**TECHNIQUES**

**D’APPRENTISSAGE**

**IFT712**

**Projet de session**

Rapport présenté à

M. Martin Vallières

Par :

**(Nom) (Matricule)**

Simon Giard-Leroux 12095680

Université de Sherbrooke

Automne 2020

Table des matières

[1. Code 1](#_Toc58695508)

[2. Choix de design 1](#_Toc58695509)

[3. Gestion de projet 1](#_Toc58695510)

[4. Démarche scientifique 1](#_Toc58695511)

[4.1 Données d’entrée 2](#_Toc58695512)

[4.2 Pré-traitement des données 2](#_Toc58695513)

[4.2.1 Méthode 1 : Données brutes 2](#_Toc58695514)

[4.2.2 Méthode 2 : Normalisation des données selon la moyenne et l’écart type 3](#_Toc58695515)

[4.2.3 Méthode 3 : Normalisation des données selon le maximum et le minimum 4](#_Toc58695516)

[4.2.4 Méthode 4 : Groupement des classes par genre (« genera ») 4](#_Toc58695517)

[4.2.5 Méthode 5 : Groupement par classes similaires en utilisant t-SNE 4](#_Toc58695518)

[4.3 Méthodes de classification choisies 6](#_Toc58695519)

[4.4 Recherche d’hyper-paramètres 6](#_Toc58695520)

[5. Analyse des résultats 7](#_Toc58695521)

Liste des tableaux

[Tableau 1 : Colonnes dans les fichiers de la base de données 2](#_Toc58695522)

[Tableau 2 : Espèces des dix premières données du fichier « train.csv » 4](#_Toc58695523)

[Tableau 3 : Méthodes de classification choisies 6](#_Toc58695524)

[Tableau 4 : Liste des hyper-paramètres testés 7](#_Toc58695525)

Liste des figures

[Figure 1 : Valeurs dans les données d’entrée pour chaque caractéristique (« feature ») 3](#_Toc58695526)

[Figure 2 : Résultat 2D de l’algorithme t-SNE sur les données d’entrée 5](#_Toc58695527)

[Figure 3 : Résultat 2D de l’algorithme t-SNE sur les données d’entrée (groupes de classes) 5](#_Toc58695528)

# 1. Code

Lien vers le répertoire GitHub du projet : <https://github.com/sgiardl/IFT712-Projet>

# 2. Choix de design

**Choix de design :** vous devez organiser votre code de façon professionnelle. Pour ce faire, on s’attend à une hiérarchie de classes cohérente, pas seulement une panoplie de fonctions disparates. Aussi, du code dans un script « qui fait tout » se verra automatiquement attribuer la note de zéro. Bien que non requis, on vous encourage à faire un design de classes avant de commencer à coder et à présenter un diagramme de classe dans votre rapport. Aussi, le code, les données et la documentation doivent être organisés suivant une bonne structure de répertoires. Pour vous aider, vous pouvez utiliser le projet « cookiecutter » ([**https://github.com/audreyr/cookiecutter**](https://github.com/audreyr/cookiecutter)). La solution proposée doit aussi être facile à utiliser. Bien que non requis, on vous encourage à présenter votre solution sous forme de jupyter notebook(s).

Le diagramme de classes ci-dessous montre les classes programmées dans le projet.

Figure  : Diagramme de classes

Description des classes

# 3. Gestion de projet

**Gestion de projet :** comme tout projet qui se respecte, vous devez utiliser un gestionnaire de version de code. On vous demande d’utiliser « git » via la plateforme « GitHub » (incluez votre lien dans votre rapport). On s’attend également à ce que vous fassiez une bonne utilisation de git. Par exemple : évitez de « pousser » du code dans le master sans merge, éviter les « méga » commits, etc. Bien que non requis, on vous encourage aussi à utiliser Trello pour gérer votre projet à haut niveau.

La plateforme en ligne Trello a été utilisée pour suivre les tâches du projet à haut niveau. La figure ci-dessous montre une capture d’écran de la planche principale du Trello.

Figure  : Planche principale du Trello

Lien vers le Trello du projet : <https://trello.com/b/qc2TNWm0/ift712-projet-de-session>

# 4. Démarche scientifique

**Démarche scientifique :** pour ce volet, vous devez vous poser les questions suivantes : avez-vous bien « cross-validé » vos méthodes? Avez-vous bien fait votre recherche d’hyper-paramètres? Avez-vous entraîné et testé vos méthodes sur les mêmes données? Est-ce que cela transparaît dans le rapport? Avez-vous uniquement utilisé les données brutes ou avez-vous essayé de les réorganiser pour améliorer vos résultats? Etc.

## 4.1 Données d’entrée

La base de données Kaggle choisie pour le projet est celle de classification de feuilles d’arbres proposée dans les instructions du projet sur Moodle et disponible au lien suivant : <http://www.kaggle.com/c/leaf-classification>. La base de données est distribuée en deux fichiers de données en format .csv : « *train.csv*» de 990 données et « *test.csv*» de 594 données. Le tableau ci-dessous montre les colonnes disponibles dans les deux fichiers :

|  |  |  |
| --- | --- | --- |
| ***train.csv*** | ***test.csv*** | ***Type*** |
| id | id | Nombre entier séquentiel |
| species |  | Chaîne de caractères |
| margin1 @ margin64 | margin1 @ margin64 | Nombre décimal |
| shape1 @ shape64 | shape1 @ shape64 | Nombre décimal |
| texture1 @ texture64 | texture1 @ texture64 | Nombre décimal |

Tableau 1 : Colonnes dans les fichiers de la base de données

Le champ « *id* » correspond à un nombre entier séquentiel qui indique le numéro de la donnée, le champ « *species* » à une chaîne de caractères indiquant le nom de l’espèce d’arbre et les champs « *margin* », « *shape* » et « *texture* » correspondent à des caractéristiques (ou « *features* ») extraites d’images des feuilles d’arbre en nombre décimaux. Il existe 99 classes différentes d’espèces d’arbres dans la base de données.

Puisque le fichier « *test.csv* » ne contient pas de colonne « *species* », les données à l’intérieur ne peuvent pas être utilisées pour calculer la justesse des méthodes de classification, car leur vérité terrain n’est pas connue et ne peut pas être comparée à la prédiction des méthodes de classification. Ainsi, ce fichier n’a pas été utilisé au cours du projet.

L’ensemble de données de test a plutôt été généré à partir d’un sous-ensemble correspondant à 20 % des données du fichier « *train.csv* » et l’ensemble des données d’entraînement a été généré à partir du reste, soit 80 % des données. Donc, l’ensemble d’entraînement contient 792 données et l’ensemble de test contient 198 données.

La méthode « *StratifiedKFold* » de la librairie « *sklearn*» a été implémentée afin de séparer d’une façon aléatoire et stratifiée les données entre l’ensemble d’entraînement et de test pour s’assurer d’avoir une représentation réaliste et proportionnelle de toutes les classes dans les deux ensembles de données. Le paramètre « *n\_splits* » de la méthode « *StratifiedKFold* » indique le nombre d’itérations de mélange aléatoire et de stratification. Ce paramètre a été spécifié à 5, soit l’inverse du pourcentage de 20 % de l’ensemble de test, pour s’assurer d’avoir une mesure de justesse moyenne sur des ensembles différents d’entraînement et de test afin d’éliminer la chance d’avoir un mélange aléatoire « chanceux » où la justesse serait plus élevée que pour un autre mélange aléatoire. De plus, l’utilisation de cette méthode « *K-Fold* » assure que chaque donnée sera utilisée à la fois dans l’ensemble d’entraînement et dans l’ensemble de test au moins une fois. La justesse d’entraînement et de test moyenne sur les 5 itérations est présentée dans les résultats.

## 4.2 Pré-traitement des données

### 4.2.1 Méthode 1 : Données brutes

Les méthodes de classification ont été testées sur les données brutes, mais certaines méthodes de pré-traitement des données ont également été testées afin d’observer si elles permettaient d’obtenir des meilleures performances.

### 4.2.2 Méthode 2 : Normalisation des données selon la moyenne et l’écart type

En traçant un graphique des valeurs contenues dans la base de données d’entrée pour chaque caractéristique (« *feature* »), il est possible de constater une grande variance des valeurs possibles pour chaque caractéristique. En effet, sur le graphique ci-dessous, la plage de valeurs possibles pour les caractéristiques « *margin* » et « *texture* » est beaucoup plus grande que pour les caractéristiques « *shape* ».

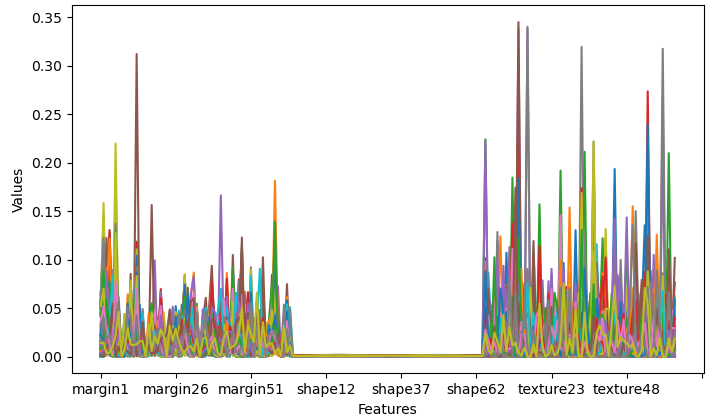


Figure 1 : Valeurs dans les données d’entrée pour chaque caractéristique (« feature »)

En appliquant une normalisation sur les données d’entrée, il est ainsi attendu que l’information contenue dans les caractéristiques « *shape* » puisse être plus exprimée que sans normalisation, puisque les échelles de départ ne sont pas concordantes entre chaque caractéristique.

La première méthode de normalisation utilisée est celle utilisant la moyenne et l’écart-type avec la formule ci-dessous.

Les valeurs de moyennes et d’écart-types pour chaque caractéristique sont extraites des données d’entraînement seulement. Ainsi, la normalisation est effectuée sur les données de d’entraînement et de test en utilisant seulement les moyennes et écart-types calculées avec les données d’entraînement.

### 4.2.3 Méthode 3 : Normalisation des données selon le maximum et le minimum

La deuxième méthode de normalisation utilisée est celle utilisant le maximum et le minimum avec la formule ci-dessous.

Les valeurs de maximum et minimum pour chaque caractéristique sont extraites des données d’entraînement seulement. Ainsi, la normalisation est effectuée sur les données de d’entraînement et de test en utilisant seulement les maximums et minimums calculés avec les données d’entraînement.

### 4.2.4 Méthode 4 : Groupement des classes par genre (« genera »)

Le nom des espèces d’arbres contenues dans le champ « *species* » de la base de données sont sous-divisés en deux parties séparées par un symbole \_. Par exemple, voici une liste des espèces des dix premières données dans le fichier « *train.csv* » :

|  |  |
| --- | --- |
| ***id*** | ***species*** |
| 1 | Acer\_Opalus |
| 2 | Pterocarya\_Stenoptera |
| 3 | Quercus\_Hartwissiana |
| 5 | Tilia\_Tomentosa |
| 6 | Quercus\_Variabilis |
| 8 | Magnolia\_Salicifolia |
| 10 | Quercus\_Canariensis |
| 11 | Quercus\_Rubra |
| 14 | Quercus\_Brantii |
| 15 | Salix\_Fragilis |

Tableau 2 : Espèces des dix premières données du fichier « train.csv »

La première partie de l’espèce correspond à son nom de genre (« *genera* »), tandis que la deuxième partie correspond à son épithète d’espèce. Il est attendu que deux espèces qui ont un nom de genre en commun ont un lien évolutionnaire plus rapproché que deux espèces avec des noms de genres différents, et ainsi des caractéristiques plus similaires. Un regroupement par nom de genre a ainsi été fait en enlevant le premier \_ ainsi que tous les caractères après le premier \_ trouvé dans chaque chaîne de caractère du champ « *species* » à partir de la gauche.

### 4.2.5 Méthode 5 : Groupement par classes similaires en utilisant t-SNE

L’algorithme t-SNE (« *t-distributed stochastic neighbor embedding* ») est une technique de réduction de dimensionnalité qui permet de visualiser des données dans un espace dimensionnel 2D ou 3D. Les données d’entrée ont 3 \* 64 = 192 dimensions de caractéristiques et l’algorithme t-SNE a été utilisé avec la classe « *TSNE* » de la librairie « *sklearn* » pour réduire ce nombre dimensions à deux. Le graphique ci-dessous montre le résultat de l’algorithme t-SNE, chaque symbole et couleur correspond à une classe d’espèce d’arbre différente.

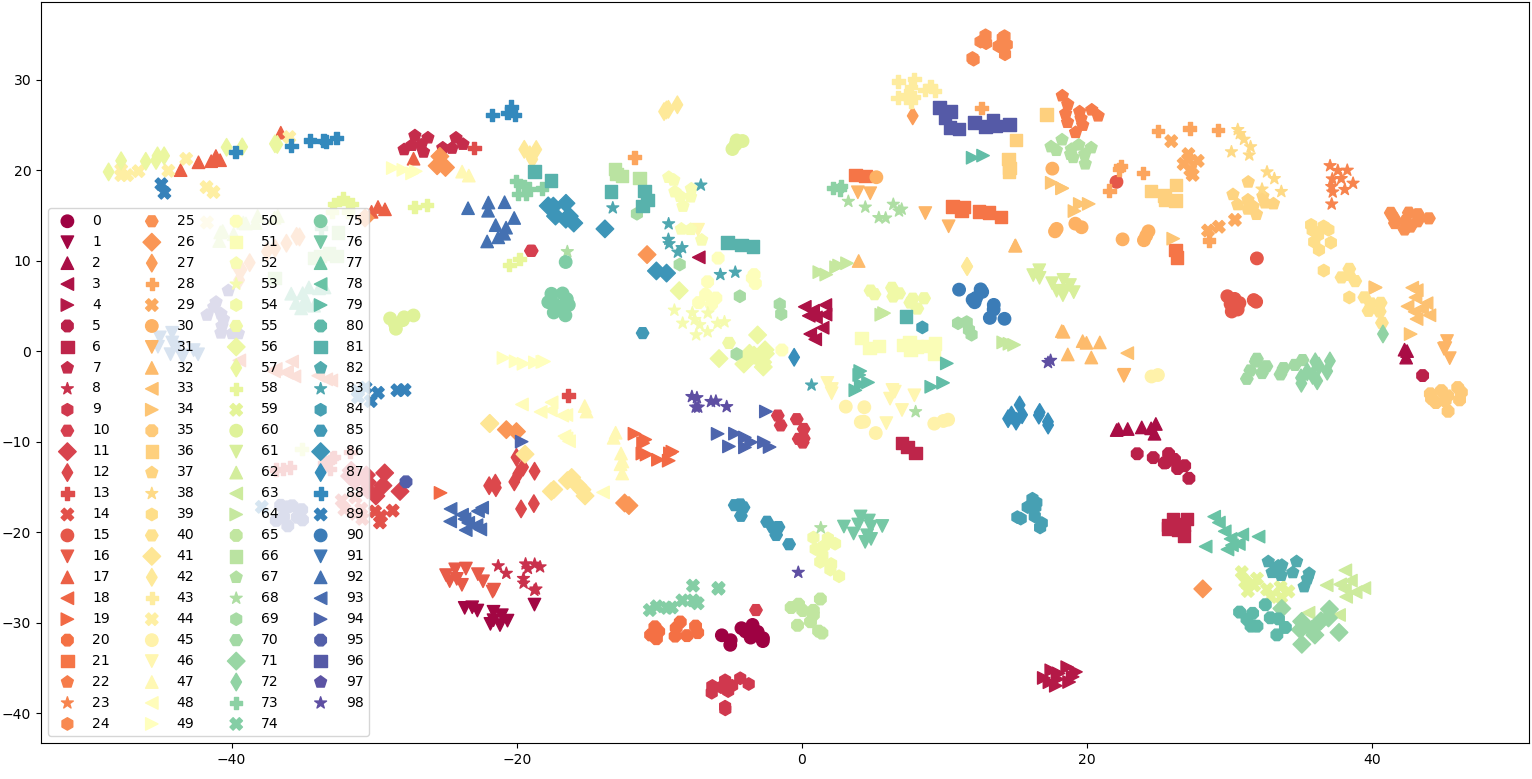


Figure 2 : Résultat 2D de l’algorithme t-SNE sur les données d’entrée

Par la suite, des groupes de classes ont été formés basées sur une analyse visuelle des données. Une importance a été attribuée au maintien de la balance entre les classes. Un algorithme de type « *k-means clustering* » n’a pas été utilisé pour former ces groupes, puisque la méthode de classification « *k-nearest neighbors*» compte déjà parmi les méthodes de classifications testées. Le graphique ci-dessous montre les groupes formés, qui représentent les classes mises en commun. Ainsi, il est possible de réduire le nombre de classes de 99 initialement à 29.

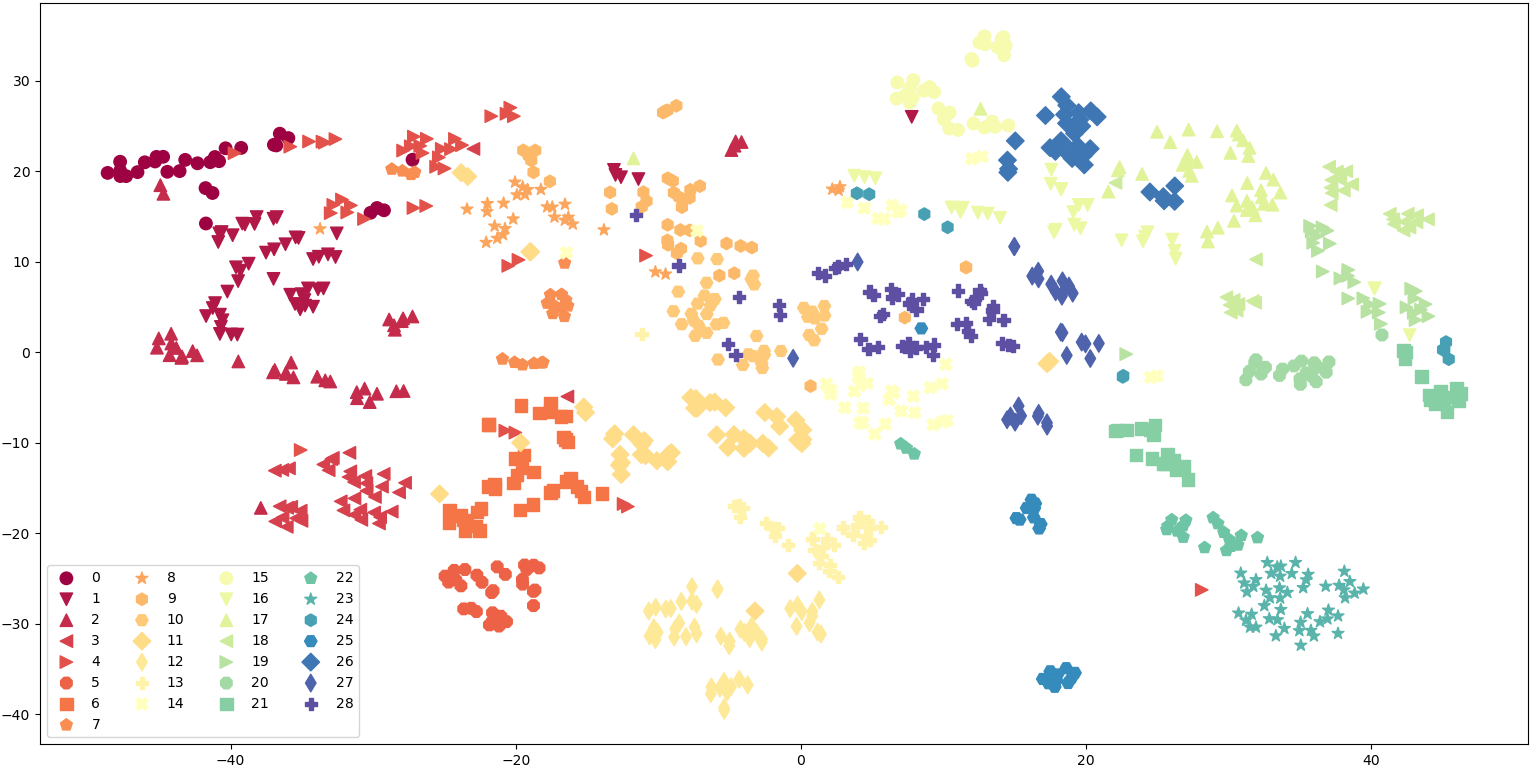


Figure 3 : Résultat 2D de l’algorithme t-SNE sur les données d’entrée (classes groupées)

## 4.3 Méthodes de classification choisies

Six méthodes de classification ont été choisies arbitrairement pour le projet. Le tableau ci-dessous montre les méthodes de classification choisies.

|  |  |  |
| --- | --- | --- |
| **Méthode** | **Classe dans « *sklearn* »** | **Classe créée dans le projet** |
| Ridge | RidgeClassifier | Ridge |
| Machine à vecteur de support | SVC | SupportVectorMachine |
| K-Plus proches voisins | KNeighborsClassifier | KNearestNeighbors |
| Perceptron multicouche | MLPClassifier | MultiLayerPerceptron |
| Forêt aléatoire | RandomForestClassifier | RandomForest |
| Naïve bayésienne | GaussianNB | NaiveBayes |

Tableau 3 : Méthodes de classification choisies

## 4.4 Recherche d’hyper-paramètres

La recherche d’hyper-paramètres a été faite à l’aide de la méthode « *GridSearchCV* » dans la librairie « *sklearn* » qui implémente un algorithme de validation croisée. Le paramètre « *cv* » dans la méthode d’initialisation de cette classe sert à indiquer le nombre de validations croisées à effectuer pour chaque combinaison possible d’hyper-paramètres spécifiées dans le dictionnaire « *param\_grid* » en entrée dans la méthode d’initialisation de cette classe. Ainsi, le choix de la taille de l’ensemble de validation est en fait l’inverse du paramètre « *cv* ». Dans le projet, chaque ensemble de validation est formé de 20 % des données de l’ensemble d’entraînement, donc le paramètre « *cv* » utilisé est 5, pour que 5 validations croisées soient effectuées pour chaque combinaison possible d’hyper-paramètres.

Le tableau ci-dessous montre les hyperparamètres donnés en entrée dans le paramètre « *param\_grid* » pour chaque méthode de classification testée. Les valeurs à tester pour chaque hyper-paramètre ont été sélectionnées afin de représenter un éventail réaliste et large des différentes choix disponibles.

|  |  |  |
| --- | --- | --- |
| **Méthode** | **Hyper-paramètre** | **Valeurs testées** |
| Ridge | alpha | 1e-1, 2e-1, 1e-2, 1e-3, 1e-4 |
| Machine à vecteur de support | C | 1e3, 5e3, 1e4, 5e4, 1e5 |
| gamma | 0.0001, 0.0005, 0.001, 0.005, 0.01, 0.1 |
| kernel | 'linear', 'poly', 'rbf', 'sigmoid' |
| K-Plus proches voisins | n\_neighbors | 1, 2, 3, 4, 5 |
| weights | 'uniform', 'distance' |
| algorithm | 'ball\_tree', 'kd\_tree', 'brute', 'auto' |
| leaf\_size | 10, 20, 30, 40, 50 |
| p | 1, 2 |
| Perceptron multicouche | hidden\_layer\_sizes | (50,), (80,), (100,) |
| learning\_rate\_init | 1e-1, 1e-2, 1e-3 |
| solver | 'adam', 'sgd' |
| activation | 'relu', 'logistic' |
| Forêt aléatoire | n\_estimators | 350, 400, 450 |
| max\_depth | 20, 25, 30, 35 |
| Naïve bayésienne | var\_smoothing | 1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-4, 1e-3 |

Tableau 4 : Liste des hyper-paramètres testés

# 5. Analyse des résultats

**Analyse des résultats :** que vos résultats soient bons ou non, vous devez en produire une analyse cohérente, potentiellement en les comparant aux résultats de tests déjà présents en ligne sur le site du challenge.

Les tableaux ci-dessous montrent les justesses moyennes d’entraînement et de test pour chaque méthode de classification ainsi que chaque méthode de pré-traitement.

|  | **Méthode de pré-traitement (voir sect. 4.2)** | | | | |
| --- | --- | --- | --- | --- | --- |
| **Méthode de classification** | **1** | **2** | **3** | **4** | **5** |
| Ridge |  |  |  |  |  |
| Machine à vecteur de support |  |  |  |  |  |
| K-Plus proches voisins |  |  |  |  |  |
| Perceptron multicouche |  |  |  |  |  |
| Forêt aléatoire |  |  |  |  |  |
| Naïve bayésienne |  |  |  |  |  |

Tableau 5 : Justesse moyenne d’entraînement

|  | **Méthode de pré-traitement (voir sect. 4.2)** | | | | |
| --- | --- | --- | --- | --- | --- |
| **Méthode de classification** | **1** | **2** | **3** | **4** | **5** |
| Ridge |  |  |  |  |  |
| Machine à vecteur de support |  |  |  |  |  |
| K-Plus proches voisins |  |  |  |  |  |
| Perceptron multicouche |  |  |  |  |  |
| Forêt aléatoire |  |  |  |  |  |
| Naïve bayésienne |  |  |  |  |  |

Tableau 5 : Justesse moyenne de test

Pour chaque méthode de pré-traitement des données et de classification, la justesse d’entraînement et de test ont été comparées à l’aide de diagrammes à bandes. Les diagrammes à bande générés sont montrés ci-dessous.

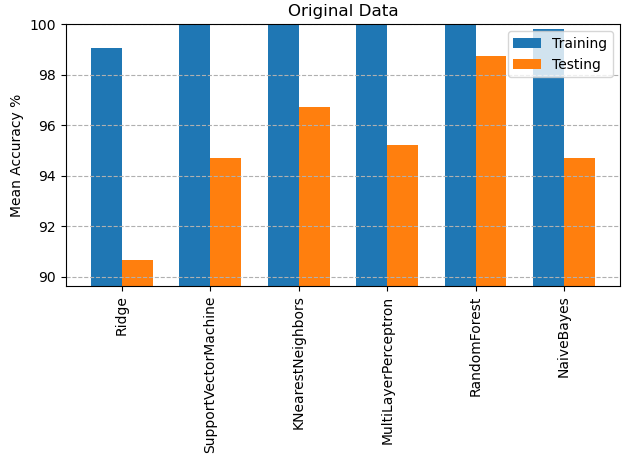


Figure  : Diagramme à bandes – Méthode 1 de pré-traitement

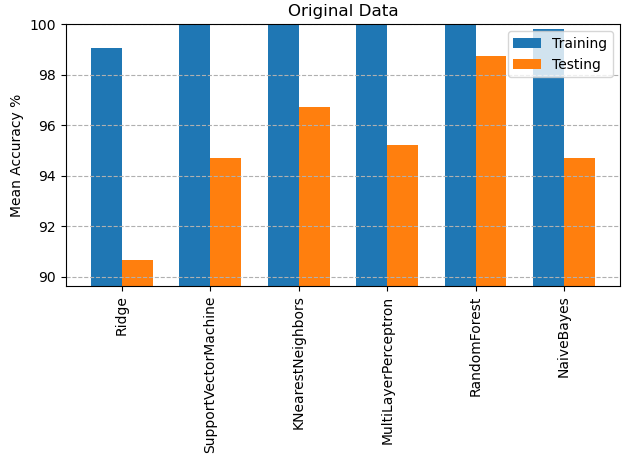


Figure  : Diagramme à bandes – Méthode 2 de pré-traitement

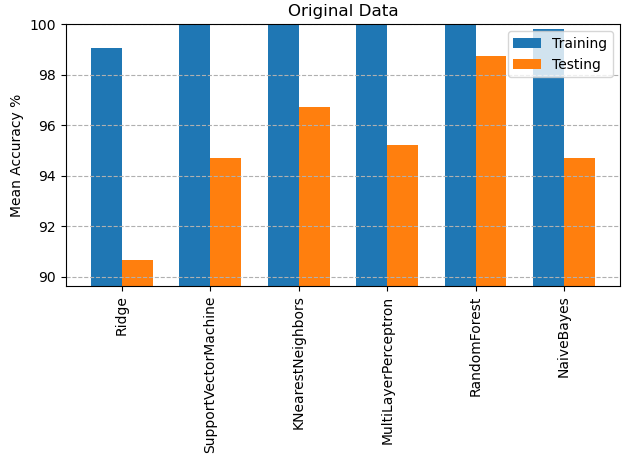


Figure  : Diagramme à bandes – Méthode 3 de pré-traitement

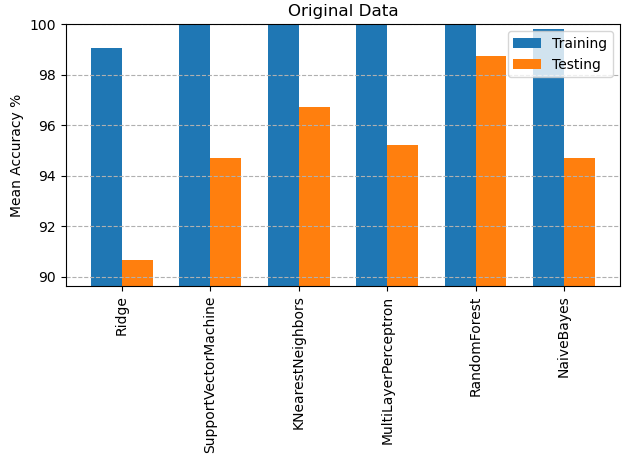


Figure  : Diagramme à bandes – Méthode 4 de pré-traitement

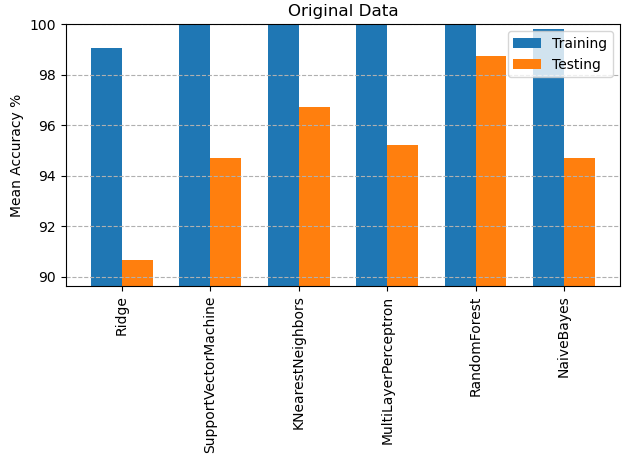


Figure  : Diagramme à bandes – Méthode 5 de pré-traitement

Les tableaux et graphiques ci-dessus montrent que la méthode de pré-traitement la plus efficace est X et que la méthode de classification la plus efficace est Y.

En se comparant aux résultats montrés sur le « leaderboard » du site Kaggle de la base de données (<https://www.kaggle.com/c/leaf-classification/leaderboard>), il est possible de voir que plusieurs personnes ont obtenu des scores de perte multiclasse de 0, ce qui correspond à une justesse de classification de 100 %. Les résultats obtenus durant ce projet se rapprochent ainsi des résultats obtenus par les autres personnes ayant participé à cette compétition de classification sur Kaggle.