Projet Technique d'apprentissage

IFT-712

Par
Timothée Blanchy (*timb1101*)
Mengling Guo (*guom3201*)
Oussama Khaloui (*khao1201*)

Lien github
https://github.com/KHALOUIoussama/IFT712-Project

Automne 2023
Faculté des sciences de Sherbrooke
Département informatique

Université de Sherbrooke

Table de Matières

I. Introduction

Notre projet vise à explorer en profondeur diverses méthodologies de classification appliquées à un ensemble de données Kaggle dans le but de classer des feuilles d'arbre.

L'objectif principal de ce projet est d'évaluer et de comparer six méthodes de classification distinctes dans le contexte de l'identification des feuilles d'arbres. Notre approche met l'accent sur l'utilisation de la validation croisée et de l'ajustement des hyperparamètres pour optimiser un modèle afin de trouver les solutions les plus efficaces pour la classification des feuilles.

Nos méthodes de classification comprennent AdaBoost, les arbres de décision, le perceptron, les réseaux neuronaux, les machines à vecteurs de support (SVM) et les K-voisins.

II. Description du dataset

L'objectif principal de notre modèle est de déterminer l'espèce à laquelle appartient une feuille donnée, en se basant sur les caractéristiques extraites des images des feuilles de notre dataset. Ces caractéristiques sont représentées par trois vecteurs de dimension 64, correspondant respectivement aux attributs de marge (margin), de forme (shape) et de texture.

Notre base de données se compose d'environ 1 584 images de feuilles, représentant 16 échantillons pour chacune des 99 espèces. Chaque image est traitée pour obtenir une représentation en noir et blanc, où les feuilles sont en noir sur un fond blanc. Trois ensembles distincts de caractéristiques sont extraits de ces images :

- Un descripteur continu de la forme de la feuille.
- Un histogramme représentant la texture intérieure de la feuille.
- Un histogramme détaillé de la marge de la feuille.

Pour chaque échantillon de feuille, nous disposons d'un vecteur de 64 attributs pour chacune des trois caractéristiques mentionnées. Le jeu de données comprend les attributs suivants :

- id : un identifiant unique pour chaque image.
- species : l'espèce à laquelle appartient l'échantillon (disponible uniquement dans le dataset d'entraînement).
- margin_1 à margin_64 : les 64 attributs pour la caractéristique de marge.
- shape_1 à shape_64 : les 64 attributs pour la caractéristique de forme.
- texture_1 à texture_64 : les 64 attributs pour la caractéristique de texture.

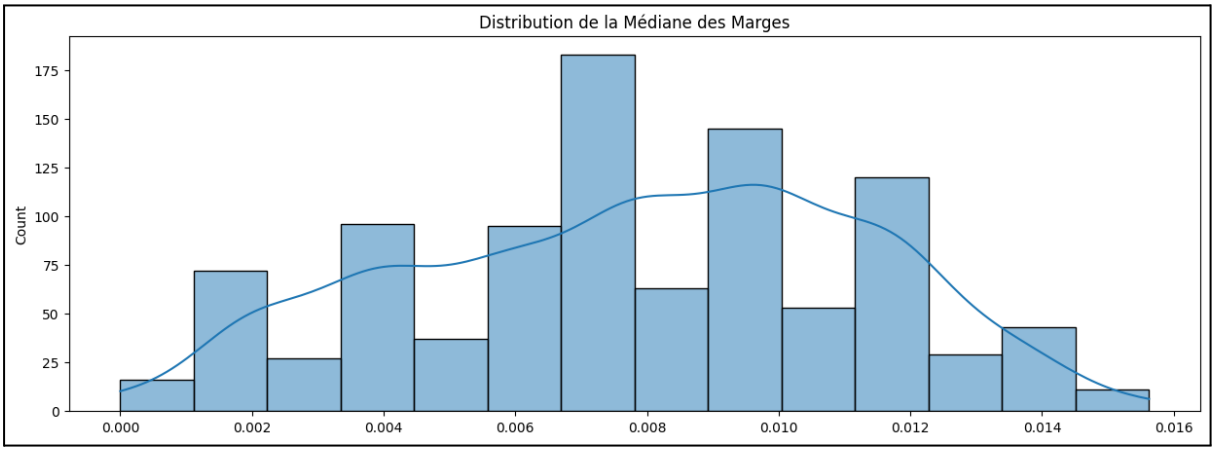
Les données sont divisées en deux ensembles : un ensemble d'entraînement (train.csv) contenant 990 lignes et un ensemble de test (test.csv) comprenant 594 lignes, suivant un ratio de 63% pour l'entraînement et 37% pour les tests. Le fichier test.csv est dépourvu des étiquettes des espèces et est utilisé exclusivement pour la soumission des résultats sur la plateforme Kaggle. En revanche, le fichier train.csv contient les étiquettes nécessaires et est utilisé pour l'entraînement et l'évaluation de nos modèles.

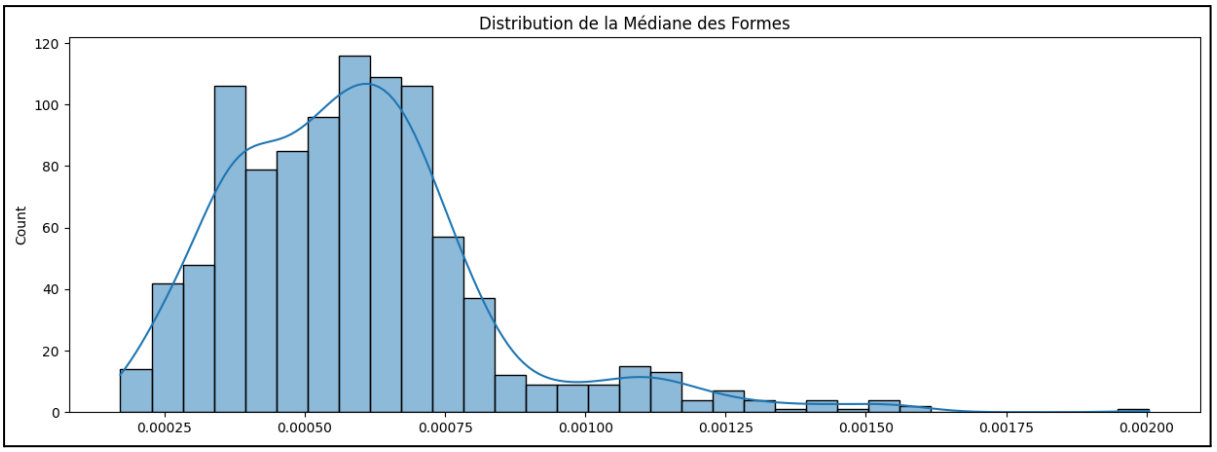
III. Analyse exploratoire des données

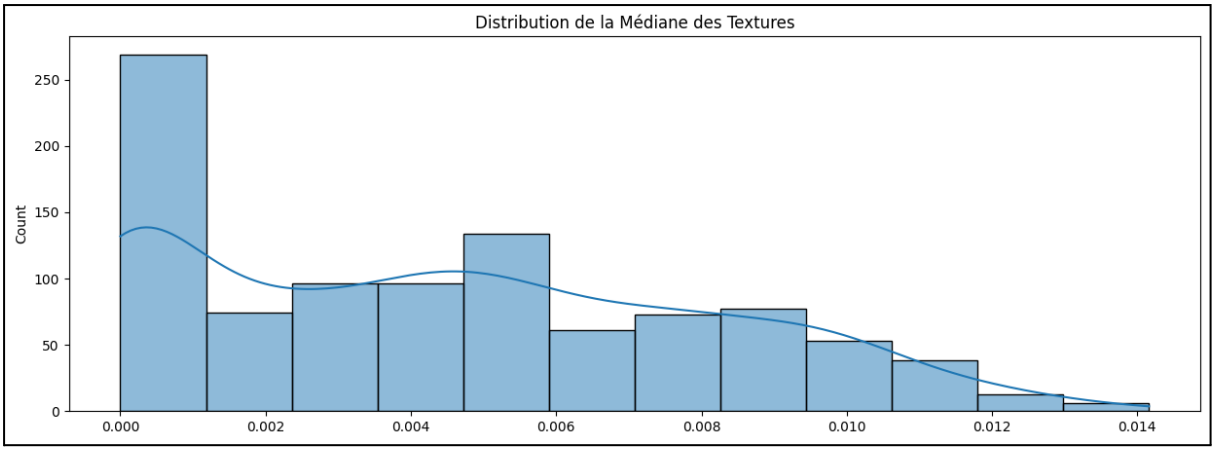
Cette section vise à approfondir notre compréhension des données dont nous disposons.

Les données d'entraînement présentent une répartition équilibrée, avec exactement 10 échantillons de feuilles par espèce. Cet équilibre assure que notre modèle ne sera pas biaisé en faveur d'une espèce particulière lors de l'apprentissage.

Dans la deuxième phase de notre analyse, nous avons cherché à déchiffrer la forme et la distribution des valeurs au sein des trois vecteurs de caractéristiques (marge, texture, forme). Compte tenu que chaque vecteur est composé de 64 valeurs, la médiane a été calculée pour chaque vecteur. Cette mesure de tendance centrale a été choisie pour sa résistance aux valeurs extrêmes, nous fournissant ainsi un aperçu fiable de la distribution typique des valeurs pour chaque caractéristique.







Ensuite, nous nous sommes attelés à une analyse plus poussée des corrélations entre ces différents vecteurs. Les résultats obtenus révèlent les niveaux d'interdépendance entre les formes, les textures et les marges des feuilles. Ces informations sont cruciales pour la sélection des caractéristiques et la construction d'un modèle prédictif robuste.

Combination	Mean Correlation (%)
Shape	68.864458
Margin	25.653617
Texture	15.866776
Texture-Margin	7.806652
Shape-Margin	6.745317
Shape-Texture	5.475600

Avec une corrélation moyenne de 68.86%, les caractéristiques liées à la forme des feuilles présentent une forte corrélation interne. Cela pourrait suggérer que les variables au sein de cet ensemble sont peut-être redondantes ou qu'elles capturent des informations très similaires sur la forme des feuilles.

Les ensembles de caractéristiques de forme (Shape) présentent la plus grande similarité entre eux, ce qui pourrait nécessiter une attention particulière lors de la sélection des caractéristiques pour éviter la multicollinéarité. En revanche, la combinaison des caractéristiques de forme et de texture pourrait offrir la meilleure diversité d'informations pour le modèle de prédiction des espèces de feuilles.

IV. Démarche scientifique

A. Préparation des données

Dans notre processus de prétraitement des données, nous avons suivi plusieurs étapes clés :

- Séparation du Jeu de Données : Nous avons divisé nos données en deux ensembles distincts : un pour l'entraînement de notre modèle et l'autre pour tester son efficacité. Cette séparation a été réalisée en utilisant la méthode *Stratified Shuffle Split*. Étant donné la quantité limitée de données disponibles par espèce, cette approche garantit une représentation équilibrée de chaque espèce dans l'ensemble de test. Cela est crucial pour obtenir des résultats de test fiables et représentatifs.

- Gestion des Données Manquantes : Nous avons constaté l'absence de données manquantes dans notre jeu de données, ce qui suggère que nos données ont déjà été préalablement nettoyées.

- Transformation des caractéristiques : Pour une manipulation plus aisée, nous avons converti les données catégoriques (comme les espèces) en données numériques.

Pour optimiser nos données pour l'entraînement, nous avons envisagé quatre méthodes :

1) **Normalisation des Données** : Utilisation du Z-score pour standardiser nos données. Cette approche aide à neutraliser l'effet des échelles variables et à faciliter la convergence du modèle lors de l'entraînement.

2) **Détection des Données Aberrantes** : Utilisation de l'Isolation Forest (iso_forest) pour identifier les outliers. Nous avons expérimenté avec différents seuils (1%, 3%, et 5%) pour déterminer le pourcentage optimal de données à considérer comme aberrantes. Une attention particulière a été portée pour éviter la perte excessive de données dans les ensembles d'entraînement ou de test, surtout pour les espèces moins représentées.

3) **Sélection des Caractéristiques** : Utilisation de la méthode Recursive Feature Elimination (RFE) pour identifier et conserver les caractéristiques les plus influentes. L'objectif est de minimiser la perte d'information tout en simplifiant le modèle, réduisant ainsi le temps d'entraînement et améliorant la capacité de généralisation.

4) **Réduction de Dimensionnalité** : Application de l'Analyse en Composantes Principales (ACP) pour réduire la multicolinéarité en transformant les variables corrélées en nouvelles variables indépendantes.Un défi associé est la détermination du nombre optimal de composantes principales, un aspect également soumis à des tests pour chaque modèle.

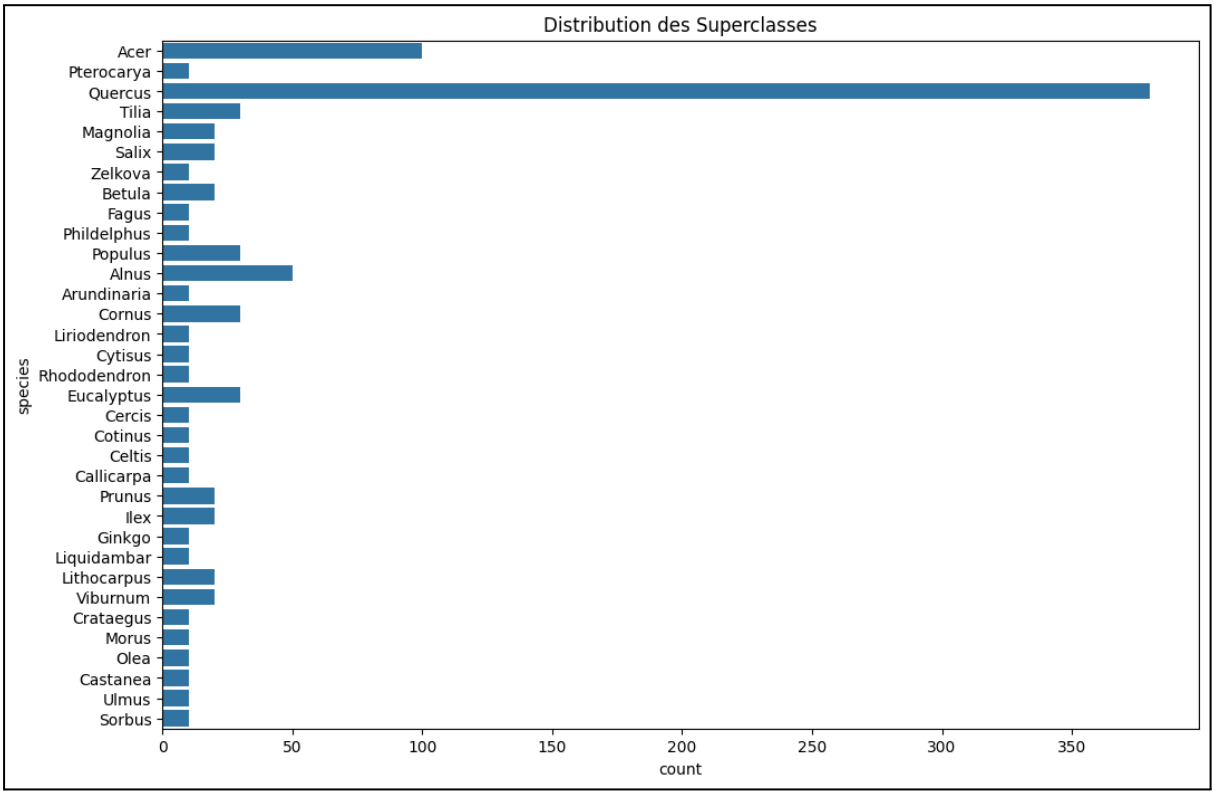
Combinaison des Méthodes d'Optimisation : En plus de tester chaque méthode individuellement, nous avons également exploré des combinaisons de ces techniques pour

optimiser nos ensembles de données. Par exemple, l'utilisation conjointe de la normalisation Z-score avec la détection des outliers. Ces combinaisons peuvent offrir des avantages synergiques, comme une meilleure gestion des biais et une réduction plus efficace de la complexité des données, tout en préservant les informations essentielles.

B. Super-Classes

Face à la contrainte d'un nombre restreint d'échantillons par espèce dans notre jeu de données - seulement 10 observations pour chacune – nous avons envisagé une stratégie pour enrichir notre ensemble de données. Notre approche consistait à regrouper les espèces en superclasses basées sur le genre botanique, c'est-à-dire en utilisant le préfixe commun du nom des espèces (par exemple, regrouper Acer_Opalus sous la superclasse Acer). Cette reclassification a augmenté significativement le nombre de données disponibles par superclasse, dépassant le seuil initial de 10.

Cependant, cette méthode a introduit un nouveau défi : une distribution inégale des échantillons parmi les superclasses. Par exemple, la superclasse Quercus s'est avérée être particulièrement volumineuse avec plus de 350 échantillons, tandis que d'autres superclasses ne comprenaient que le minimum de 10 échantillons. Cette disparité dans la taille des superclasses pourrait entraîner un biais dans les résultats de notre modèle, favorisant les superclasses avec un plus grand nombre d'échantillons.



Les fichiers CSV correspondant à chaque méthode de traitement des données sont disponibles dans le dossier ./data/processed/

C. Modèles utilisés

Nous avons essayé dans notre projet de prendre des classifieurs qui ne se ressemblaient pas :

- Perceptron
- Réseaux de neurones
- K-voisins
- Arbre de décision
- Vecteurs de support
- AdaBoost

En effet, nous observons qu'il y a des méthodes de type réseau (réseaux de neurones, perceptron), de type arbre (arbre de décision), de type machine à vecteurs de support, et de boosting (AdaBoost).

Le but de cette sélection est d'aller chercher un éventail varié de méthodes ayant chacune leurs forces et leurs faiblesses. Par exemple, le **Perceptron** est un algorithme simple d'implémentation, rapide, mais très limité au fait que les données doivent être linéairement séparables. Dans le cas contraire, des fonctions de bases peuvent être utilisées afin de changer la dimensionnalité de notre modèle et de projeter nos données dans un nouvel espace dimensionnel dans lequel nos données sont linéairement séparables.

Un algorithme de type arbre, quant à lui, comme l'**arbre de décision**, effectue sa classification sans nécessiter trop de calculs et peut très bien supporter autant des données catégoriques que des données continues. Toutefois, ce type de modèle peut être très coûteux en temps de calcul à entraîner. Dans les cas où nous aurions un nombre N de données à séparer, ces mêmes données doivent toutes être séparées à chaque nœud du graphe étant donné la feature sélectionnée. Ces divisions répétées à chaque nœud peuvent rapidement devenir coûteuses en temps et en puissance de calcul.

D. Recherche d'hyper-paramètres

Dans l'optimisation de nos modèles de machine learning, la recherche d'hyperparamètres est déterminante pour améliorer la performance. Nous utilisons le Grid Search, une méthode d'exploration exhaustive, pour tester une variété de valeurs d'hyperparamètres et identifier celles qui maximisent l'accuracy, notre métrique de performance choisie.

Nous avons choisi l'accuracy car elle fournit une mesure directe et intuitive de la réussite globale du modèle.

Dans notre approche de Grid Search, chaque combinaison d'hyperparamètres est évaluée par le biais d'une validation croisée à cinq plis (cv=5). Ce choix repose sur un compromis entre la précision de l'estimation et la contrainte des ressources. Avec cinq plis, nous obtenons une estimation fiable de la performance du modèle tout en maintenant la charge de calcul à un niveau gérable. Cela signifie que pour chaque configuration d'hyperparamètres, le modèle est entraîné cinq fois, à chaque fois sur un sous-ensemble différent du jeu de données, ce qui assure une évaluation moins sujette au biais dû aux particularités d'une division spécifique des données.

Pour effectuer cette recherche de manière rationnelle, nous avons adopté la stratégie suivante :

- Pré-sélection : Nous avons d'abord établi une gamme d'hyperparamètres pour chaque modèle.
- Granularité : Nous avons choisi des intervalles d'hyperparamètres avec une granularité modérée. Cela nous permet d'explorer l'espace des hyperparamètres de manière approfondie sans pour autant faire des calculs interminables pour des différences minimes.

Cette approche méthodique nous permet de maximiser la performance de nos modèles tout en tenant compte des contraintes pratiques et en assurant l'efficacité du processus.

V. Structure du projet et implémentation

Nous avons décidé de structurer clairement notre projet pour qu'il soit clair et simple à prendre en main, améliorer ou debugger.

Pour cela nous sommes partis de la structure de cookiecutter (Lien github) qui nous donne les bases pour un projet d'apprentissage et nous avons adapté ce projet à nos besoins.

La structure complète peut être retrouvée dans le fichier `readme.md`, mais en voici les parties essentielles :

- ★ Le dossier `data` contient les différents csv de données réparties dans `raw` et `processeed`.
- ★ Le dossier `graphs` contient tous les graphiques produits dans le code.
- ★ Comme son nom l'indique, le dossier `notebooks` contient les différents notebooks lançant les différents éléments du projet.
- ★ Enfin le dossier `src` contient les sources de notre projets réparties en :
  - `data`, contenant le traitement des données
  - `models`, contenant les différents modèles implémentés
  - `visualization`, contenant les scripts permettant l'exploitation des résultats

Nous avons également décidé d'utiliser des classes pour mieux structurer notre projet. Vous pouvez retrouver ci-dessous notre diagramme de classes.

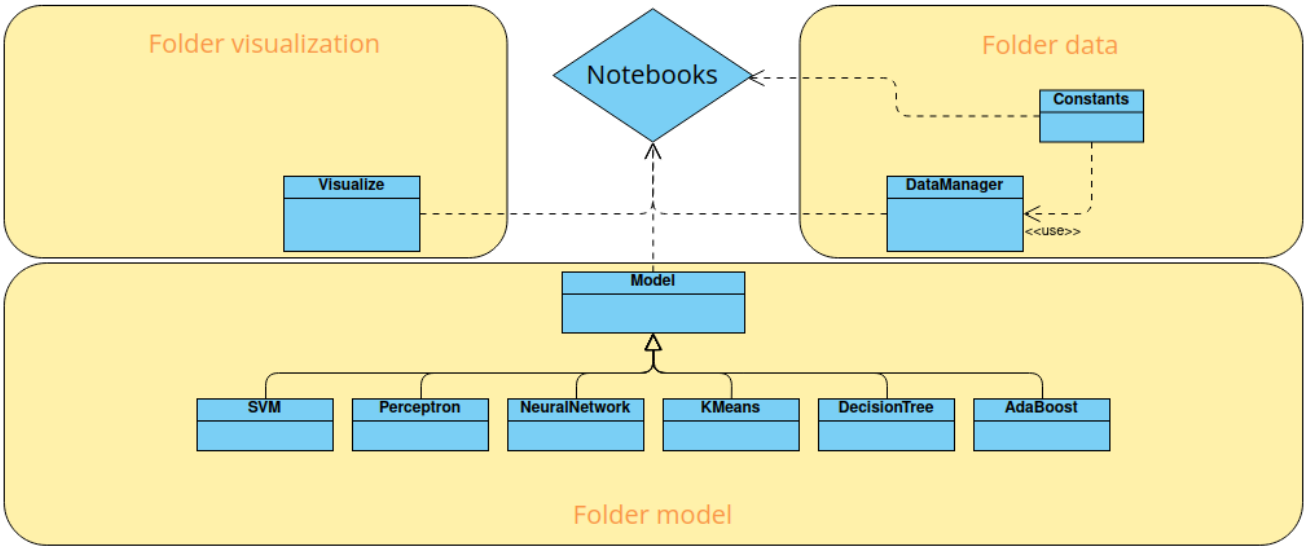


Diagramme de classe

Pour rentrer un peu dans les détails, nous avons généralisé un maximum les modèles avec la classe abstraite Model implémentée par les 6 modèles que nous avons choisis.

VI. Comparaison des résultats

A. Choix des métriques

Pour évaluer de manière approfondie la performance de nos algorithmes de classification et établir des comparaisons fiables entre eux, nous avons retenu un ensemble de métriques pertinentes. Ces métriques comprennent :

- ★ Accuracy : Elle mesure la proportion des prédictions correctes par rapport au total des prédictions effectuées. Une valeur de 1 est le meilleur résultat possible.
- ★ Précision : Ce ratio indique le nombre d'observations positives correctement identifiées parmi toutes les prédictions positives. Une précision de 1 est l'idéal.
- ★ Recall : Il évalue la proportion d'observations positives réelles correctement identifiées par le modèle. Le recall parfait est également de 1.
- ★ F1-score : C'est la moyenne harmonique de la précision et du rappel, offrant un équilibre entre les deux lorsque leur valeur est proche. Un F1-score de 1 indique une précision et un rappel optimaux.
- ★ Log-loss : Cette métrique évalue la performance d'un modèle de classification où la prédiction est une probabilité entre 0 et 1. La perte augmente à mesure que la prédiction s'éloigne de la valeur réelle, avec un score parfait étant 0.
    - *Il est à noter que pas tous les classifieurs sont capables de calculer la log-loss, vu qu'il nécessite que le modèle produise des probabilités pour chaque classe. Les algorithmes qui génèrent uniquement des étiquettes de classe, sans estimer la probabilité sous-jacente ne peuvent pas fournir les informations nécessaires pour cette métrique. (voir annexe)*

Ces indicateurs sont largement reconnus et employés dans le domaine de l'apprentissage automatique pour leur capacité à fournir une analyse complète de la performance des modèles. Nous avons opté pour leur représentation sous forme de diagrammes à barres, ce qui en améliore la clarté et la facilité d'interprétation.

Néanmoins, il est important de ne pas sous-estimer l'analyse du sous-apprentissage et du sur-apprentissage, qui peuvent sérieusement compromettre l'efficacité d'un modèle. Pour identifier ces phénomènes, nous utilisons des courbes d'apprentissage qui montrent l'évolution de la fonction de perte sur les ensembles d'entraînement et de validation au fil de l''apprentissage. Une courbe d'apprentissage idéale montre une convergence des erreurs de

perte sur les ensembles d'entraînement et de validation, signifiant que le modèle est généralisé de manière optimale pour de nouvelles données non vues.

B. Résultats

Pour chacun des modèles nous avons affiché :

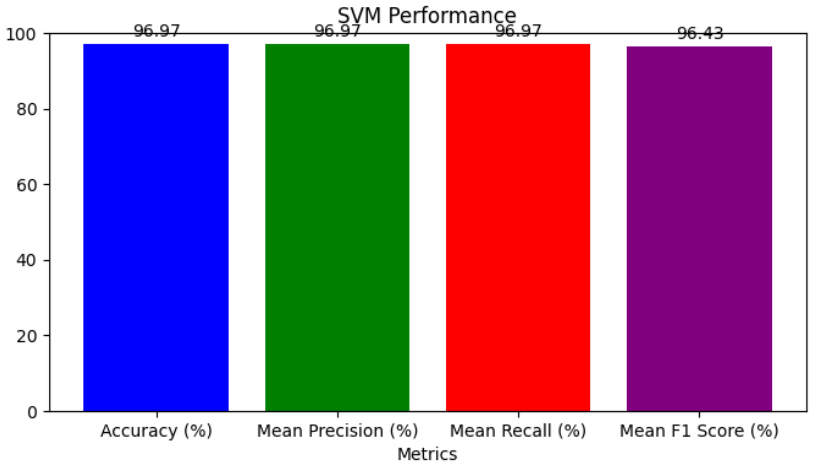
- ➔ Les métriques
- ➔ La matrice de confusion
- ➔ La courbe d'apprentissage
- ➔ La comparaison des métriques avec l'entraînement sur d'autres dataset (avec super classes, normalisé, en supprimant les 1% des données les plus aberrantes, normalisé ET sans les 1% des données les plus aberrantes et avec ACP "Analyse en Composantes Principales")
- ➔ La comparaison des métriques avec un entraînement sans choisir les meilleurs hyperparamètres possibles

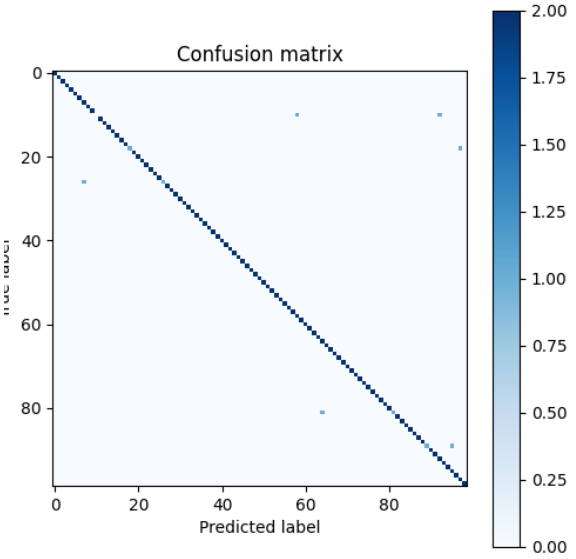
Il est possible de retrouver tous ces graphiques en annexe.

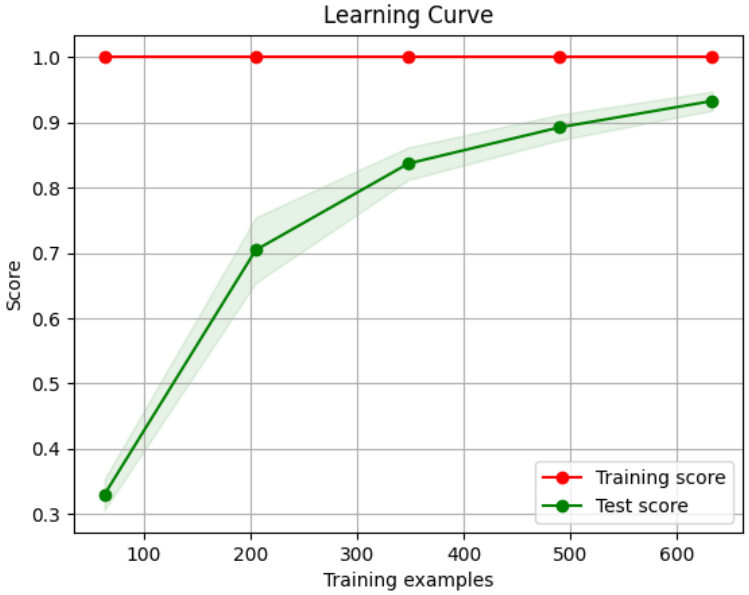
Concernant, les datasets,

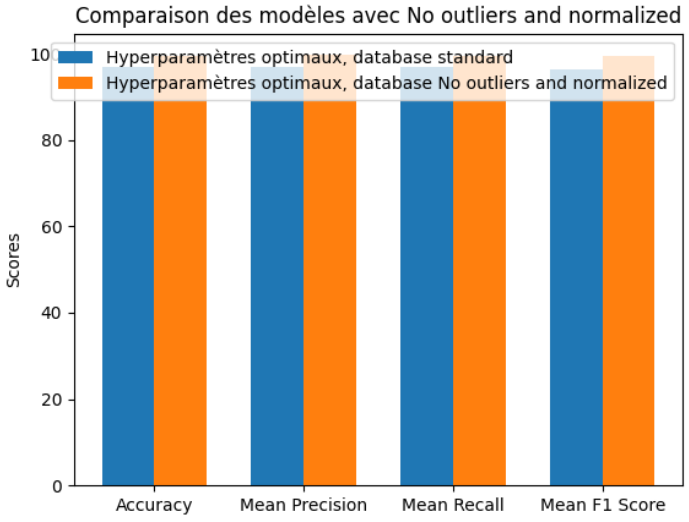
- La base de données normalisée (Z-score) donne un meilleur apprentissage pour toutes les données.
- La base de donnée en supprimant les 1% des données les plus aberrantes est meilleure en générale, sauf pour l'arbre de décision et AdaBoost
- La performance de notre modèle s'est avérée diminuée lors de l'utilisation de la base de données réduite par l'Analyse en Composantes Principales (ACP). Cette observation suggère que le nombre de composantes principales retenues, fixé arbitrairement à 68, n'était peut-être pas optimal. Le manque de temps n'a pas permis d'expérimenter avec diverses quantités de composantes pour affiner ce paramètre.
- Pour Adaboost, les arbres de décision et Svm la base de donnée avec les super classes inégalement réparties à fait dégringoler les résultats. Pour les autres, elle les a légèrement améliorés.

Notre meilleur algorithme est SVM ! Il dépasse légèrement le réseau de neurones, avec une accuracy, une précision, un recall et un f1-score d'environ 97% sur le dataset original. Ces résultats montent encore avec la base de données normalisée et sans les 1% des données les plus aberrantes.


SVM Performance
96.97
96.97
96.97
96.43
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Metrics


Confusion matrix
Predicted label


Learning Curve
Score
Training examples
Training score
Test score


Comparaison des modèles avec No outliers and normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers and normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

VII. Gestion de projet

Pour gérer ce projet, nous avons eu recours à divers outils adaptés à notre taille d'équipe : Teams, Trello, Github, Git, Vscode et Google Docs.

A. Teams

Cet outil nous a servi à communiquer entre nous. Nous avons créé au début du projet un canal de discussion avec les trois membres du groupe, et surtout nous avons pu avoir des réunions en distanciel dessus. En effet, l'un des membres du groupe habitant à Montréal, il n'était pas question de se retrouver à la bibliothèque pour avancer dans le projet.

Nos réunions teams ont été le cadre de discussions sur l'organisation du projet et la répartition du travail à faire.

B. Trello

Pour nous répartir les tâches, savoir où nous en étions, évaluer le travail restant, tester ce qui était prêt à être testé, nous avons décidé d'utiliser le site https://trello.com qui propose un tableau "kanban" permettant de facilement modéliser les différentes tâches à faire, leurs états et qui sont les personnes responsables.

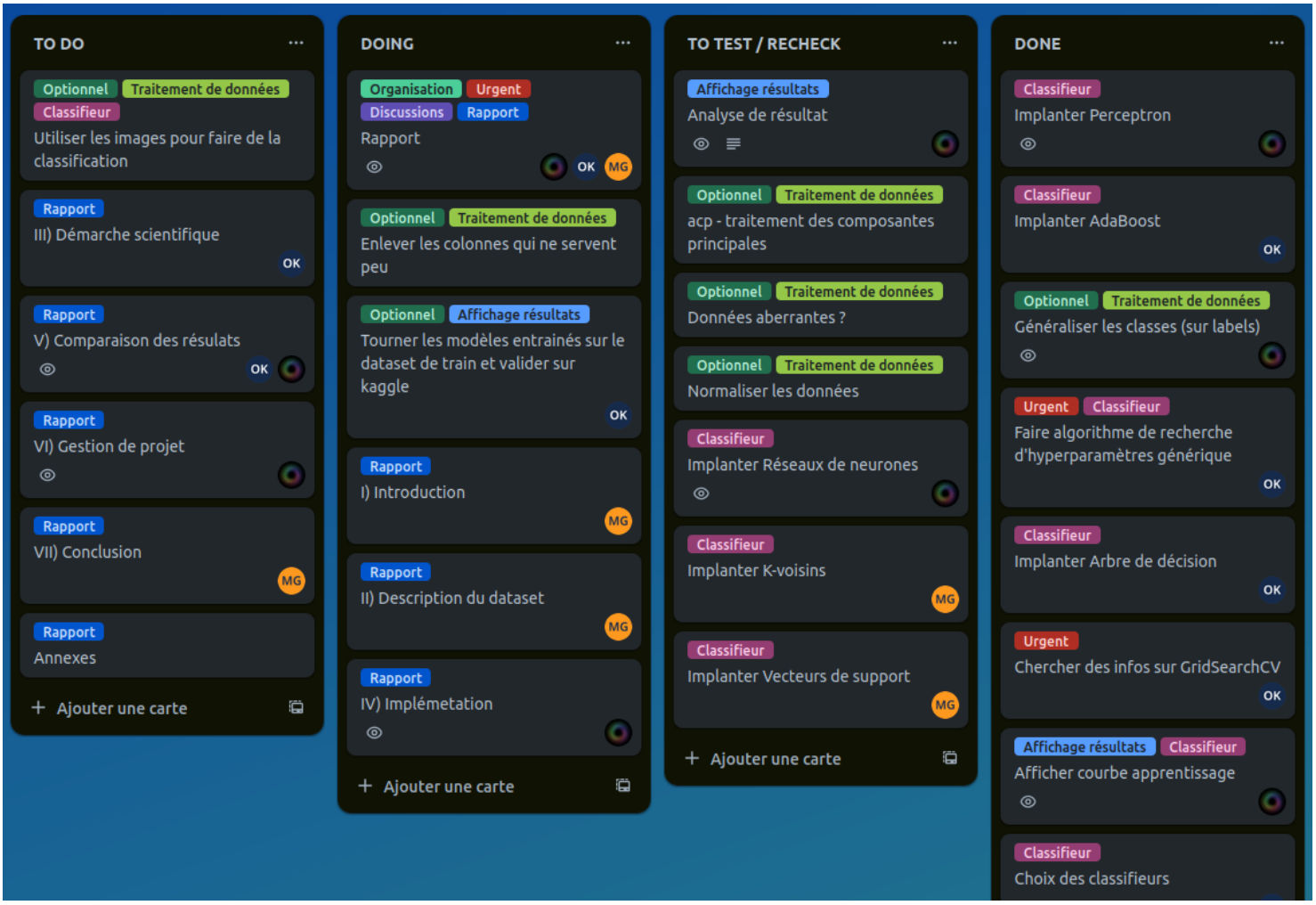
Notre trello est accessible avec le lien suivant :
https://trello.com/invite/b/l3w0aB23/ATTI536358e38e9190b8c5af3840522cd66d71ADEC7F/ift712

Nous avons utilisé 4 colonnes :

- ➔ TO DO : la liste des tâches à faire
- ➔ DOING : la liste des tâches en cours
- ➔ TO TEST / RECHECK : la liste des tâches à vérifier
- ➔ DONE : la liste des tâches finies

Nous avons aussi utilisé les étiquettes ci-joint pour décrire les tâches.

Ce trello nous a permis de mieux gérer notre travail, malgré le fait d'être à distance.



Aperçu du trello

C. Git et github

Ces deux outils nous ont servi pour le suivi du code que nous avons produit. Il s'agit d'un outil de versionnage de code accompagné d'une plateforme d'accès en ligne.

Notre github est accessible en public avec le lien suivant :
https://github.com/KHALOUIoussama/IFT712-Project

Nous avons utilisé différentes branches pour les différentes parties du projet avant de les fusionner dans la branche principale.

Git nous a permis de nous assurer que nous ne risquions pas de perdre du code par erreur en le supprimant sans faire exprès.

D. VScode

Vscode a été notre environnement de travail tout au long du projet. Cet éditeur de code a plusieurs avantages que nous avons exploités, en installant quelques extensions.

Nous avons pu ouvrir, modifier et lancer le notebook depuis l'éditeur de code, ainsi que le code python.

Lors de nos séances en distanciel, nous avons pu nous partager une session de code via l'extension "liveshare". Celle-ci permet de modifier en même temps et ensemble le code sur l'un des ordinateurs des membres du groupe.

E. Google Docs

Enfin, nous avons utilisé Google Docs pour réaliser notre rapport. Cet outil permet de modifier en temps réel, ensemble, un document texte.

Il permet également de mettre des commentaires et des suggestions de modifications, permettant de proposer des modifications aux autres sans supprimer ce qu'ils ont fait.

VIII. Conclusion

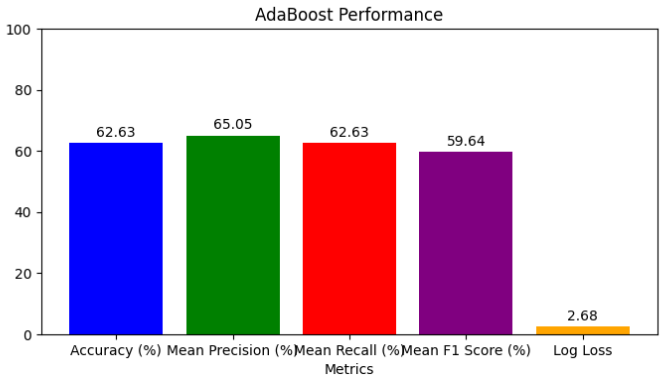
Pour conclure ce projet, nous sommes heureux de l'avoir réalisé car cela nous a permis de mettre en pratique de manière réaliste une partie de ce que l'on a appris durant ces cours. Il nous a aussi permis de mieux appréhender certains aspects des outils que nous avons déjà l'habitude d'utiliser. Par exemple, c'est une des premières fois que nous utilisons des branches sur git.

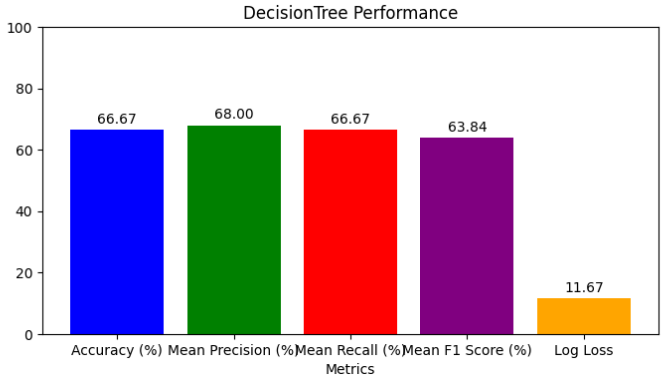
Concernant les résultats de notre analyse, les meilleurs sont obtenus avec SVM et le réseau de neurones avec une base de données normalisée avec 1% des données les plus aberrantes supprimées.

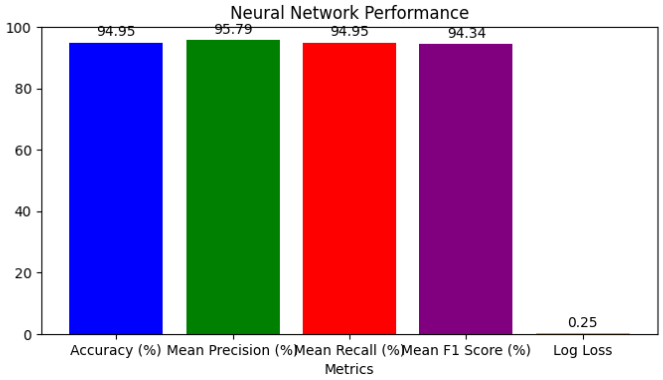
Il y a certains que nous auront pu améliorer si nous avions pu consacrer un peu plus de temps à ce projet :

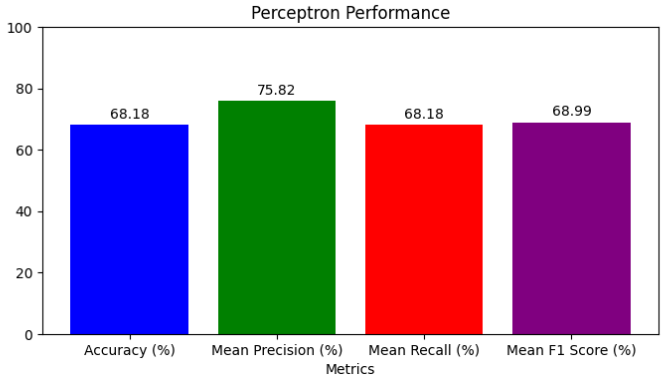
- ➔ Utiliser des modèles avancés de réseaux de neurones tels que RetinaNet, EfficientNet,...
- ➔ Utiliser les images pour classifier les feuilles, par exemple avec le réseau de neurones YOLOv8. [Feature extraction from images]

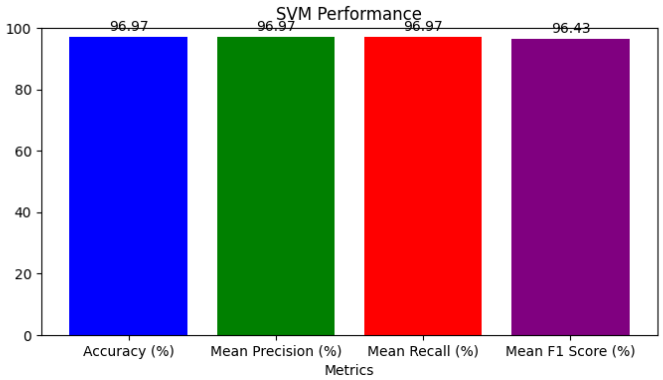
Annexes

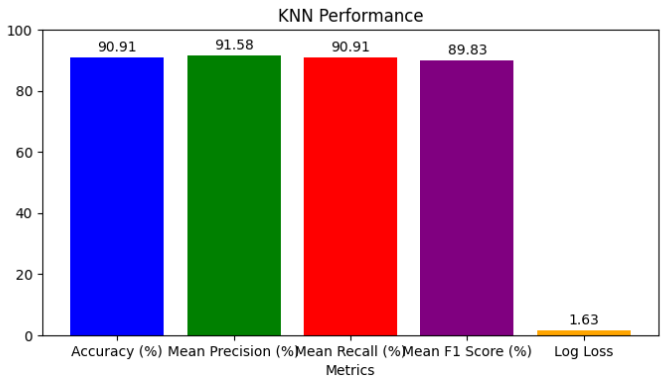

AdaBoost Performance
100
80
60
40
20
0
62.63
65.05
62.63
59.64
2.68
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Log Loss
Metrics


DecisionTree Performance
100
80
60
40
20
0
66.67
68.00
66.67
63.84
11.67
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Log Loss
Metrics

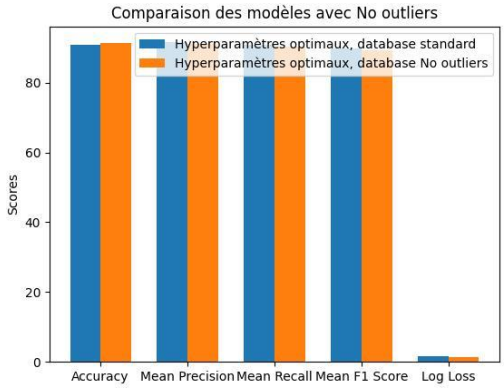

Neural Network Performance
100
80
60
40
20
0
94.95
95.79
94.95
94.34
0.25
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Log Loss
Metrics

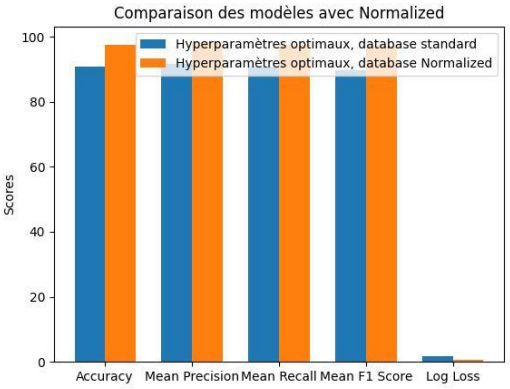

Perceptron Performance
100
80
60
40
20
0
68.18
75.82
68.18
68.99
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Metrics

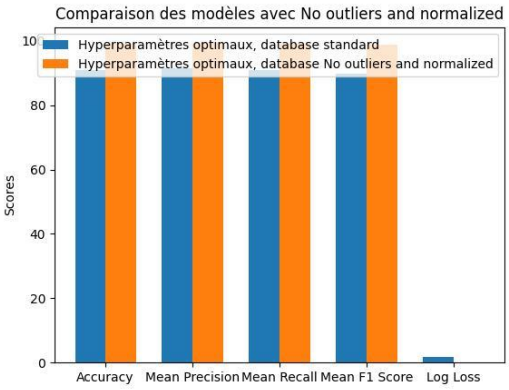

SVM Performance
100
80
60
40
20
0
96.97
96.97
96.97
96.43
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Metrics

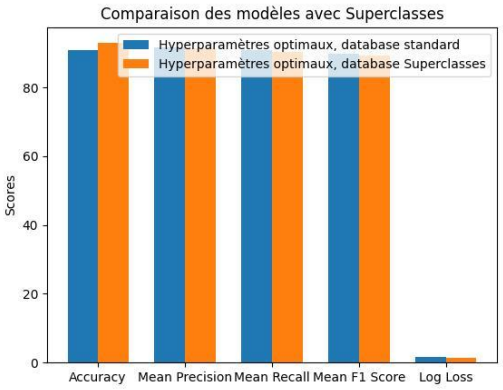

KNN Performance
100
80
60
40
20
0
90.91
91.58
90.91
89.83
1.63
Accuracy (%)
Mean Precision (%)
Mean Recall (%)
Mean F1 Score (%)
Log Loss
Metrics

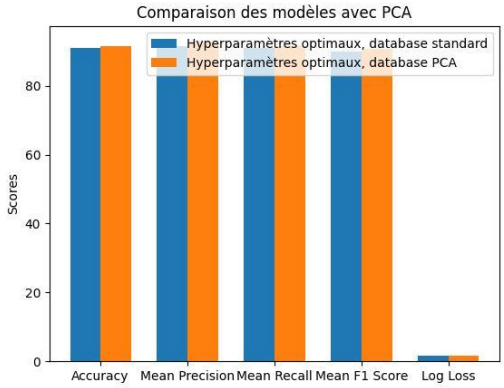
KNN :

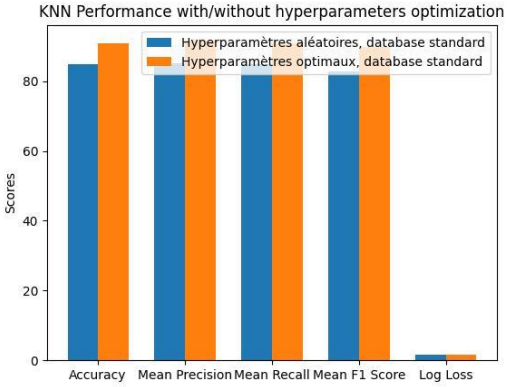

Comparaison des modèles avec No outliers
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

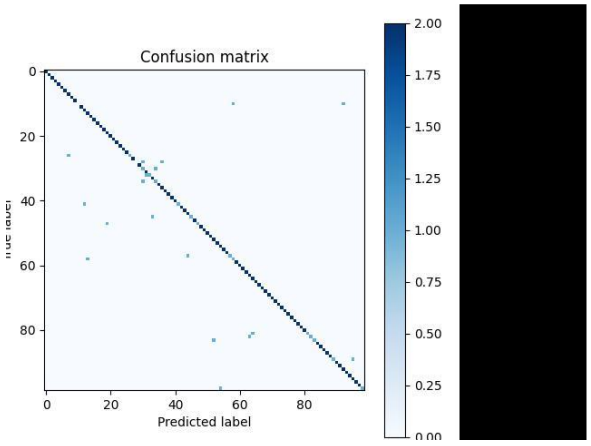

Comparaison des modèles avec Normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

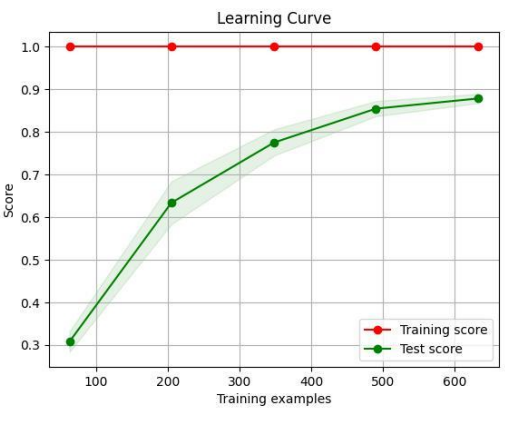

Comparaison des modèles avec No outliers and normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers and normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss


Comparaison des modèles avec Superclasses
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Superclasses
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

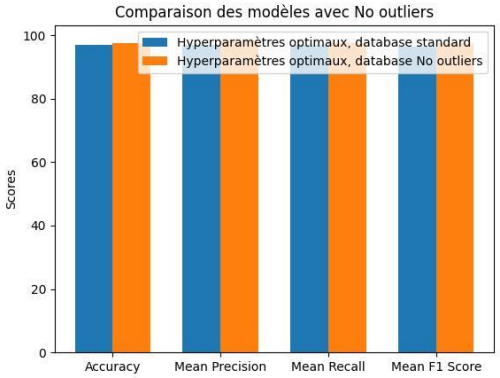

Comparaison des modèles avec PCA
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database PCA
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

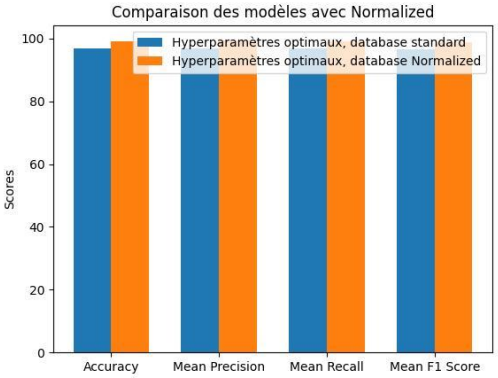

KNN Performance with/without hyperparameters optimization
Hyperparamètres aléatoires, database standard
Hyperparamètres optimaux, database standard
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

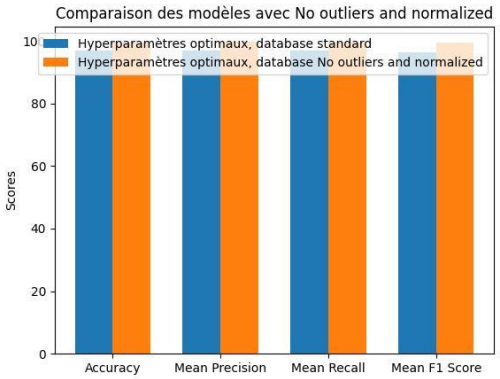

Confusion matrix
Predicted label

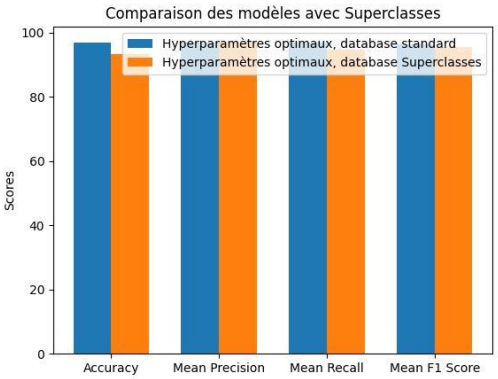

Learning Curve
Score
Training examples
Training score
Test score

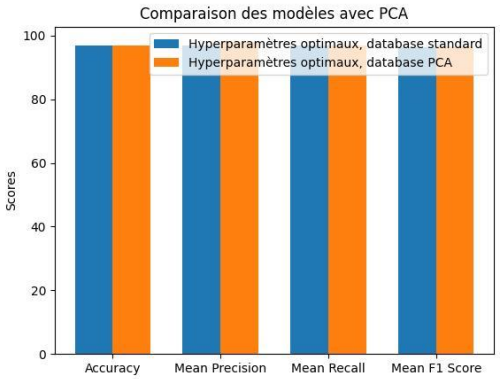
SVM :

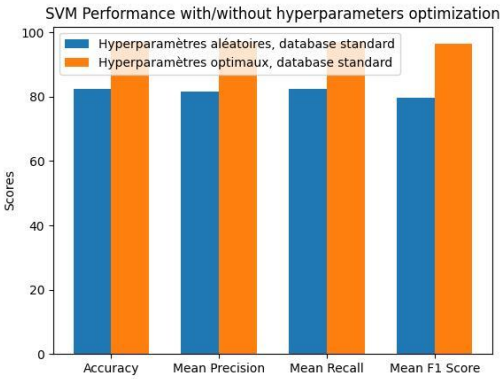

Comparaison des modèles avec No outliers
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

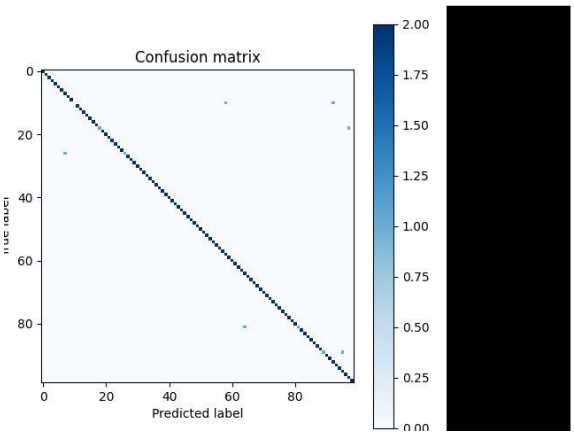

Comparaison des modèles avec Normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

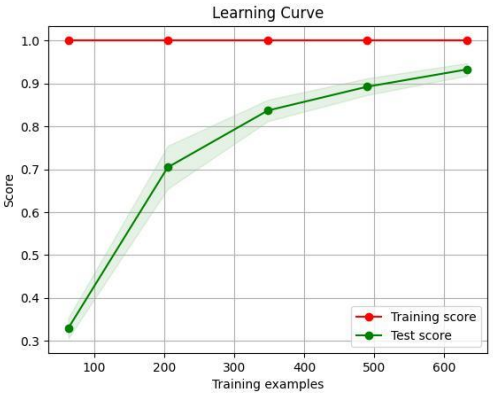

Comparaison des modèles avec No outliers and normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers and normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score


Comparaison des modèles avec Superclasses
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Superclasses
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

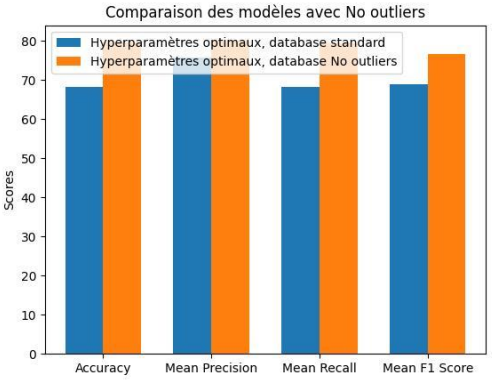

Comparaison des modèles avec PCA
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database PCA
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

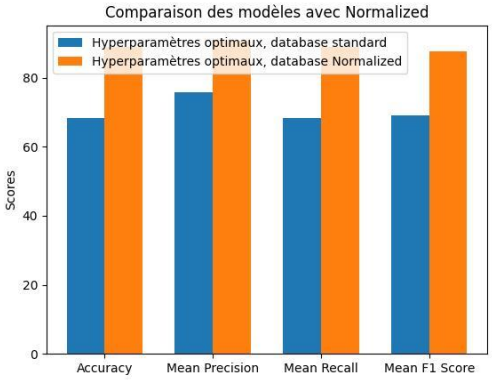

SVM Performance with/without hyperparameters optimization
Hyperparamètres aléatoires, database standard
Hyperparamètres optimaux, database standard
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

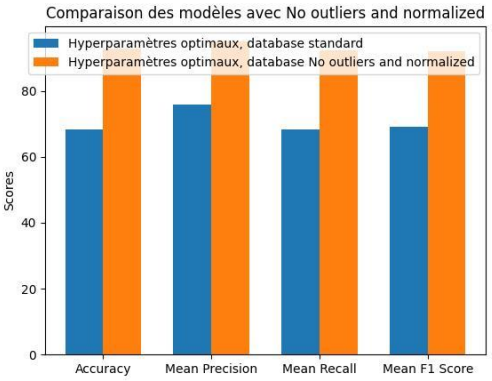

Confusion matrix
Predicted label

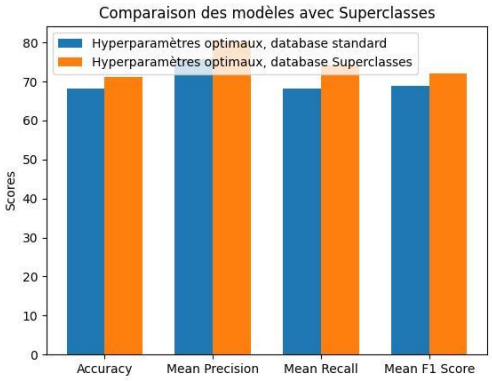

Learning Curve
Score
Training score
Test score
Training examples

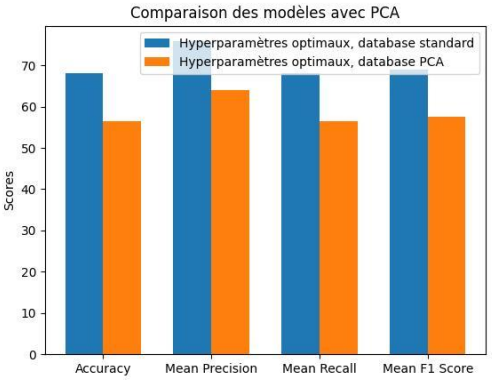
Perceptron :

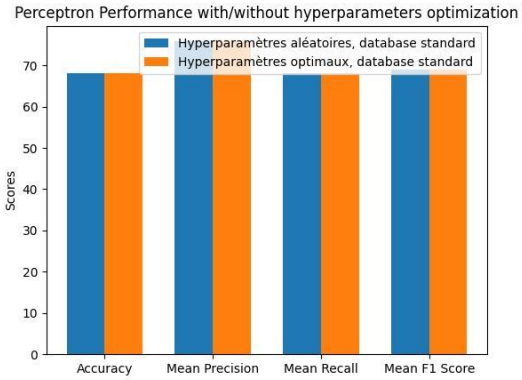

Comparaison des modèles avec No outliers
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

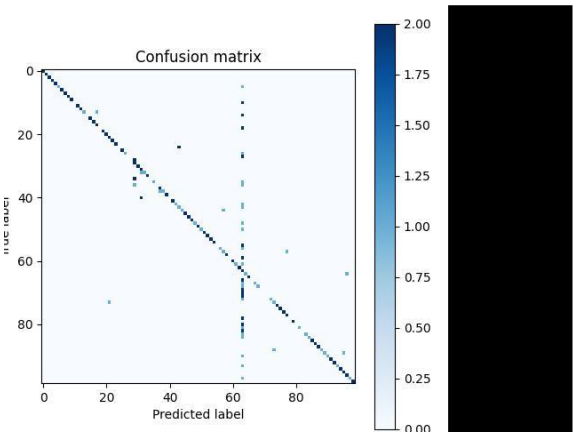

Comparaison des modèles avec Normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

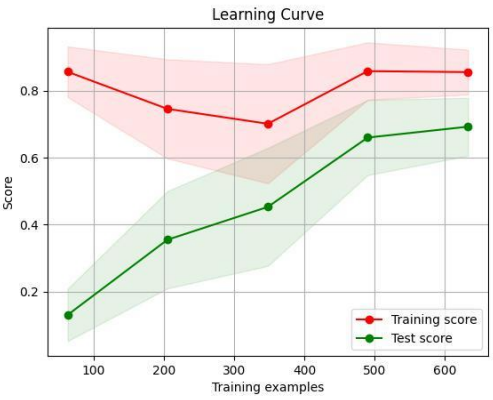

Comparaison des modèles avec No outliers and normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers and normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score


Comparaison des modèles avec Superclasses
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Superclasses
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

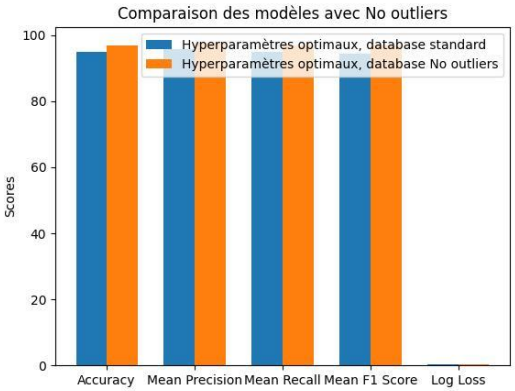

Comparaison des modèles avec PCA
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database PCA
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

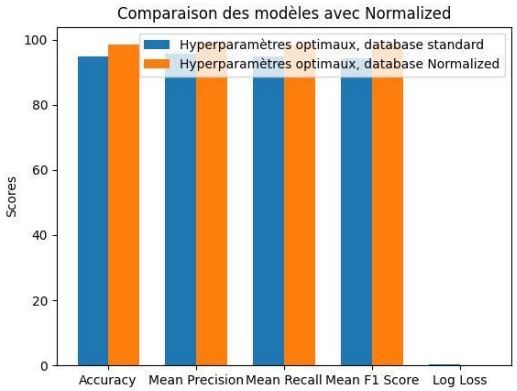

Perceptron Performance with/without hyperparameters optimization
Hyperparamètres aléatoires, database standard
Hyperparamètres optimaux, database standard
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score

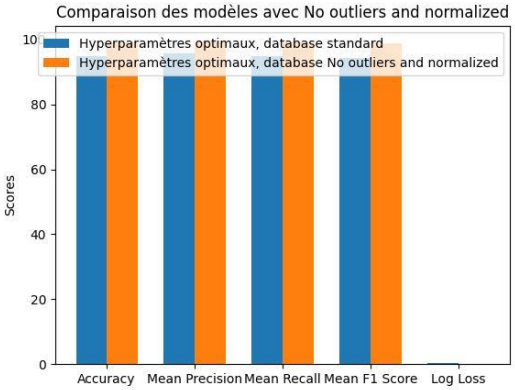

Confusion matrix
Predicted label

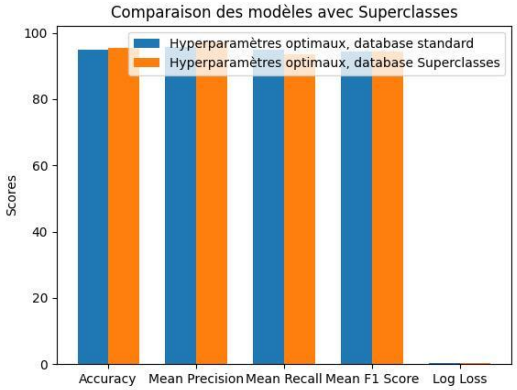

Learning Curve
Score
Training examples
Training score
Test score

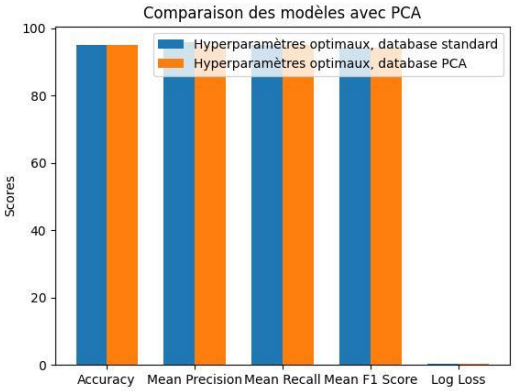
Neural Network :

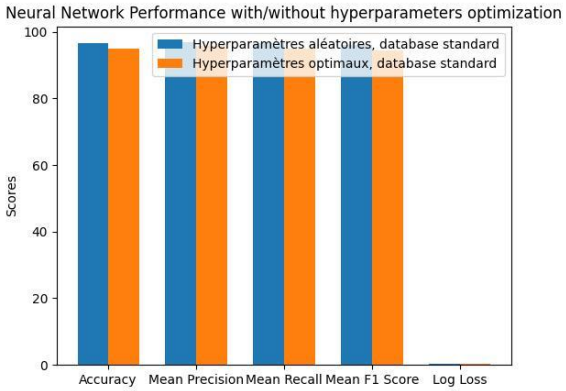


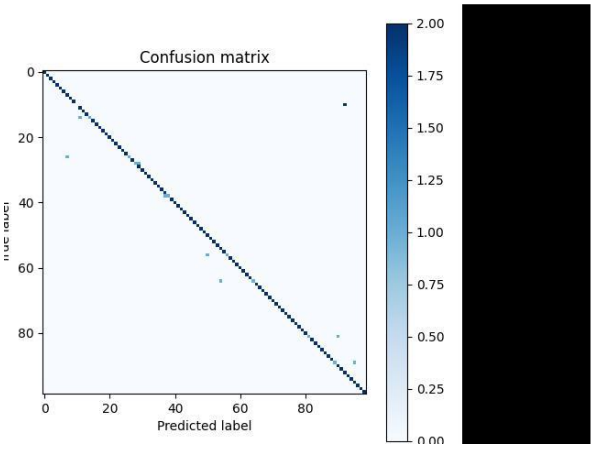


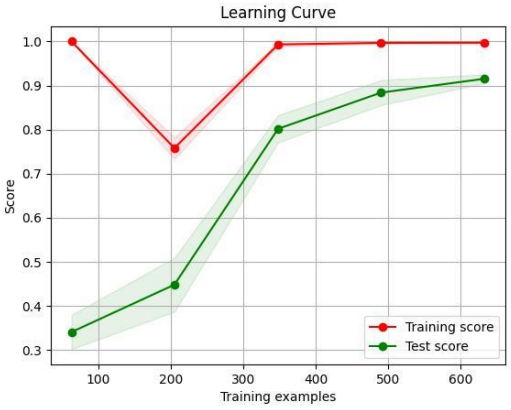




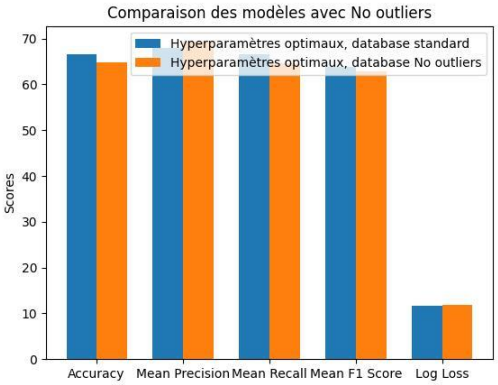


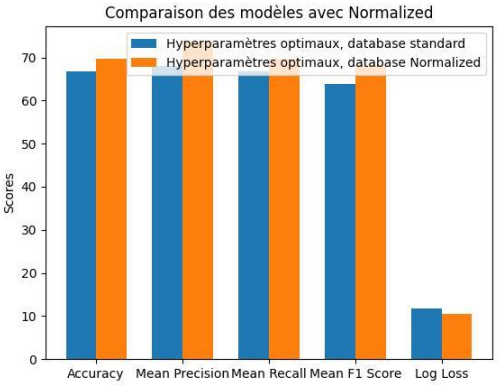


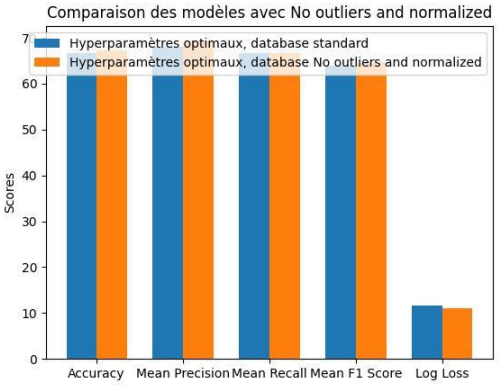


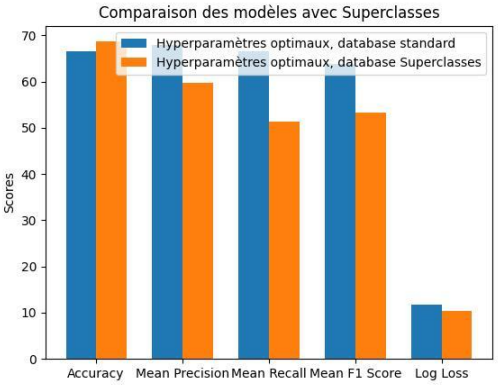


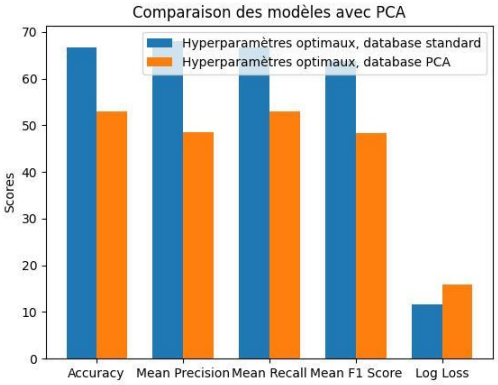
DecisionTree :

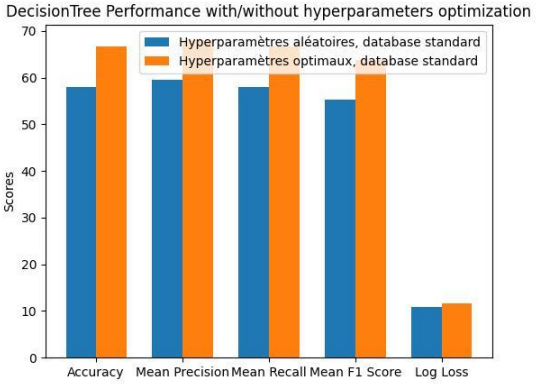


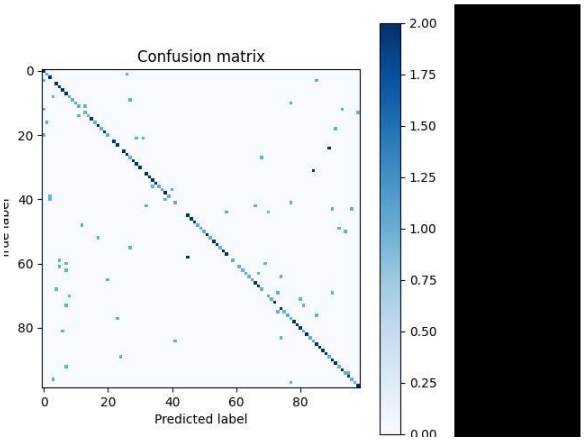


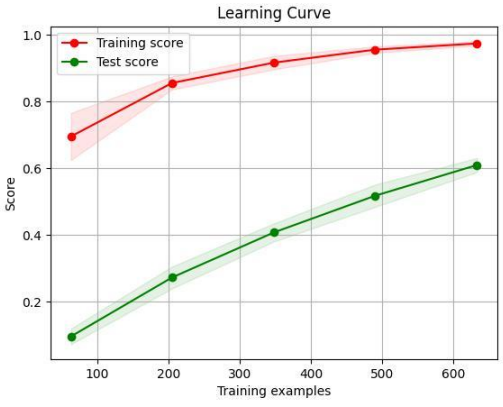




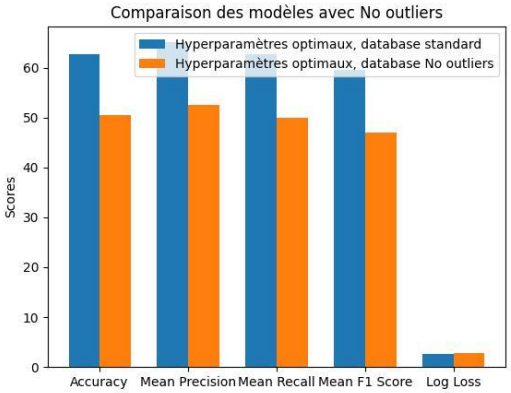


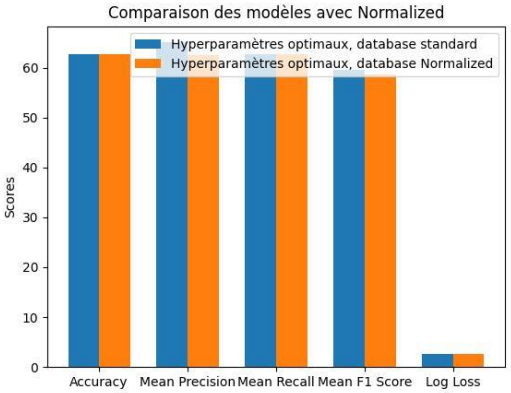


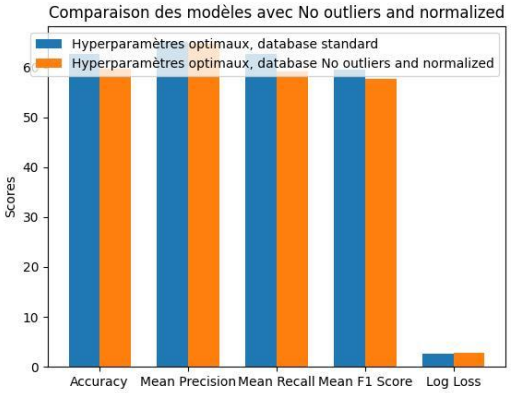


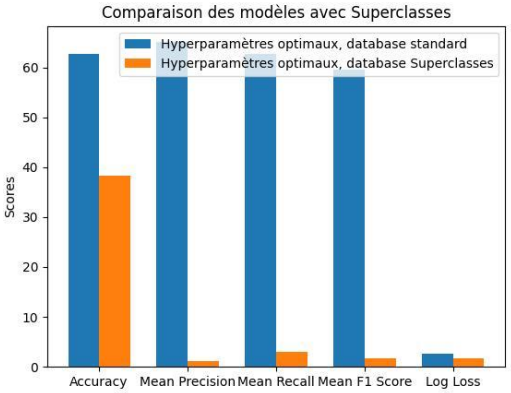


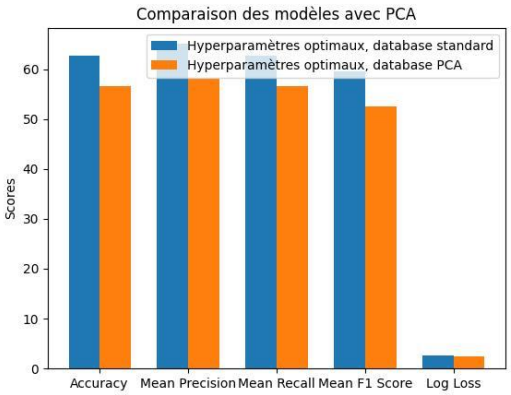
AdaBoost :

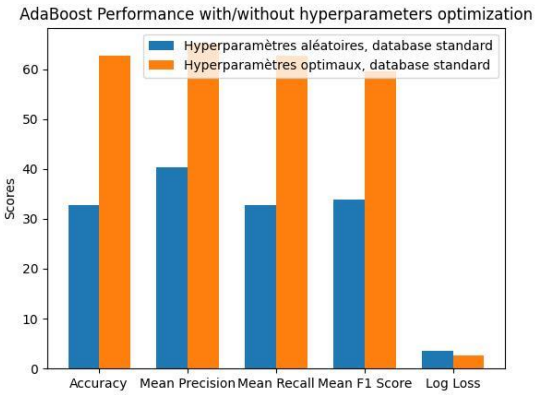

Comparaison des modèles avec No outliers
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

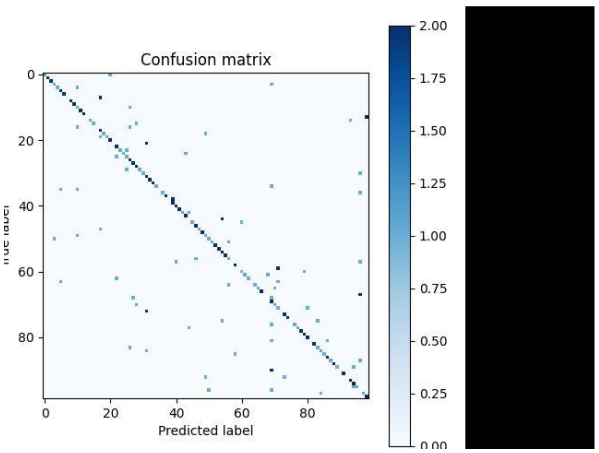

Comparaison des modèles avec Normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss

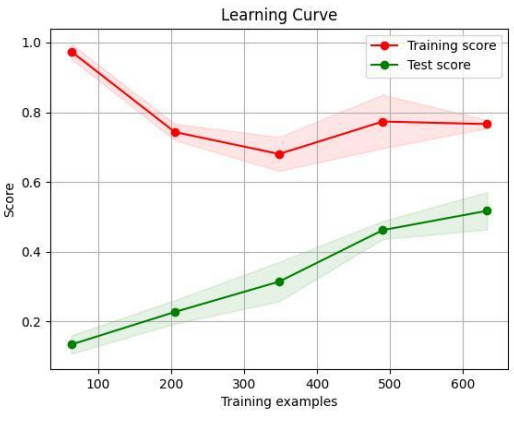

Comparaison des modèles avec No outliers and normalized
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database No outliers and normalized
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss


Comparaison des modèles avec Superclasses
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database Superclasses
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss


Comparaison des modèles avec PCA
Hyperparamètres optimaux, database standard
Hyperparamètres optimaux, database PCA
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss


AdaBoost Performance with/without hyperparameters optimization
Hyperparamètres aléatoires, database standard
Hyperparamètres optimaux, database standard
Scores
Accuracy
Mean Precision
Mean Recall
Mean F1 Score
Log Loss


Confusion matrix
Predicted label


Learning Curve
Training score
Test score
Score
Training examples