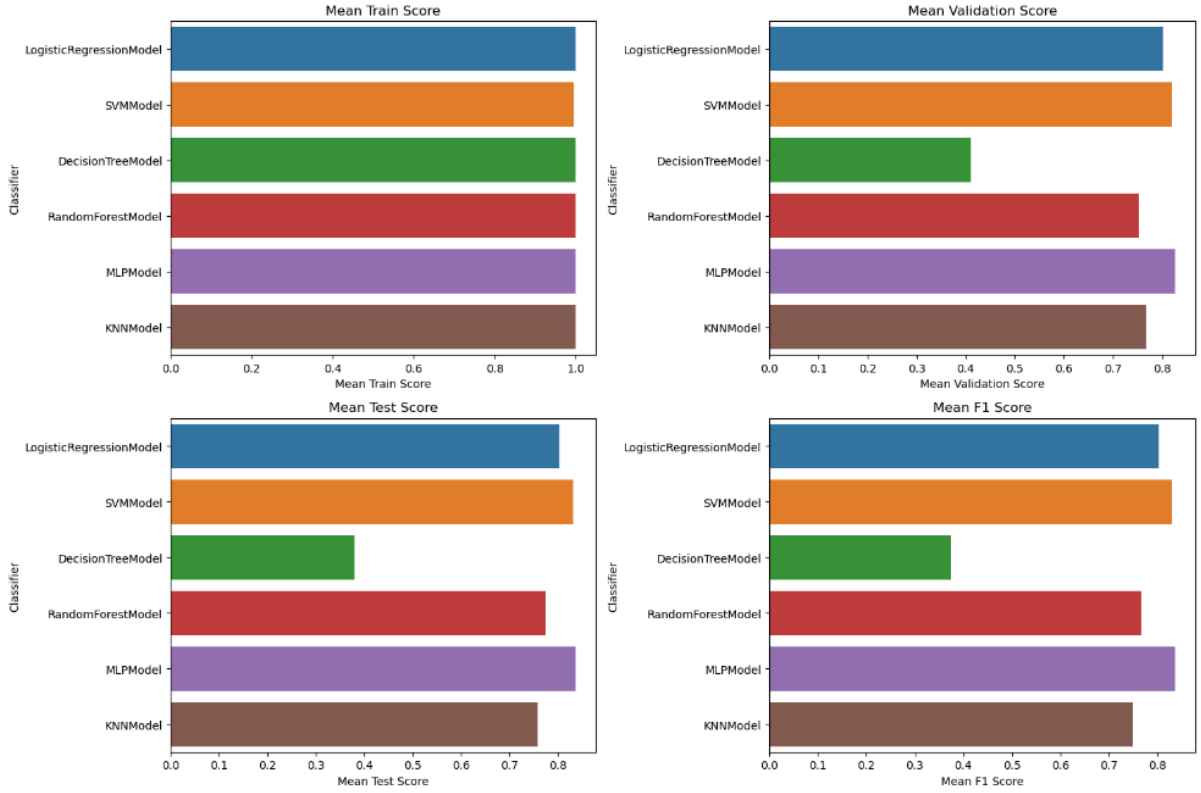

Figure 28 : Score moyen des modèles après l'ACP

6.2.1.3 Courbes d'apprentissage et de validation

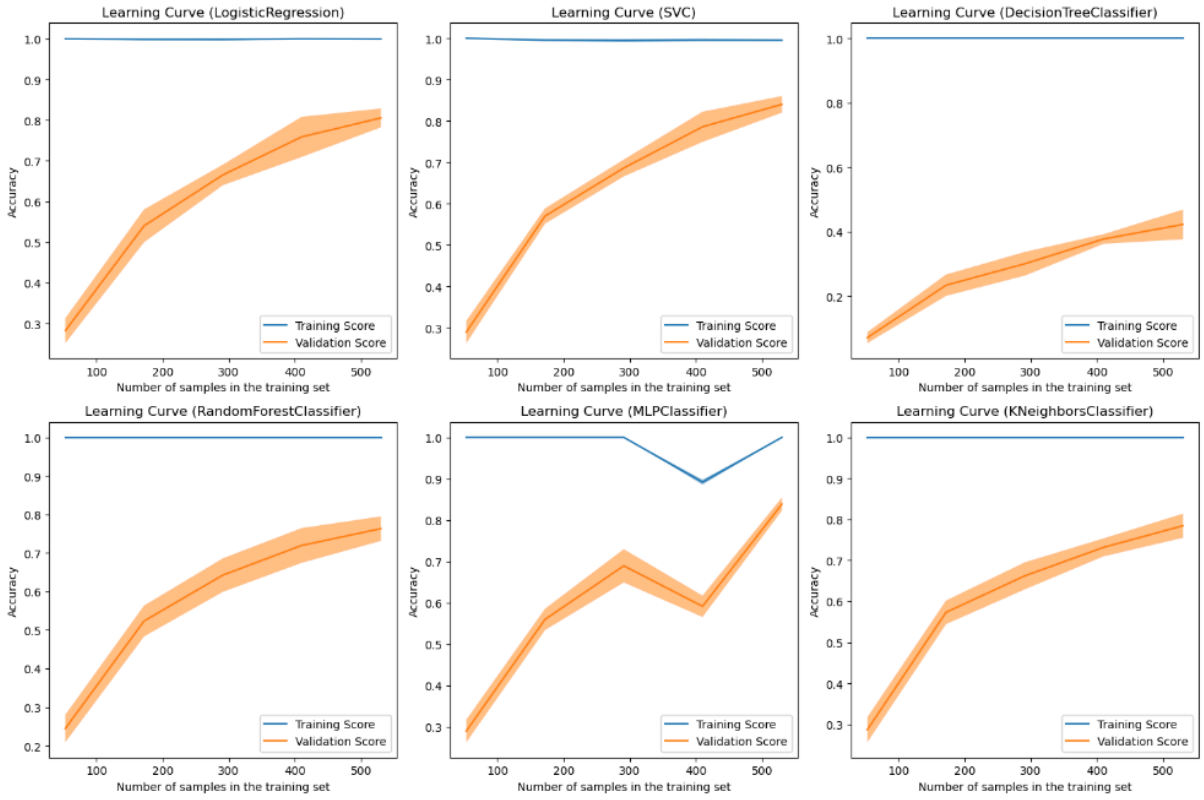

Figure 29 : Courbe d'apprentissage et de validation après l'ACP

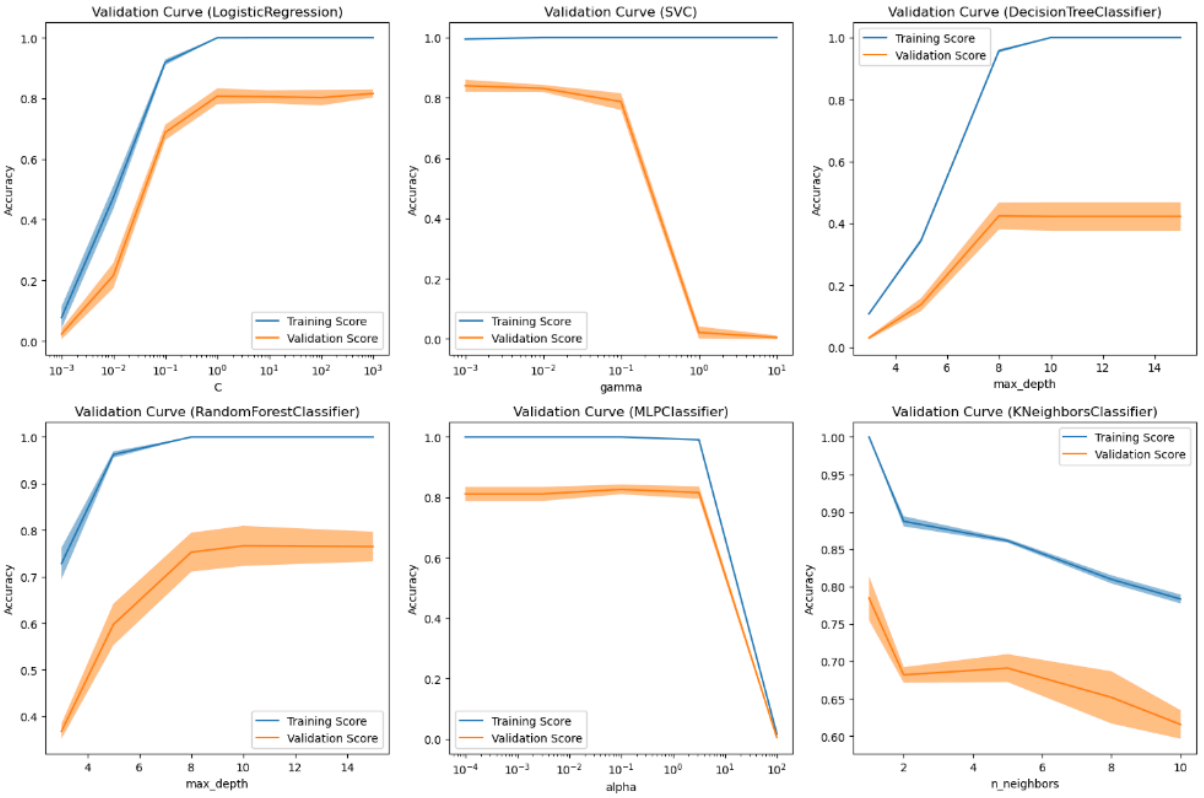

Figure 30 : Courbe d'apprentissage et de validation en fonction de certains paramètres après l'ACP

6.2.1.4 Courbes ROC

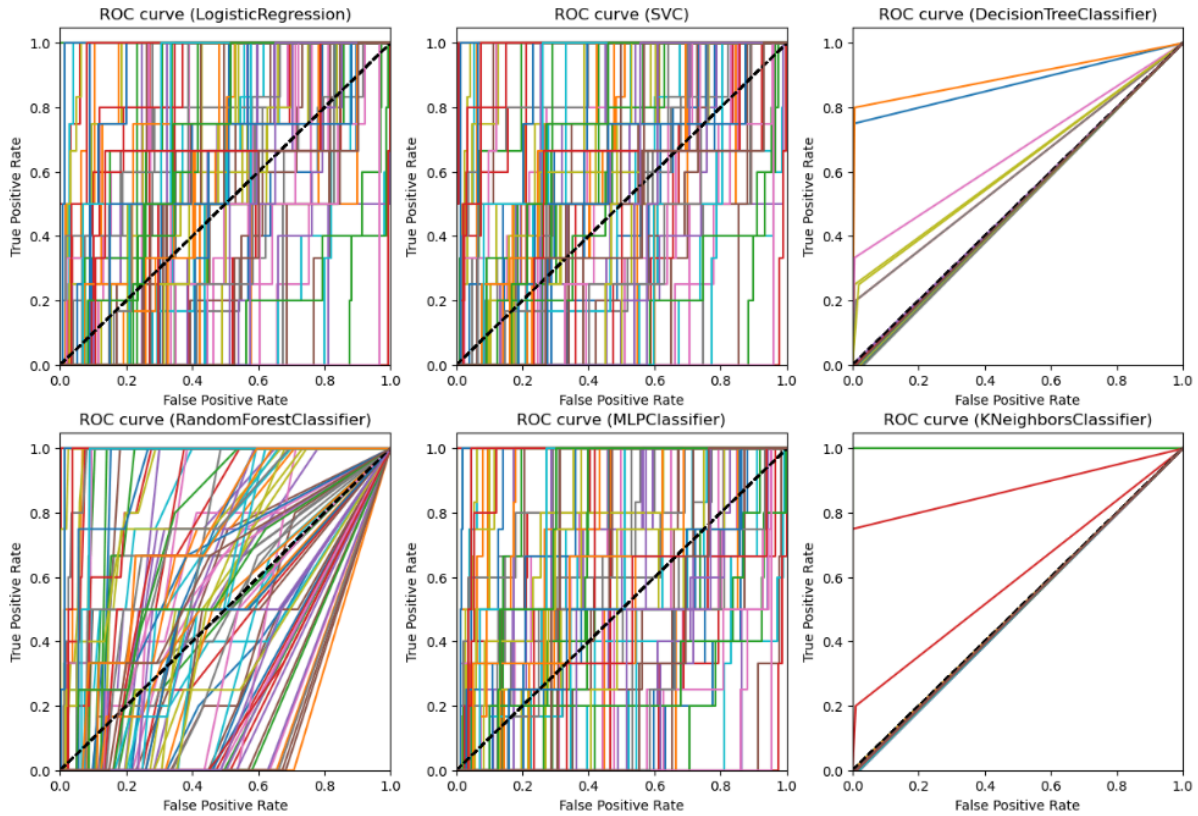

Figure 31 : Courbe roc pour chaque modèle pour toutes les classes de notre variable cible après l'ACP

6.2.1.5 Courbes ROC moyenne

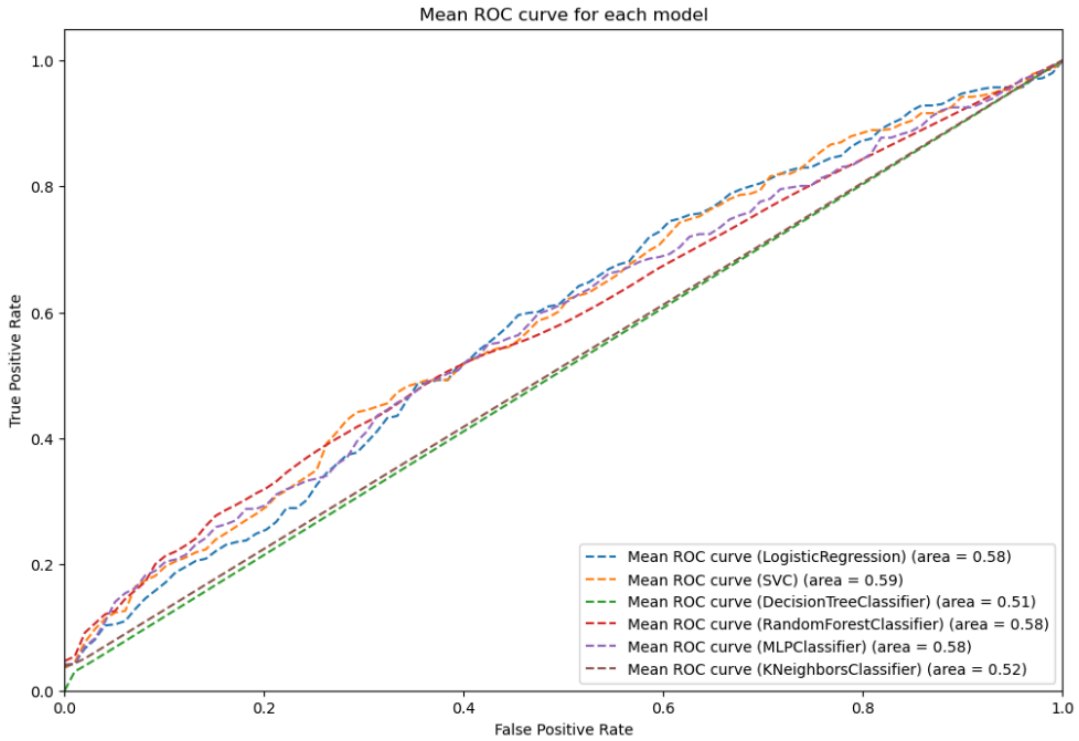

Figure 32 : Courbe roc moyenne pour chaque modèle après l'ACP

6.2.2 Observations

On observe une baisse assez importante de la précision de nos modèles contrairement à ce que l'on avait l'habitude de voir (80% au lieu de 96 et 94% sur l'ensemble de test en dehors des arbres de décisions). Cependant, cela paraît totalement logique vu que l'on vient appliquer une réduction de dimension à partir d'une sélection d'attribut qui est elle-même une réduction de dimension, une perte d'information est forcément observée. Cette perte d'information s'aperçoit aussi sur les courbes ROC. Cependant, l'utilisation de l'ACP reste convaincante car on a réduit la dimensionnalité drastiquement en l'appliquant sur les 91 attributs. Si on regarde au global, on passe de 184 à 20 attributs ! La précision reste très correcte. Nous sommes satisfaits de ces résultats.

7 Conclusion et lien avec notre problématique

On rappelle notre problématique initiale : La science des données et le machine learning peuvent-ils nous aider dans la tâche de différenciation des espèces végétales ?

Les modèles que nous avons créés montrent une grande précision dans leur capacité à prédire les espèces végétales. Malgré un surapprentissage observé, les modèles généralisent bien sur des données qu'ils n'ont jamais vues. De plus, les pratiques telles que la validation croisée, l'algorithme de grid search, la normalisation des données ou encore la réduction des dimensions sont des techniques qui ont montré leur utilité tout au long du projet. La majorité de ce que nous avons construit au travers de ce projet montre donc qu'il est possible d'utiliser la science des données comme outils sur lequel peuvent se baser les experts du domaine afin, dans un premier temps de différencier les espèce végétales mais aussi, en voyant plus loin, de prévenir des incendies, préserver des écosystèmes forestiers et même optimiser les pratiques agricoles pour une production plus efficace et durable. Bien sûr, notre projet présente des améliorations envisageables. Nous pouvons par exemple utiliser des données de type image sur lequel nous pourrions appliquer des algorithmes bien plus robustes que ceux utilisés dans ce projet comme des réseaux à convolutions par exemple. Aussi, le nombre de données que nous disposons pourrait être bien plus grand afin d'améliorer la généralisation du modèle et ne pas surapprendre sur le peu de données que nous disposons. Cependant, les idées derrière ce projet reste plus que valable et nous sommes satisfaits de ce que nous avons proposés.

8 Références bibliographiques

- [1] Mohammad Keivani, Jalil Mazloum, Ezatollah Sedaghatfar, Mohammad Bagher Tavakoli, 2021. *Automated Analysis of Leaf Shape, Texture, and Color Features for Plant Classification.*
- [2] James S. Cope, David Corney , Jonathan Y. Clark, Paolo Remagnino, Paul Wilkin, 2012. *Plant species identification using digital morphometrics: A review*
- [3] Neeraj Kumar, Peter N. Belhumeur, Arijit Biswas, David W. Jacobs, W. John Kress, Ida Lopez, and Jo˜ao V. B. Soares. *Leafsnap: A Computer Vision System for Automatic Plant Species Identification.*
- *https://scikit-learn.org/stable/*
- *https://seaborn.pydata.org/*
- *https://matplotlib.org/*
- *https://numpy.org/*
- *https://pandas.pydata.org/*