UNIVERSITÉ DE SHERBROOKE

IFT712 – Techniques d'apprentissage

Projet de Session : Classification de Feuilles d'Arbres

Équipe :
Antoine Debin (deba0971)
Antoine Sabatier (saba5483)
Gabriel Généreux (geng0394)

Lien GitHub : `https://github.com/AntoineDbn/Leaf-Classification-`

15 décembre 2025

Table des matières

1 Introduction

Dans le cadre du cours IFT712, ce projet vise à concevoir et comparer des systèmes de classification automatique pour identifier des espèces végétales. Nous nous appuyons sur le jeu de données *Leaf Classification* issu de la plateforme Kaggle.

L'objectif principal est de mettre en œuvre une démarche scientifique pour comparer six algorithmes d'apprentissage supervisé (KNN, SVM, Random Forest, MLP, Gradient Boosting et Régression Logistique). Une attention particulière est portée à la méthodologie de validation pour éviter le sur-apprentissage, un risque majeur compte tenu du faible nombre d'exemples par classe (environ 10 images pour 99 espèces).

2 Description des Données

Le jeu de données est constitué de feuilles préalablement traitées. Contrairement à une approche de vision par ordinateur brute (CNN sur pixels), nous travaillons sur des données tabulaires.

- **Volume :** 990 échantillons d'entraînement.
- **Classes :** 99 espèces d'arbres différentes.
- **Caractéristiques (Features) :** 192 variables numériques extraites mathématiquement décrivant :
  - La Forme (*Shape*)
  - La Marge / le contour (*Margin*)
  - La Texture (*Texture*)

3 Méthodologie

Cette section détaille le protocole expérimental mis en place pour garantir la fiabilité des résultats.

3.1 Protocole de Validation (Hold-out & Cross-Validation)

Pour respecter les bonnes pratiques scientifiques et éviter toute *data leakage* :

1. **Séparation Train / Test :** Avant tout traitement, nous avons isolé **20%** des données (198 échantillons) pour constituer un ensemble de test final (Hold-out set). Ce jeu de données n'a jamais été vu par les modèles durant la phase d'entraînement.
2. **Validation Croisée Stratifiée :** Sur les 80% restants, nous avons utilisé une *Stratified K-Fold Cross-Validation* (k=5). L'aspect stratifié est crucial ici pour assurer que chaque pli de validation contienne une proportion représentative des 99 classes.

3.2 Prétraitement (Preprocessing)

Nous avons encapsulé le prétraitement dans un `Pipeline` Scikit-Learn.

- **StandardScaler :** Toutes les données ont été normalisées (moyenne = 0, variance = 1). Cette étape est indispensable pour la convergence du MLP et pour le calcul de distance du KNN et du SVM.
- **LabelEncoder :** Encodage des noms d'espèces en entiers.

3.3 Recherche d'Hyper-paramètres

Chaque modèle a été optimisé via un `GridSearchCV` testant diverses combinaisons (ex : noyau *rbf* vs *linear* pour le SVM, nombre de neurones pour le MLP).

4 Architecture et Design Logiciel

Le projet respecte une structure modulaire professionnelle sous forme de package Python, séparant clairement le code, les données et les tests.

- `src/data_loader.py` : classe responsable du chargement robuste des fichiers CSV et de la préparation des données.
- `src/model_trainer.py` : classe gérant l'entraînement des modèles, la validation croisée et la recherche d'hyper-paramètres à l'aide de pipelines scikit-learn.
- `src/models.py` : module utilitaire centralisant la définition des modèles et de leurs grilles d'hyper-paramètres sous forme de configurations.

Cette organisation favorise une forte extensibilité : l'ajout d'un nouveau modèle se fait exclusivement dans le module de configuration, sans modification de la logique d'entraînement ni du reste du code.

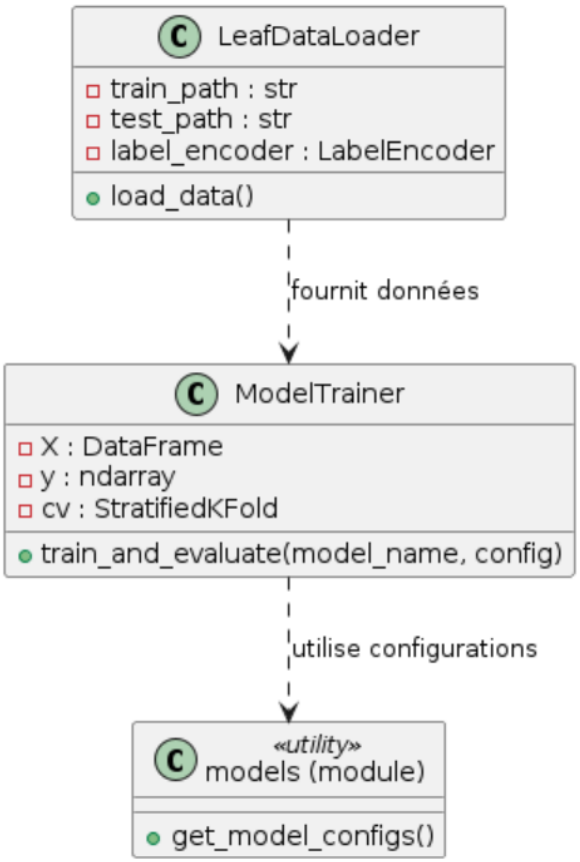

FIGURE 1 – Diagramme de classes UML de la solution implémentée

La Figure 1 présente le diagramme UML correspondant à l'architecture logicielle du projet. Afin d'éviter toute ambiguïté, les choix de modélisation suivants sont précisés. Le diagramme UML présente uniquement les classes métier réellement définies dans le projet. Les composants internes de scikit-learn (Pipeline, GridSearchCV, classifieurs) ne

sont volontairement pas représentés, car ils relèvent de détails d'implémentation et non du design logiciel propre au projet. Le module models est traité comme un fournisseur de configurations (factory fonctionnelle) et non comme une classe instanciable

5 Résultats et Analyse

5.1 Comparaison des Performances

Le tableau ci-dessous résume les performances obtenues après optimisation. Le **Score Test** est la métrique la plus importante car calculée sur des données inconnues.

Modèle	Score CV (Train)	Score Test (Réel)	Meilleurs Paramètr
MLP (Réseau de Neurones)	0.977	**0.995**	tanh, (100,)
Régression Logistique	0.986	0.990	C=1, solver=lbfgs
SVM	0.983	0.990	Kernel Linear, C=0.
Random Forest	0.980	0.985	100 arbres
KNN	0.968	0.970	k=5, Manhattan
Gradient Boosting	0.607	0.611	lr=0.1, depth=3

TABLE 1 – Résultats finaux (Classés par performance sur le Test Set)

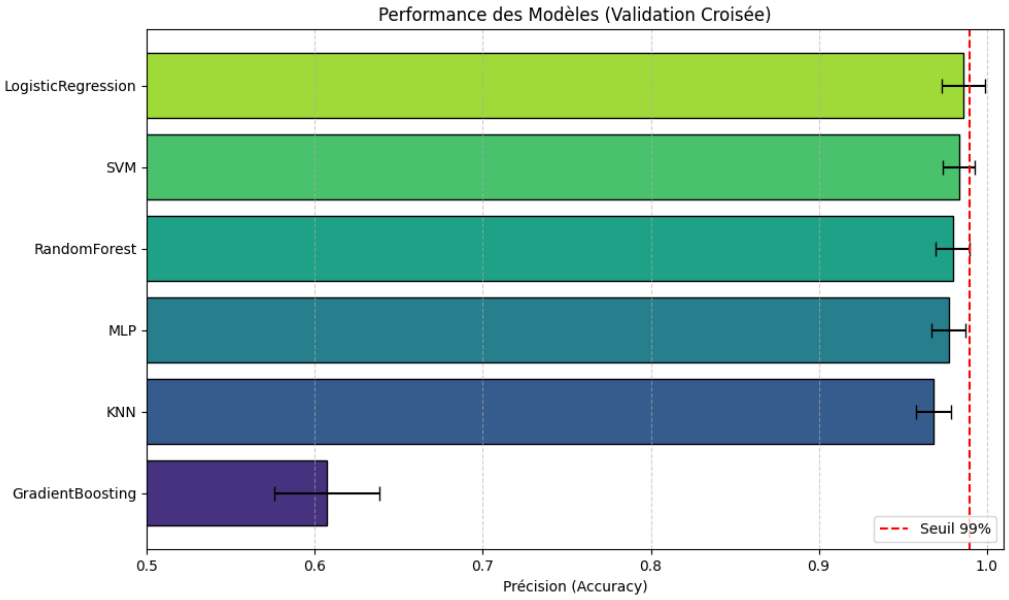

FIGURE 2 – Comparaison des performances (Accuracy moyen + Écart-type)

5.2 Analyse Critique

1. **La domination du MLP (99.5%) :** Le Perceptron Multicouche s'impose comme le meilleur modèle. Avec une seule couche cachée de 100 neurones et une fonction d'activation *tanh*, il a su capturer parfaitement les subtilités géométriques des feuilles.

2. **Linéarité du problème :** Les excellents scores du SVM (noyau linéaire) et de la Régression Logistique (99%) indiquent que les classes sont **linéairement séparables** dans l'espace des 192 caractéristiques. Les features extraites (Marge, Forme, Texture) sont donc de très haute qualité.
3. **L'échec du Gradient Boosting (61%) :** Ce résultat, bien qu'inférieur, est très instructif. Le Boosting est un algorithme complexe qui requiert un grand volume de données pour généraliser. Avec seulement ≈ 8 images par classe dans le set d'entraînement, le modèle a souffert d'un sur-apprentissage massif, n'arrivant pas à converger vers une solution généralisable.

5.3 Analyse des Erreurs (Matrice de Confusion)

Étant donné la précision exceptionnelle du modèle (99.5%), afficher une matrice complète de 99x99 classes serait illisible. La figure ci-dessous présente une **matrice filtrée**, ne montrant que les rares classes ayant généré des erreurs.

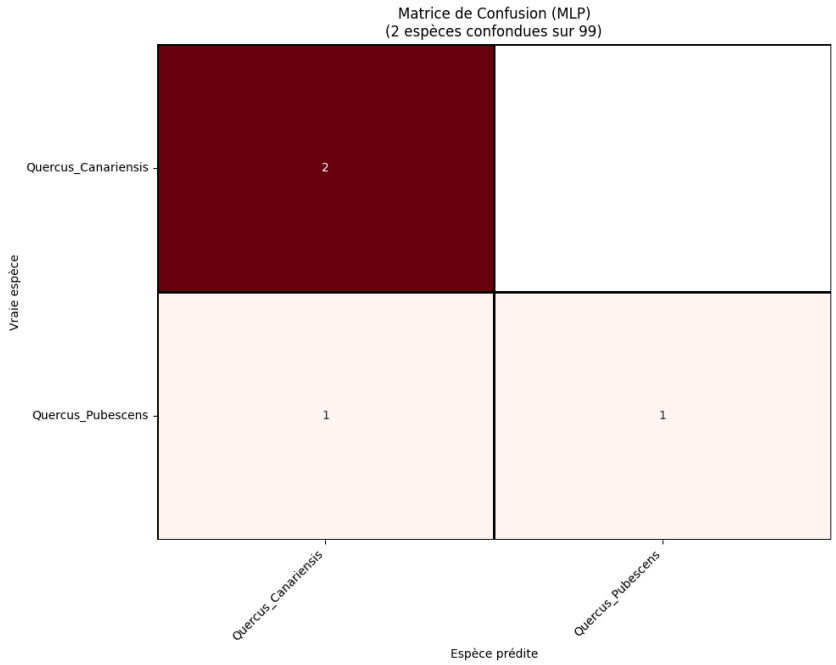

FIGURE 3 – Matrice de Confusion (Zoom sur les erreurs uniquement)

La matrice révèle une confusion spécifique entre **Quercus Pubescens** et **Quercus Canariensis**. Cette erreur est cohérente d'un point de vue botanique, ces deux espèces appartenant au même genre (les Chênes) et partageant des caractéristiques morphologiques très proches (forme lobée des feuilles). Le modèle a néanmoins correctement identifié les 97 autres espèces sans aucune erreur.

6 Gestion de Projet

Le développement a suivi les bonnes pratiques logicielles :
— Utilisation de **Git** et **GitHub** pour le versionnage.
— Flux de travail basé sur les branches (*Feature Branch Workflow*).

— Exclusion des fichiers de données et binaires via `.gitignore`.
— Reproductibilité assurée par un fichier `requirements.txt`.

7 Conclusion

Ce projet a permis de développer un classifieur de feuilles d'arbres atteignant une précision de **99.5%** sur des données inédites. L'analyse comparative a mis en évidence que pour ce type de données tabulaires à haute dimensionnalité mais faible volumétrie, les approches comme le MLP ou le SVM sont nettement supérieures aux méthodes d'ensemble complexes comme le Gradient Boosting. La structure modulaire du code et la rigueur du protocole de validation garantissent la fiabilité scientifique de ces conclusions.

8 Sources

- Cours de machine learning de l'Isep. - TP fait cette année en FT712 – Techniques d'apprentissage - Kaggle - Leaf Classification Challenge - Documentation de Scikit-Learn - Genre Quercus (Chênes) - Git / GitHub