Rapport de projet

IFT 712 - Technique d'apprentissage

Automne 2023

Étudiants

	Courriel	CIP
Anne-Sophia LIM	lima1401@usherbrooke.ca	lima1401
Yanis PERRIN	pery2002@usherbrooke.ca	pery2002
Paul TISSEDRE	tisp1301@usherbrooke.ca	tisp1301

Lien Github : https://github.com/YanisPerrin/IFT712

Sommaire

1 Introduction....3
2 La base de données....3
3 Organisation du projet....4
3.1 Arborescence et diagramme de classe....4
3.2 Gestion de projet....6
4 Importation du jeu de données....6
5 Analyse et prétraitement....6
5.1 Données aberrantes....6
5.2 Écart interquartile et KNN imputer....9
5.3 Mélange et séparation des données....10
6 Algorithme grid search et validation croisée....10
6.1 Grid Search....10
6.2 Validation croisée....11
6.2.1 Standardisation des données....12
7 Les 6 modèles retenus....12
7.1 La régression logistique....12
7.2 Machine à vecteur de support....13
7.3 Arbres de décision....13
7.4 Forêts aléatoires....14
7.5 Perceptron multicouches....14
7.6 Classification par algorithme des K plus proches voisins....15
8 Analyse des résultats et démarche scientifique....17
8.1 Score validation croisée....17
8.1.1 Score d'apprentissage, validation et de test....17
8.1.2 Score F1....18
8.1.3 Score moyen....19
8.2 Courbes (Sanity Checks)....20
8.2.1 Courbes d'apprentissage/Validation....20
8.2.2 Courbes d'apprentissage/validation paramètre....21
8.2.3 Courbes ROC....23
8.2.3.1 Courbes Roc pour tous les labels....23
8.2.3.1 Courbes Roc moyenne....24

1 Introduction

L'évolution rapide de l'informatique et l'expansion d'internet ont engendré une explosion sans précédent de la quantité de données disponibles, notamment sur des plateformes comme Kaggle. Cette abondance de données a positionné la science des données au cœur des préoccupations des chercheurs et des entreprises. En effet, la capacité à extraire des informations significatives à partir de ces vastes ensembles de données représente un avantage concurrentiel crucial.

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 comme mentionné précédemment. Dans le cadre de notre projet IFT 712, nous nous pencherons sur une base de données provenant de Kaggle (Leaf classification). L'objectif principal est d'évaluer les performances de six méthodes de classification différentes, toutes mises en œuvre grâce à la bibliothèque bien établie scikit-learn.

Au travers de ce rapport, nous verrons des pratiques telles que la suppression et le remplacement de valeurs aberrantes, l'utilisation de la validation croisée ainsi que de l'algorithme grid search afin d'identifier la meilleure solution possible pour résoudre le problème. Puis, nous évaluerons nos résultats via différentes métriques d'évaluation sans oublier d'apporter une démarche scientifique à chaque résultat observé.

2 La base de 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.

Le jeu de données présente à l'origine 990 observations pour 193 caractéristiques. Chaque observation est associée à l'image d'une feuille noire binaire sur fond blanc. La donnée cible, l'espèce, est ici décliné en 99 classes distinctes qui comportent chacune 10 observations. Le jeu de données est donc bien balancé (pas de classe plus présente que d'autres).

Aperçu des données à l'origine :

	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

3 Organisation du projet

3.1 Arborescence et diagramme de classe

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

```
|   README.md
|   requirements.txt
|
├───data                    <-- Données brutes
|       test.csv
|       train.csv
|
└───src                     <-- Code source du projet
    |   projet_ift712.ipynb <-- Notebook associé au projet
    |
    ├───.ipynb_checkpoints
    |       projet_ift712-checkpoint.ipynb
    |
    ├───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 1 : Arborescence du projet (1/2)

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

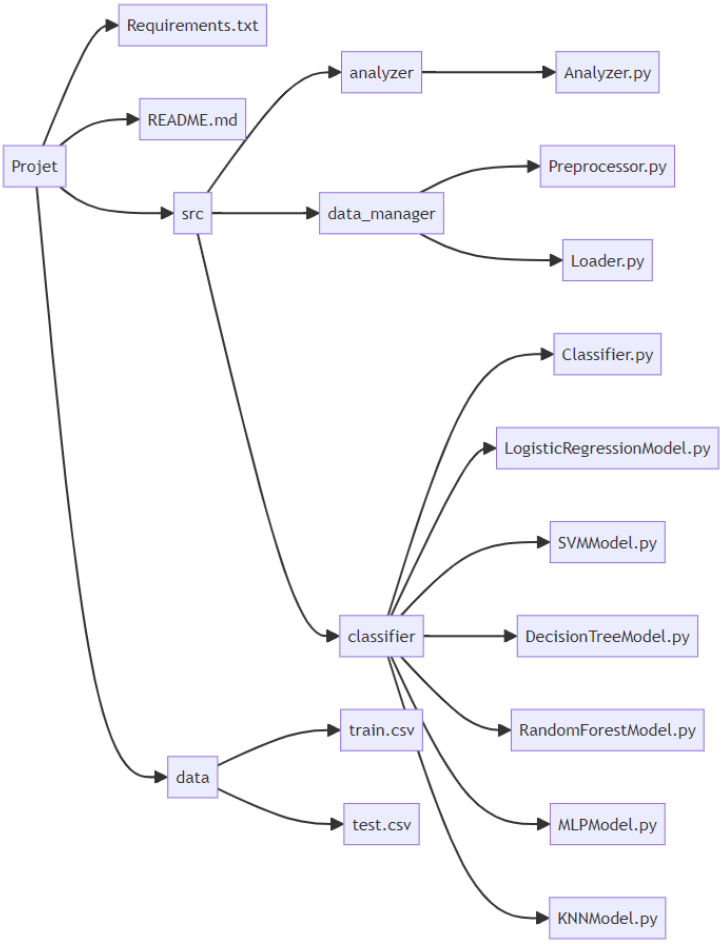

Figure 2 : Arborescence de notre projet (2/2)

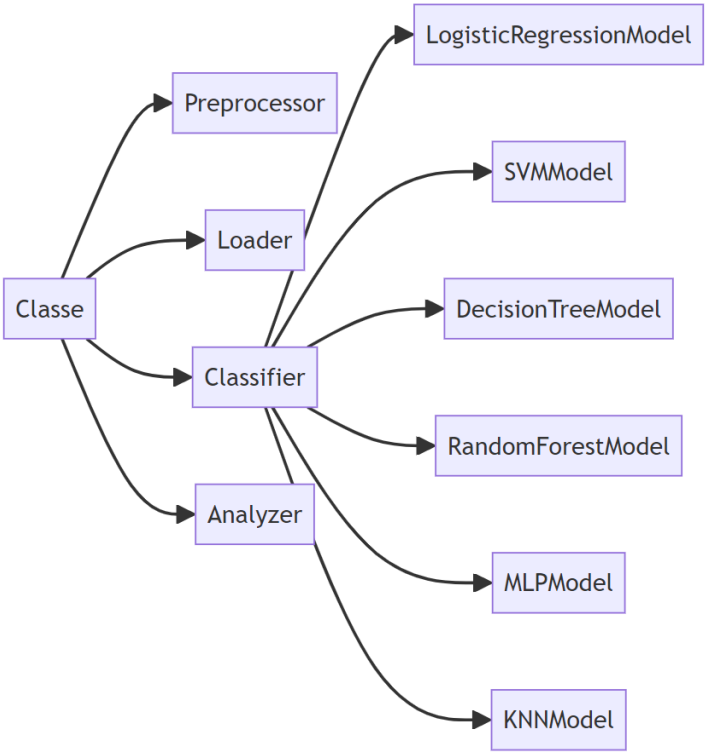

Figure 3 : Diagramme de classe

3.2 Gestion de projet

En ce qui concerne la gestion de projet nous avons mis en place un Trello comme conseillé ainsi qu'un github qui nous a permis de mieux gérer notre projet via la gestion de versionning implanté par celui-ci. Celui-ci contient toutes les informations nécessaires à la visualisation de notre projet.

Lien Github : https://github.com/YanisPerrin/IFT712
Lien Trello : https://trello.com/b/BSflAjuJ/organization

4 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.

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").

5 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 13 fonctions distinctes que nous présenterons au fil du rapport.

5.1 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. 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 4 : 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() :

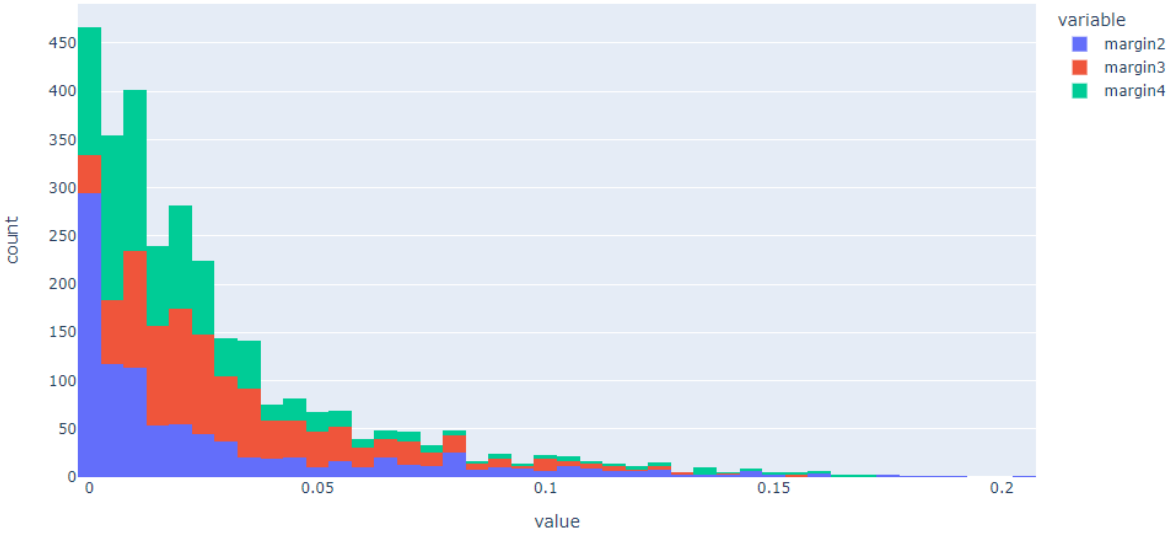

Figure 5 : Histogramme du comptage des différentes valeurs des variables “margin2”, “margin3” et “margin4”.

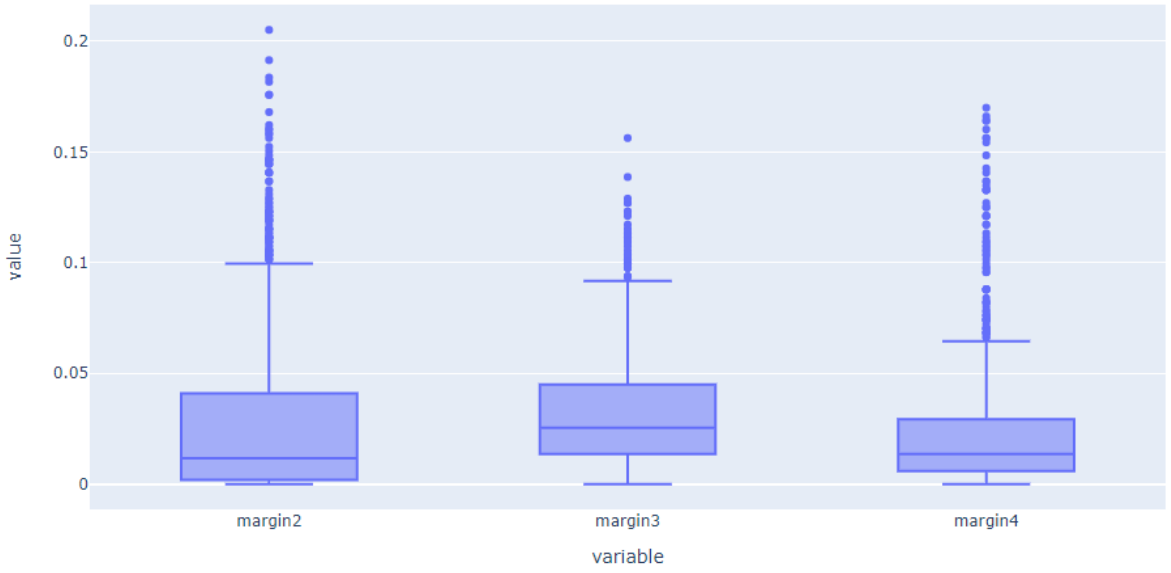

Figure 6 : Boxplot des différentes valeurs des variables “margin2”, “margin3” et “margin4”.

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.

5.2 Écart interquartile et KNN imputer

Afin de supprimer les données aberrantes et de commencer l'étape de prétraitement des données, nous créons une classe Preprocessor(), cette classe présente 8 fonctions qui nous aideront à nettoyer le jeu de données. La fonction replace_outliers_na() est une fonction que nous avons créés 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 7 : 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 la caractéristiques qui présente 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.

5.3 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. Nous séparons les données en base d'apprentissage et de test à l'aide des fonctions split_dataset() d'Analyzer() qui sépare la variable cible des autres variables et 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.

6 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.

6.1 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.

6.2 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 :

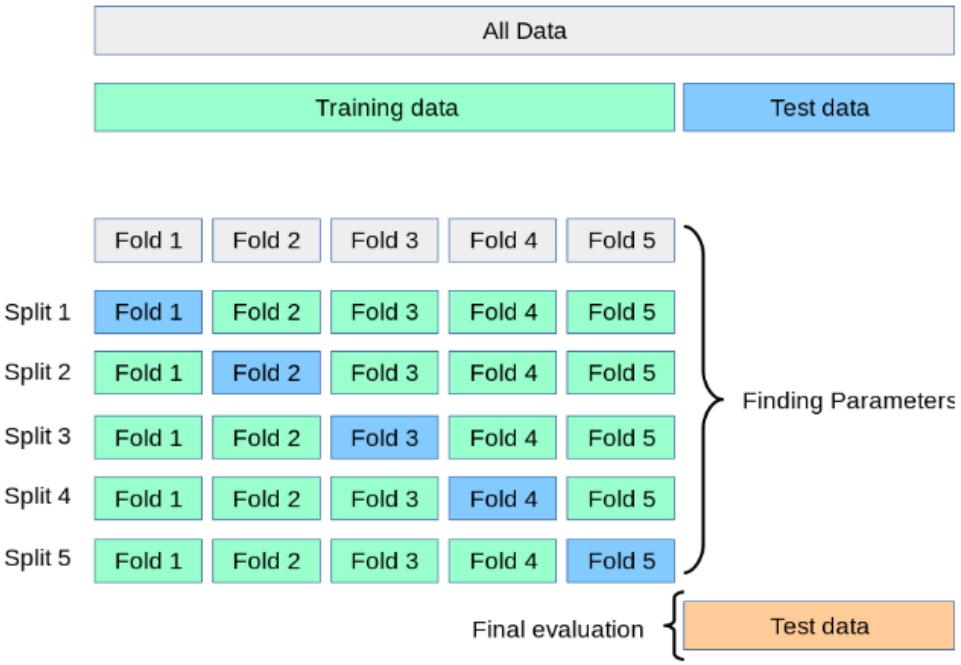

Figure 9 : Fonctionnement de la validation croisée

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.

6.2.1 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 tous ses fonctions associées.

7 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 multicouches
6. Classification par algorithme des K plus proches voisins

7.1 La régression logistique

La régression logistique est un algorithme d'apprentissage supervisé utilisé pour la classification. Comme vue en cours, c'est une amélioration du perceptron qui utilise une nouvelle fonction d'activation (la sigmoïde) ainsi qu'une nouvelle fonction de coût (l'entropie croisée).

Fonctionnement :

La fonction sigmoïde est utilisée pour modéliser la probabilité que la variable dépendante soit égale à 1 en fonction des variables indépendantes.
La fonction de coût mesure la différence entre les valeurs prédites par le modèle et les valeurs réelles observées. Les coefficients du modèle sont ajustés pendant la phase d'entraînement à l'aide d'une technique d'optimisation (généralement la descente de gradient) pour minimiser la fonction de coût.
Une fois le modèle entraîné, il peut être utilisé pour prédire la probabilité qu'une nouvelle observation appartienne à la classe 1.

Dans notre projet, nous définissons une classe LogisticRegressionModel() qui importe le classifieur LogisticRegression() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont notamment le terme inverse de régularisation C et l'algorithme à utiliser dans le problème d'optimisation car ce sont selon nous les critères les plus intéressants et importants à varier.

7.2 Machine à vecteur de support

Les machines à vecteurs de support sont une classe d'algorithmes d'apprentissage supervisé utilisés pour la classification et la régression.

Fonctionnement :

Le principe fondamental des SVM est de trouver un hyperplan qui sépare au mieux les données en différentes classes. Si les données ne sont pas linéairement séparables, les SVM peuvent les transformer dans un espace de plus grande dimension grâce à une fonction appelée le noyau (kernel), permettant ainsi de trouver un hyperplan de séparation.

Comme vue en cours, les SVM cherchent à maximiser la marge, qui est la distance entre l'hyperplan de séparation et les points les plus proches de chaque classe, appelés vecteurs de support. Ces vecteurs de support sont cruciaux car ils déterminent l'emplacement et l'orientation de l'hyperplan.

Une fois l'hyperplan trouvé, les SVM peuvent être utilisées pour classer de nouvelles données en fonction de leur position par rapport à cet hyperplan.

Dans notre projet, nous définissons une classe SVMModel() qui importe le classifieur SVC() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont notamment le terme inverse de régularisation C, le type de noyau Kernel et gamma le coefficient du noyau car ce sont selon nous les critères les plus intéressants et importants à varier.

7.3 Arbres de décision

Les arbres de décision sont des modèles d'apprentissage automatique utilisés pour résoudre des problèmes de classification et de régression.

Fonctionnement :

Les modèles d'arbres de décision prennent la forme d'une structure arborescente composée de nœuds, où chaque nœud représente une décision basée sur une caractéristique particulière. La décision de subdiviser ou non un nœud dépend d'une notion d'impureté. Si l'impureté est élevée alors on subdivise. Les feuilles de l'arbre contiennent les résultats de la classification ou de la prédiction.

Comme vues en cours, au fur et à mesure de l'apprentissage les arbres de décision classifient les nœuds en utilisant les modèles de types "stumps" comme des classifieurs perpendiculaires à un axe. En combinant ces modèles on obtient des frontières de décision "crénelées". Ce type de modèle mène souvent au surapprentissage des données d'entraînement (nous y reviendrons par la suite).

Dans notre projet, nous définissons une classe DecisionTreeModel() qui importe le classifieur DecisionTreeClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le critère d'impureté (gini ou entropie) et la profondeur maximale de l'arbre car ce sont selon nous les critères les plus intéressants et importants à varier.

7.4 Forêts aléatoires

Les forêts aléatoires sont une technique d'ensemble en apprentissage automatique qui combine plusieurs modèles d'arbres de décision pour améliorer la robustesse et la généralisation du modèle global. Comme énoncé précédemment, les arbres de décisions sont très souvent soumis au sur-apprentissage, les forêts aléatoires permettent de combiner les arbres de décisions à l'aide d'un vote majoritaire afin de réduire la variance des modèles.

Dans notre projet, nous définissons une classe RandomForestModel() qui importe le classifieur RandomForestClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le critère d'impureté (gini ou entropie), la profondeur maximale de l'arbre ainsi que le nombre d'arbres combinés pour effectuer la prédiction. Parmi les différents paramètres disponibles, nous avons choisis ces paramètres car ce sont selon nous les critères les plus intéressants et importants à varier.

7.5 Perceptron multicouches

Le perceptron multicouche est un type d'architecture de réseau de neurones artificiels. C'est une extension du perceptron simple, introduisant plusieurs couches de neurones, d'où le terme « multicouche ». Ces réseaux sont également souvent appelés réseaux de neurones profonds lorsqu'ils ont plusieurs couches cachées.

Fonctionnement :

Les neurones sont les unités fondamentales du réseau, représentant des entités qui effectuent des opérations mathématiques sur les entrées.

Comme vue en cours, chaque neurone est associé à une fonction d'activation qui détermine sa sortie en fonction des entrées.Un perceptron multicouche est composé d'au moins trois couches : une couche d'entrée, une ou plusieurs couches cachées, et une couche de sortie. La couche d'entrée reçoit les données d'entrée, la couche de sortie produit la sortie finale du réseau, et les couches cachées effectuent des transformations intermédiaires.
Chaque connexion entre les neurones est associée à un poids qui ajuste l'impact de l'entrée sur la sortie du neurone. Ces poids sont appris pendant la phase d'entraînement du réseau.

Les données d'entrée sont propagées à travers le réseau de la couche d'entrée à la couche de sortie. Chaque neurone calcule sa sortie en utilisant les entrées pondérées et la fonction d'activation. C'est la forward pass.

L'algorithme de rétropropagation est ensuite utilisé pour ajuster les poids du réseau en fonction de l'erreur entre les prédictions du réseau et les valeurs réelles. Il utilise la dérivée de l'erreur par rapport aux poids pour mettre à jour les poids de manière à minimiser cette erreur. C'est la backward pass.

Une fois les poids déterminés, le perceptron multicouche peut effectuer les prédictions.

Dans notre projet, nous définissons une classe MLPModel() qui importe le classifieur MLPClassifier() de scikit-learn. De nombreux hyper-paramètres peuvent être testés, spécialement pour ce modèle, ce qui entraîne un algorithme grid search assez long. C'est pour cela que nous avons établi que la fonction d'activation que nous garderons est la ReLU car nous avons vu dans le cours qu'elle fait partie des plus performantes (à l'inverse de la sigmoid ou tanh qui mène à la disparition du gradient). Une valeur de momentum égale à 0.9 est aussi conseillée dans le cours. Enfin, le solveur pour l'optimisation de la descente de gradient que nous gardons est adam, celui-ci est aussi conseillé dans le cours (combo entre le momentum et RMSprop). Ces choix nous amènent à ne tester que ces deux hyperparamètres qui nous semblaient cruciaux à faire varier : le terme de régularisation (alpha) et les différentes couches cachées du réseau (hidden_layers_sizes). Cela nous permet d'éviter des temps trop longs lors de l'algorithme grid search.

7.6 Classification par algorithme des K plus proches voisins

L'algorithme des plus proches voisins est une méthode d'apprentissage supervisé utilisée à la fois pour la classification et la régression.

Fonctionnement :

Comme explicité précédemment, cet algorithme repose sur le principe que les données similaires ont tendance à se regrouper dans l'espace. Il 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 classification.

Dans notre projet, nous définissons une classe KNNModel() qui importe le classifieur KNNClassifier() de scikit-learn. Les hyper paramètres testés par l'algorithme grid search sont le nombre de voisins pris en compte lors de l'algorithme ainsi que la fonction de mesure des poids :

- uniforme, dans ce cas tous les points de chaque quartier sont pondérés de manière égale.
- distance, dans ce cas les voisins les plus proches auront une plus grande influence que les voisins plus éloignés.

8 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.

8.1 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).

8.1.1 Score d'apprentissage, validation et de test

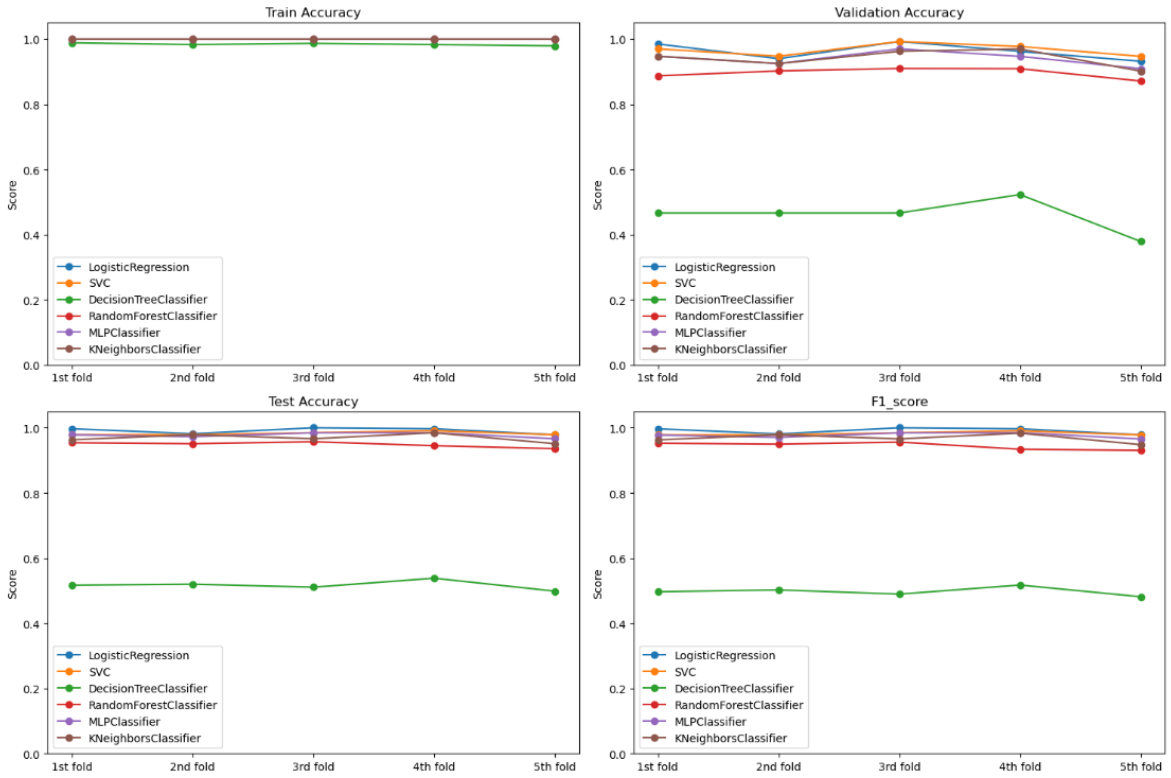

Figure 10 : Score des modèles

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. 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.

8.1.2 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.

8.1.3 Score moyen

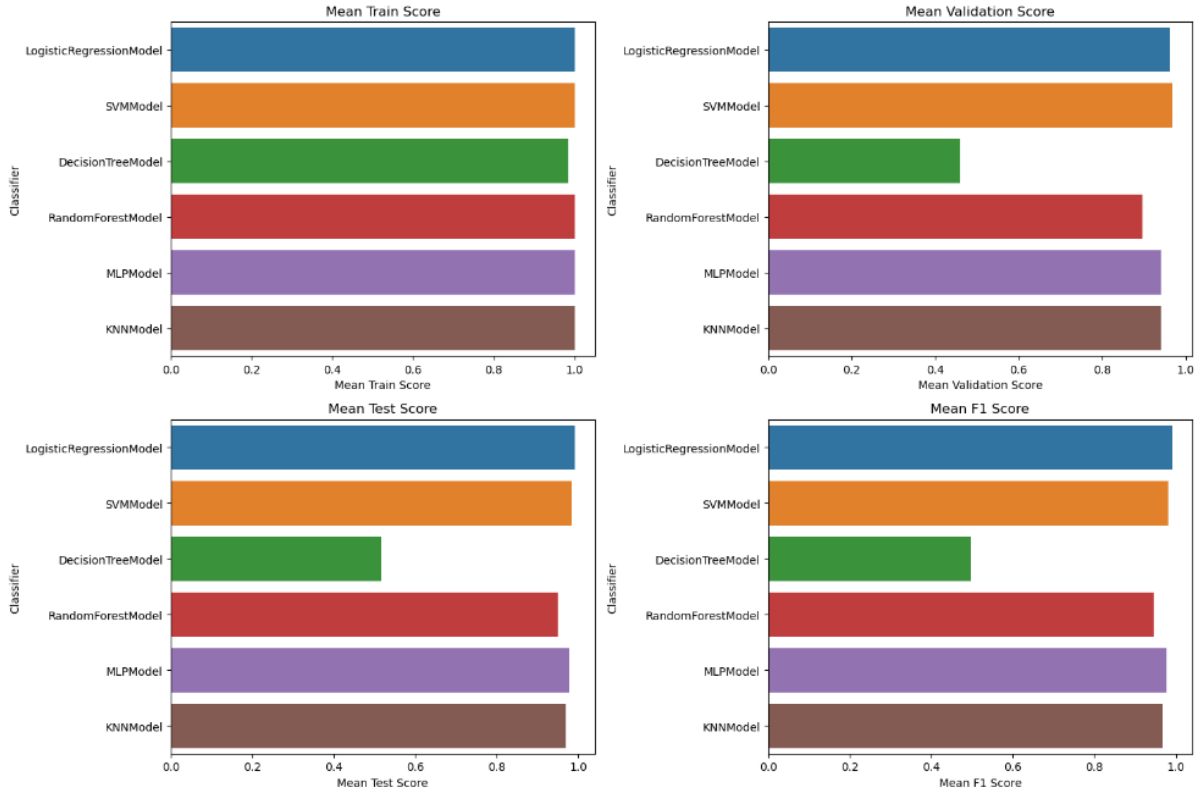

Figure 12 : Barplot représentant les scores moyens des modèles

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

8.2 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 (sanity checks).

8.2.1 Courbes d'apprentissage/Validation

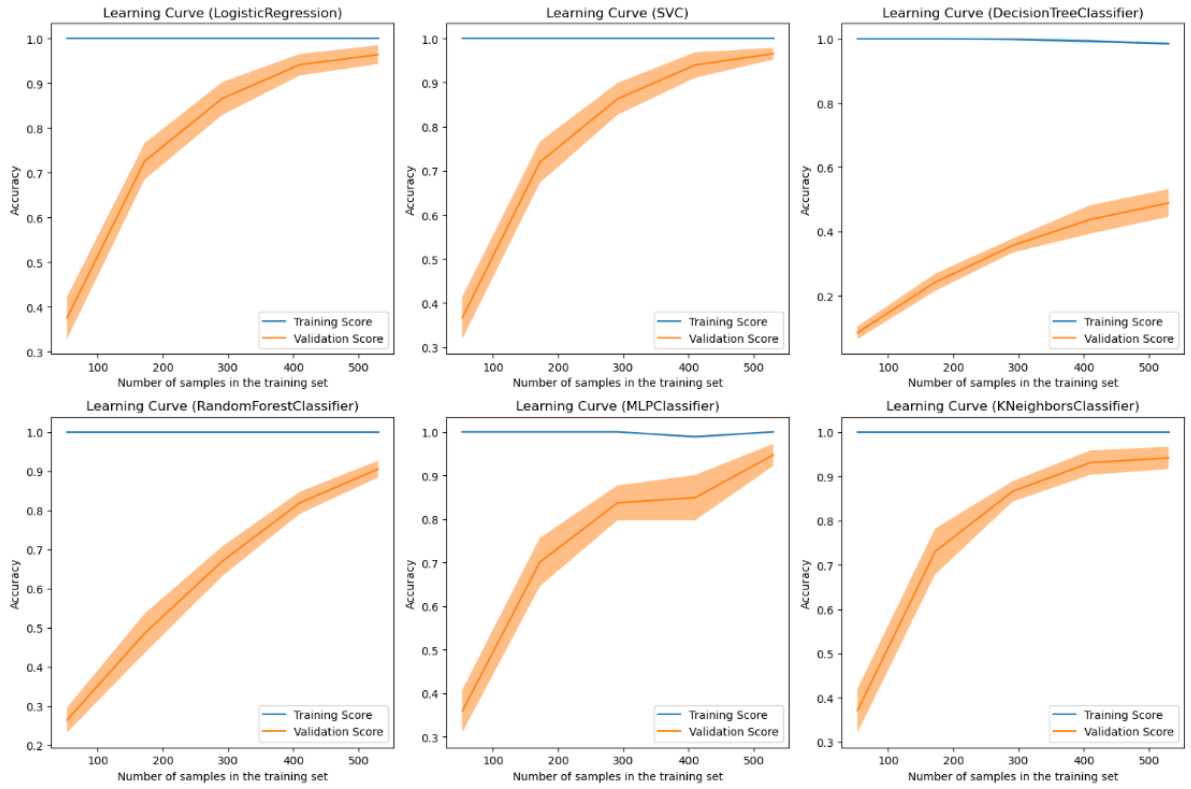

Figure 13 : Courbe d'apprentissage et de validation

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, si l'on rejoint nos notes de cours on le voit clairement : 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.

8.2.2 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.

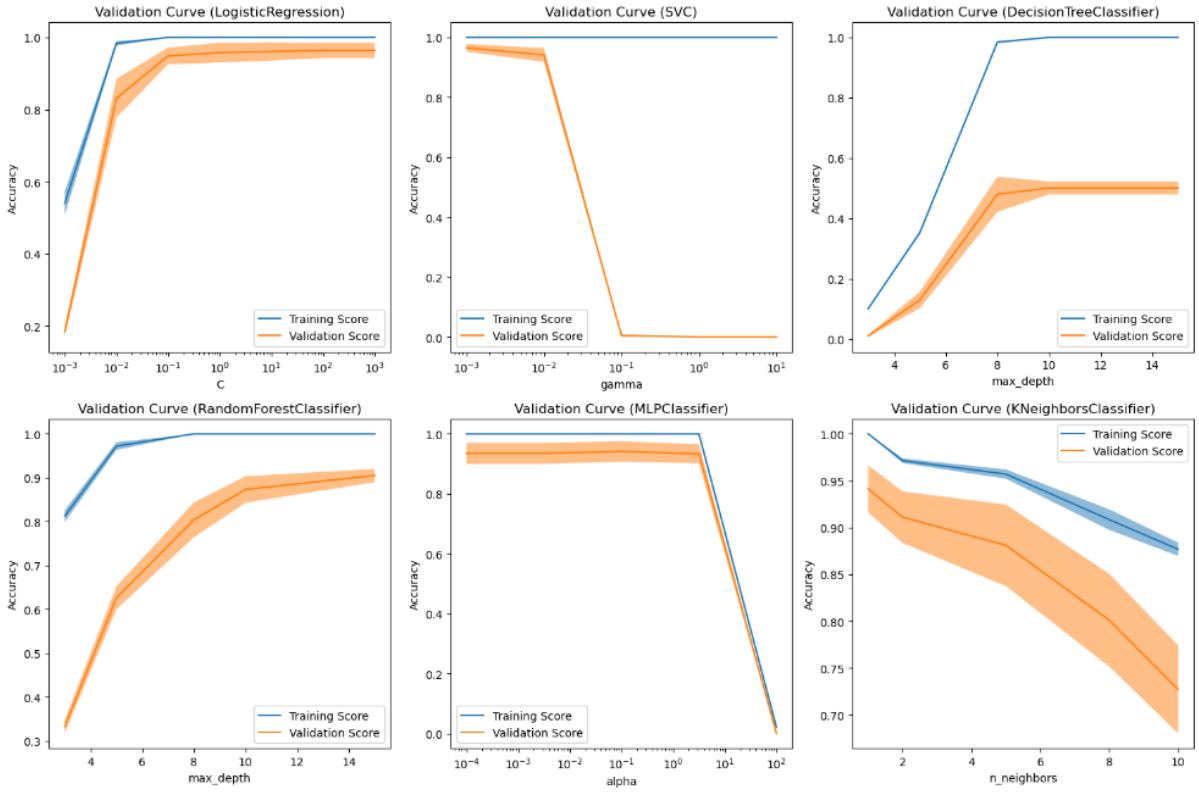

Figure 14 : Courbe d'apprentissage et de validation en fonction de la valeur de certains paramètres

L'ensemble de ces courbes sont très intéressantes car elles nous permettent de faire le lien avec ce que nous avons vu en cours.

<u>Régression logistique :</u>
Le terme inverse de régularisation C est celui-ci 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.

<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.

Perceptron multicouches :

En ce qui concerne le perceptron, on fait les observations inverses que pour la régression logistique. En effet, contrairement à la régression logistique l'hyper paramètre "alpha" que nous avons fait varier est le terme de régularisation L2 et non son inverse ainsi plus alpha augmente plus la régularisation est forte et donc plus le modèle est susceptible de moins surapprendre.

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

8.2.3 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.

8.2.3.1 Courbes Roc pour tous les labels

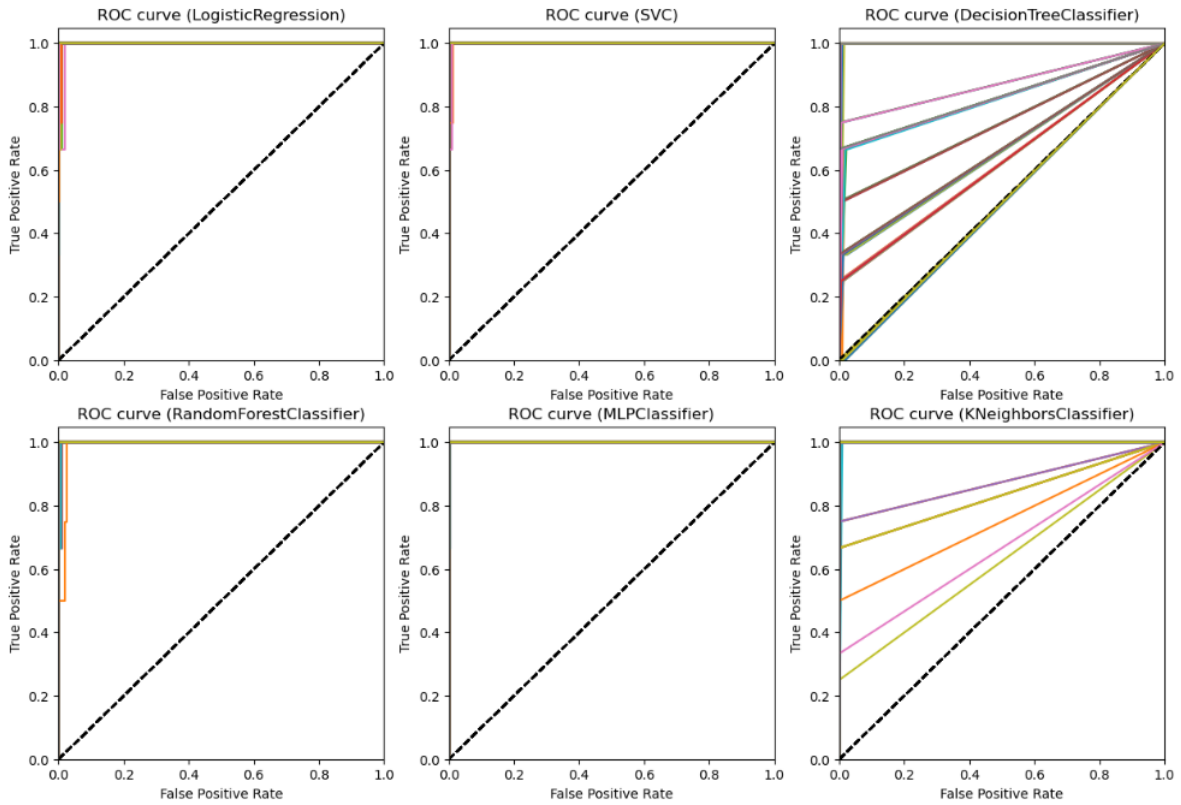

Figure 15 : Courbe roc pour chaque modèle pour toutes les classes de notre variable cible

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.

8.2.3.1 Courbes Roc moyenne

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

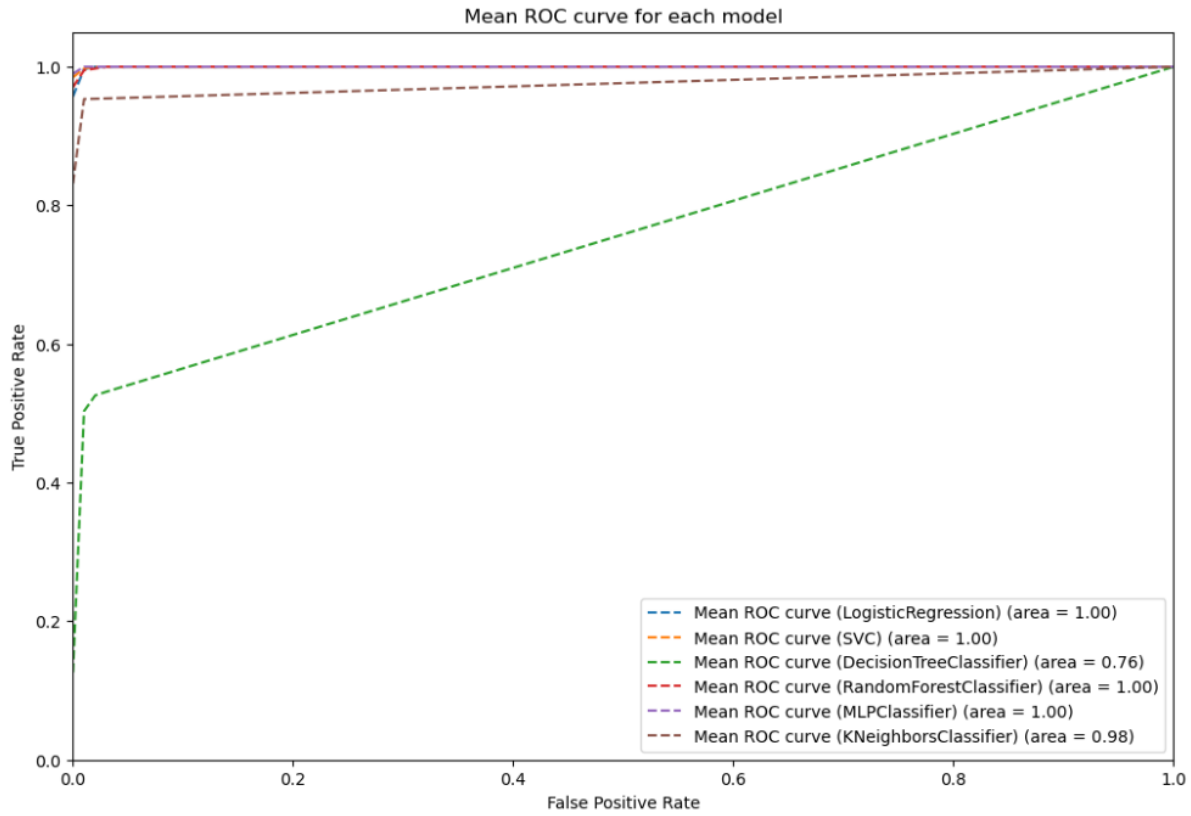

Figure 16 : Courbe roc moyenne des modèles

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 sur apprendre. 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. Il serait peut être intéressant de faire la même chose sur un jeu de données différent. Les différentes métriques utilisées tout au long du projet ont apporté leur lot de connaissances, couplé au cours. Nous sommes satisfaits de ce que nous avons produit.