Université de Sherbrooke

IFT 603-712 Techniques d'apprentissage

Projet Kaggle

Automne 2024

Enseignant :

Toby Dylan Hocking

Réalisé par :

Achraf Haijoubi	Matricule : 24 014 567
Arthur Crochemore	Matricule : 24 120 684
Emérance Lafin	Matricule : 24 068 802
Zineb Sebti	Matricule : 24 171 469

Table des figures

Table des matières

1 Introduction générale

Dans le cadre du cours IFT 712, notre projet vise à explorer et comparer différentes approches de classification sur une base de données issue de Kaggle, en mettant l'accent sur les bonnes pratiques de validation et d'optimisation des modèles. Nous avons choisi d'analyser les performances de six méthodes de classification pour des feuilles d'arbres (kaggle) : la régression logistique, les machines à vecteurs de support (SVM), les réseaux de neurones, le perceptron, les arbres de décision et le modèle sans caractère. Nous avons choisi ce dernier afin de s'en servir comme cadre référence pour comparer les résultats des autres algorithmes.

Ce projet repose sur une démarche scientifique rigoureuse, comprenant des étapes clés de pré-traitement des données, d'ajustement des hyperparamètres, et de validation croisée pour garantir la robustesse des résultats. L'objectif est d'identifier la méthode offrant les meilleures performances pour le problème de classification choisi, tout en analysant les forces et faiblesses de chaque approche.

2 Description des données

Le jeu de données se compose d'environ 1 584 images de spécimens de feuilles, avec 16 échantillons pour chacune des 99 espèces. Les images ont été converties en noir et blanc, où les feuilles apparaissent en noir sur un fond blanc. Trois ensembles de caractéristiques sont fournis pour chaque image :

— Un descripteur de forme contiguë,
— Un histogramme de texture intérieure,
— Un histogramme de la marge à fine échelle.

Pour chacune de ces caractéristiques, un vecteur de 64 attributs est fourni par échantillon de feuille.

Pour ce projet, nous avons décidé de n'utiliser que la base de données d'entrainement train.csv. Ce fichier contient 10 individus pour chacunes des 99 classes, correspondant aux différentes espèces.

Pour ce projet, nous avons décidé de n'utiliser que la base de données d'entrainement `train.csv`. Ce fichier contient 10 individus pour chacune des 99 classes, correspondant aux différentes espèces.

Afin de préparer les données pour l'entraînement et l'évaluation, nous avons divisé l'ensemble de données en deux parties : un ensemble d'entraînement(80% des données alloué) et un ensemble de test(20% des données alloué). La division des données a été effectuée de manière stratifiée, afin de préserver la proportion des différentes classes dans les deux ensembles.

3 Démarche scientifique

3.1 Prétraitement de données

Pour ce projet, les données brutes sont d'abord mises en forme et standardisées avant d'être soumises à une analyse de réduction de dimension par l'Analyse en Composantes Principales (ACP ou PCA). Ce processus permet de réduire la complexité des données tout en conservant l'information la plus significative.

3.1.1 Mise en Forme des Données

Les données sont importées depuis un fichier CSV et la colonne identifiant de chaque échantillon est supprimée car elle inutile dans le cadre de l'apprentissage. Les étiquettes des classes sont ensuite converties en indices numériques. Ce prétraitement permet de préparer les données pour les étapes suivantes. Les variables d'entrée sont standardisées à l'aide du StandardScaler pour garantir que chaque caractéristique ait la même échelle, évitant ainsi que certaines variables dominent d'autres lors de l'analyse.

3.1.2 Réduction de Dimension par ACP

L'ACP (ou PCA en anglais) est une technique statistique utilisée pour réduire le nombre de dimensions tout en préservant la variance maximale des données. Dans notre cas, nous appliquons cette méthode pour transformer nos données en un espace de dimension inférieure, facilitant ainsi la visualisation et l'analyse des relations entre les différentes classes.

— **Standardisation des données** : Les données sont d'abord standardisées pour assurer que toutes les variables soient sur la même échelle.

— **Application de l'ACP** : L'ACP est appliquée sur les données standardisées en choisissant un nombre spécifique de composantes principales. Dans ce cas, deux ou trois composantes principales sont choisies pour une meilleure visualisation.

81	0.19823065992997008
82	0.20985072763658175
83	0.20587203016684477
84	0.20677419224727758
85	0.1803116195575481
86	0.1790848441926538
87	0.1732298973136485
88	0.16907005651992468
89	0.1613885381639548
90	0.16548211870890323
91	0.16460688262360293
92	0.1545042876595211
93	0.15140364680566018
94	0.14766083080743816
95	0.14366158877044116
96	0.14546108110479142
97	0.14147234018241883
98	0.1375341153362056
99	0.13457174192994564
100	0.13019033800666488
101	0.12197288388706967
102	0.12254245364273963
103	0.12057745151836362
104	0.11866851595023963
105	0.11667256687342072
106	0.11274260342131433
107	0.11188584731768984
108	0.10856248071239842
109	0.10438256839042093
110	0.10318017255071034
111	0.09932988498828066
112	0.09747041739627166
113	0.042703477703205636
114	0.04622580090866807
115	0.0961036453409218
116	0.09702471571290194
117	0.03536645027950932
118	0.05146796200557825
119	0.054470263920769826
120	0.08870996112367191
121	0.07666736056284126
122	0.07969518090786613
123	0.07399219082444708
124	0.08195235774961572
125	0.08586690973055852
126	0.08366485841821457
127	0.0685714351055582
128	0.06925255564043245
129	0.032455938292960104
130	0.09131730326524684
131	0.09055373372034314
132	0.08528964676714076
133	0.052105538516264004
134	0.06353133957019792
135	0.05861990659184113
136	0.061843271387905976
137	0.05981703726519706
138	0.06686214474798967
139	0.06483720214589087
140	0.028182894843307
141	0.02604685372709396
142	0.023727168269990113
143	0.01987399251410385
144	0.01809019953476784
145	0.015104782796058248
146	0.016378572984453037
147	0.010583509300972951

Fig. 3.1 – Valeurs propres pour le choix du nombre de composantes principales

Pour choisir la dimensionnalité pour notre ACP, nous avons calculé les valeurs propres. Comme le graphique contient beaucoups de caractéristique, nous avons jugé qu'il n'était pas une bonne idée de faire une ACP d'une dimensionalité inférieur à 100. Ainsi, nous avons choisi la valeur 139 car au vu du tableau ci-dessous, la valeur propre 140 contient bien moins d'information que la 139. Il s'agit donc d'un bon choix pour l'ACP.

3.1.3 Visualisation des Résultats de l'ACP

Les résultats de l'ACP sont visualisés à l'aide de nuages de points en 2D et 3D, où chaque point représente un échantillon, et les couleurs ou légendes sont associées aux classes des échantillons.

— Graphique 2D : Lorsque deux composantes principales sont utilisées, un graphique 2D est généré. Les points sont coloriés en fonction de leur classe et la variation expliquée par chaque composante est indiquée sur les axes.

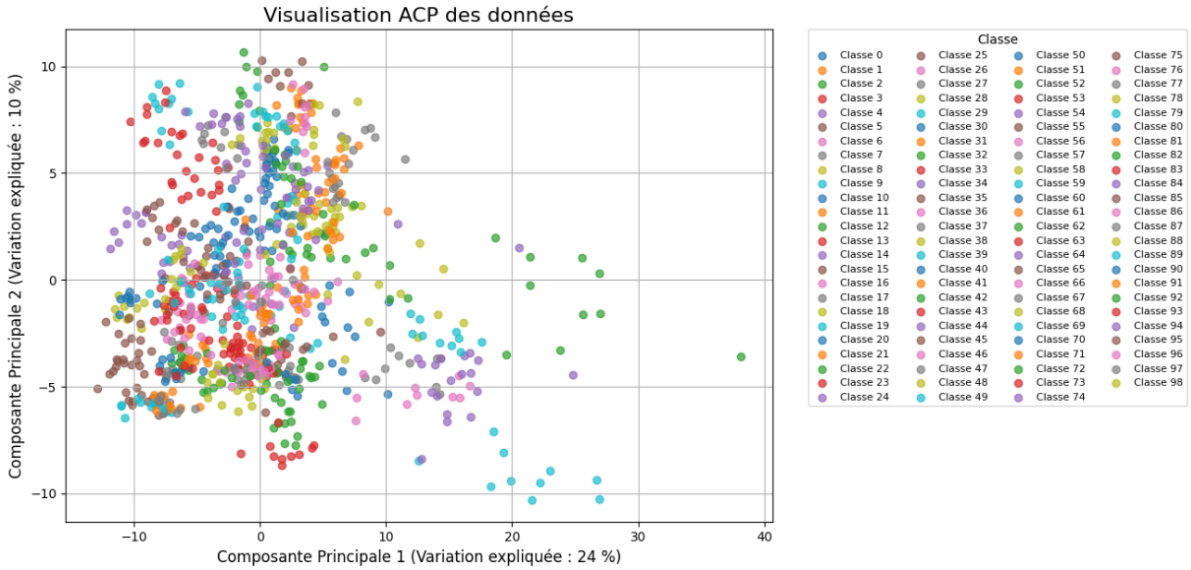


Fig. 3.2 – Visualisation ACP - 2 composantes principales

L'axe des Composantes Principales 1 (CP1) explique 24% de la variance des données. L'axe des Composantes Principales 2 (CP2) explique 10% de la variance. Ensemble, ces deux composantes expliquent environ 34% de la variance totale des données. Cela indique que même après réduction de dimension, une grande partie de l'information utile des données est conservée.
Neanmoins on observe un chevauchement important entre les points de différentes classes, suggérant que les données ne se séparent pas clairement dans ce plan 2D.

— Graphique 3D : Lorsque trois composantes principales sont utilisées, un graphique 3D est généré. Cela permet de mieux visualiser la séparation entre les classes dans l'espace tridimensionnel.

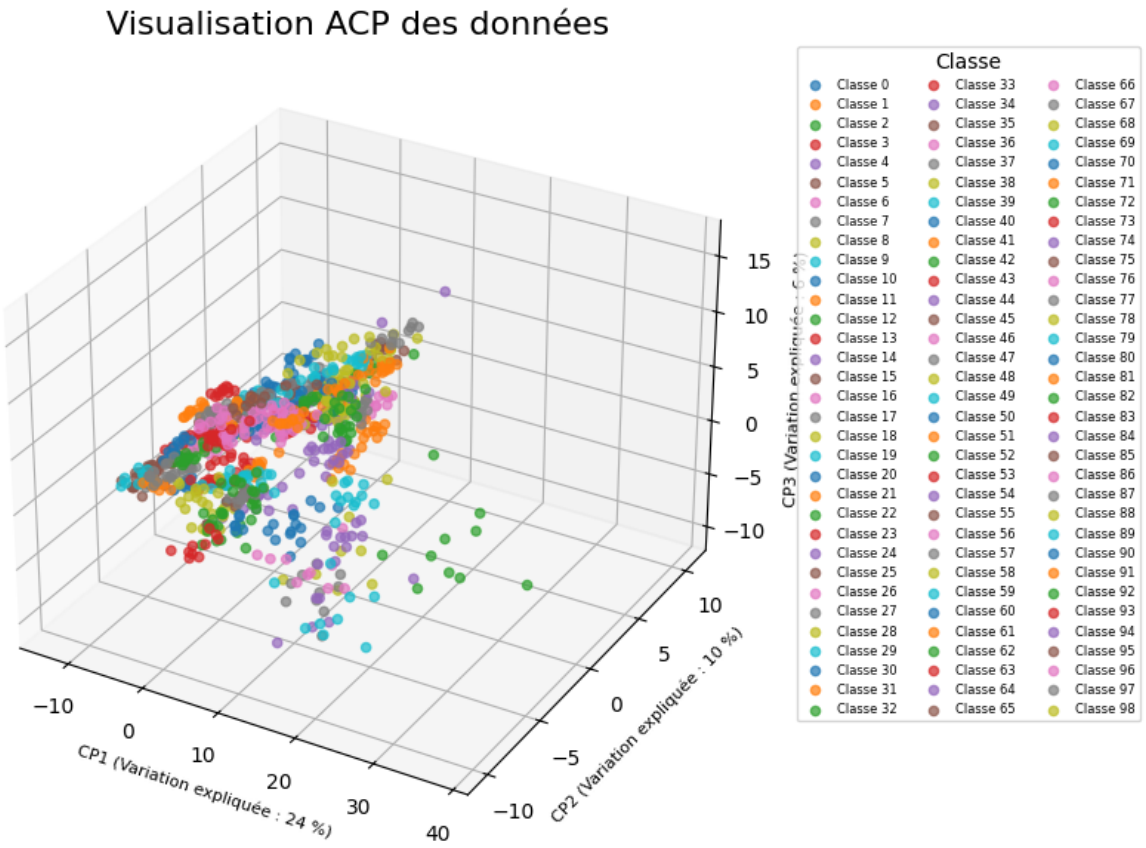


FIG. 3.3 – Visualisation ACP - 3 composantes principales

La Composante Principale 1 (CP1) explique 24 % de la variance des données. La Composante Principale 2 (CP2) explique 10 % de la variance. La Composante Principale 3 (CP3) explique 6 % de la variance. Ensemble, ces trois composantes expliquent 40 % de la variance totale, ce qui représente une amélioration par rapport à l'analyse 2D.
Cependant, on constate encore un chevauchement important entre les classes, suggérant que les espèces de feuilles ont des caractéristiques visuelles partiellement similaires.

3.2 Modèles utilisés

Nous avons utilisé exactement six modèles classifieurs, comme demandé :

— **Régression Logistique :** La Régression Logistique [4] est un modèle linéaire qui se sert de la fonction logistique pour établir la relation entre les caractéristiques X_i et la cible Y. Autrement dit, il sert à prédire la probabilité qu un événement arrive ou pas. Dans le cas multi-classes, la régression logistique de la librairie scikit-learn utilise le "one-vs-rest" pour traiter plusieurs classes, en les considérant individuellement par rapport aux autres.

— **Machines à Vecteurs de Support (SVM) :** Les SVM [9] sont des modèles puis-

sants qui cherchent à séparer les classes en maximisant la marge entre elles grâce à un hyperplan. Ils sont particulièrement utiles pour les données non linéaires, car ils permettent d'utiliser des noyaux pour transformer les données dans des espaces de plus haute dimension où elles deviennent linéairement séparables.

— **Réseaux de Neurones :** Les réseaux de neurones [5] imitent le fonctionnement du cerveau humain en utilisant des couches de neurones artificiels pour extraire des relations complexes entre les données. Ces modèles sont particulièrement performants pour capturer les relations non linéaires et peuvent être ajustés grâce à des paramètres tels que le nombre de couches ou de neurones par couche.

— **Perceptron :** Le perceptron [6] est l'un des modèles les plus simples de classification supervisée. Il s'agit d'un modèle linéaire qui ajuste les poids des caractéristiques pour trouver une frontière de décision. Bien que limité aux problèmes linéairement séparables, il constitue une base importante pour les modèles de réseaux de neurones plus complexes.

— **Arbres de Décision :** Les arbres de décision [1] sont des modèles basés sur une structure hiérarchique de décisions successives. Ils divisent les données en groupes homogènes en fonction des valeurs des caractéristiques. Leur simplicité d'interprétation et leur capacité à gérer les données catégoriques ou numériques en font une méthode populaire.

— **Sans Caracteres :** Le modèle Sans Caracteres est une implémentation simple conçue pour prédire systématiquement la classe majoritaire dans les données d'entraînement. Lors de l'entraînement, le modèle identifie la classe ayant le plus d'exemples et la retient comme classe prédite. Lorsqu'il est utilisé pour prédire de nouvelles données, il attribue toujours cette classe majoritaire à tous les échantillons. Bien que très basique, il sert souvent de point de référence pour évaluer la performance minimale acceptable par rapport à des modèles plus sophistiqués.

3.3 Recherche des hyperparamètres : GridSearch et Validation croisée

Dans le but d'optimiser nos modèles, nous avons appliqué des méthodes de recherche d'hyperparamètres. Nous avons utilisé `GridSearchCV` [3], une fonction de la bibliothèque `Scikit-learn`, qui permet de rechercher les meilleurs hyperparamètres pour un modèle d'apprentissage donné, en explorant différentes combinaisons de valeurs possibles pour certains paramètres.

Les hyperparamètres que nous avons cherché à optimiser pour chacun de nos six modèles sont les suivants :

— Arbre de décision

- **criterion** : Fonction utilisée pour mesurer la qualité d'une division (= split). Les valeurs testées sont :
  - `gini` : Gini impurity. Ce critère privilégie la séparation des classes les plus nombreuses.
  - `entropy` : information gain. On mesure la pureté d'une division basée sur la théorie de l'information de Shannon. blablanla
- **splitter** : Méthode utilisée pour diviser un nœud. Les valeurs testées sont :
  - `best` : meilleure division d'après le critère choisi dans criterion.
  - `random` : division aléatoire. Cela peut permettre d'éviter le surajustement. blablanla
- **max_depth** : Profondeur maximale de l'arbre. Cela permet de contrôler la complexité du modèle et donc le surapprentissage. Les valeurs testées sont : $max\_depth \in \{None, 5, 70\}$.
- **min_samples_split** : Nombre minimum d'échantillons requis pour diviser un nœud. Les valeurs testées sont : $min\_samples\_split \in \{2, 4, 8\}$.
- **max_features** : Nombre de fonctionnalités à prendre en compte lors de la recherche de la meilleure répartition. Par exemple, max_features = sqrt(p_features). Par défaut, max_features = p_features. Les valeurs testées sont : $max\_features \in \{None, log2, sqrt\}$.

— Régression logistique

- **C** : Paramètre de régularisation qui contrôle le compromis entre une marge plus large et les erreurs de classification. Les valeurs testées sont : $C \in \{0.01, 0.1, 1, 10\}$.
- **max_iter** : Nombre maximal d'itérations pour que les solveurs convergent. Les valeurs testées sont : $max\_iter \in \{1000, 1500, 3000, 5000\}$.
- **solver** : Algorithme à utiliser dans le problème d'optimisation. Les valeurs testées sont :
  - `saga`
  - `newton-cg`
  - `sag`

  Nous n'avons pas choisi le solveur `lbfgs` ou `liblinear` car ils sont plus appropriés pour les petits jeux de données.
- **fit_intercept** : Spécifie si une constante (également appelée biais ou interception) doit être ajoutée à la fonction de décision. Les valeurs testées sont : fit_intercept $\in \{True, False\}$.

  Nous n'avons pas utilisé **penalty** comme hyperparamètre car ajouter une régularisation apporte de meilleur résultat mais il impossible de choisir l1 car certains solver ne sont pas compatibles. Le choix de l2 par défaut est donc appliqué.

— Perceptron :

- **max_iter** : Ce paramètre contrôle le nombre maximal d'itérations pour l'algorithme d'optimisation. Nous avons testé différentes valeurs (`2000`, `3000`, `5000`) afin de déterminer le nombre d'itérations nécessaire pour une convergence optimale.

- **tol** : La tolérance pour l'arrêt de l'algorithme. Nous avons exploré les valeurs `1e-3`, `1e-4`, et `1e-5` pour évaluer leur impact sur la précision et la stabilité de la convergence.
- **penalty** : La régularisation utilisée dans le modèle. Nous avons inclus trois options :
  - `None` : Pas de régularisation, permettant au modèle de s'ajuster librement aux données.
  - `l2` : Régularisation Ridge pour réduire la complexité du modèle en pénalisant les grands coefficients.
  - `l1` : Régularisation Lasso, qui peut également conduire à une sélection de caractéristiques en mettant certains coefficients à zéro.
- **eta0** : Le taux d'apprentissage initial. Nous avons testé différentes valeurs (`0.01`, `0.1`, `1.0`, `10.0`) afin de trouver le meilleur équilibre entre vitesse de convergence et performance du modèle.
  - Pour penalty et eta0 : Nous avons décidé d'ajouter ces deux hyperparamètres afin d'améliorer la capacité d'apprentissage du perceptron. Le paramètre `penalty` permet de régulariser le modèle pour éviter le sur-apprentissage , et `eta0`, est crucial pour contrôler la vitesse à laquelle le perceptron ajuste ses poids. En variant ce paramètre, nous cherchons à optimiser la convergence du modèle, en évitant une divergence des poids avec un taux trop élevé.

— SVM :

- **kernel** : Type de noyau utilisé pour projeter les données dans un espace de dimensions plus élevées. Les valeurs testées sont :
  - `linear` : Noyau linéaire.
  - `rbf` : Noyau gaussien.
  - `poly` : Noyau polynomial.
  - `sigmoid` : Noyau sigmoïde.
- **C** : Paramètre de régularisation qui contrôle le compromis entre une marge plus large et les erreurs de classification. Les valeurs testées sont :

$$C \in \{0.001, 0.01, 0.05, 0.1, 0.2, 0.5, 1, 1.5, 2.0\}.$$

— Réseaux de neurones :

- **hidden_layer_sizes** : Nous avons cherché à varier les structures du réseau de neurones afin de tester différentes configurations, telles que des réseaux plus simples ou plus complexes, et d'observer l'impact de la taille des couches sur la performance du modèle.
- **activation** : Bien que `relu` soit généralement considéré comme le choix optimal pour les réseaux de neurones, nous avons décidé de tester d'autres fonctions d'activation, car la taille réduite de nos folds pourrait bénéficier de diverses stratégies. Cela nous a permis de comparer la performance de `logistic`, `tanh` et `relu`.
- **solver** : Le solver `adam` est souvent considéré comme le plus adapté pour des petits jeux de données, mais pour évaluer différentes approches d'optimisation, nous avons également testé le solver `sgd` (descente de gradient stochastique).

— **alpha** : Nous avons choisi de tester différentes valeurs d'alpha pour explorer l'impact de la régularisation L2 sur la performance du modèle, en particulier pour éviter le sur-apprentissage dans le contexte de notre petit jeu de données.
— **learning_rate_init** : Bien que la valeur par défaut pour `adam` soit `0.001`, nous avons décidé de tester des valeurs différentes, comme `0.01`, pour évaluer l'impact du taux d'apprentissage sur la convergence du modèle.

Nous avons effectué cette recherche avec validation croisée, utilisant 5 folds par défaut, afin d'évaluer la performance de chaque combinaison d'hyperparamètres et d'éviter le sur-apprentissage, tout en optimisant le processus d'entraînement de nos modèles.

Cependant à noter que ce nombre de folds peut changer. En effet `GridSearchCV` s'arrange pour qu'il y ait au moins un représentant de chaque classe dans ces divisions. Et un warning est donc affiché quand la division en ensemble de train et test donnent un nombre de représentant d'une classe inférieur à 5 dans le train. Afin d'éviter cela, voici comment nous avons choisi de diviser nos données :

— Au minimum on divise les données par deux pour au moins comparer le résultat des paramètres sur deux ensembles de données (un étant trop peu) ;
— Sinon on utilise le nombre d'occurence de la classe la moins représentée afin qu'elle soit présente dans chaque division de donnée ;
— Dans le dernier cas, on divise les données par 5.

4 Évaluation des Modèless

Dans ce chapitre, nous présentons les métriques utilisées pour évaluer la performance des algorithmes de classification : la **Précision**, le **Rappel** et le **Score F1**. Ces métriques sont calculées à partir de la matrice de confusion, qui résume les résultats des prédictions pour un modèle de classification.

4.1 Métriques de Classification

Les métriques de classification sont utilisès pour évaluer l'efficacité d'un modèle. Nous les définissons comme suit :

4.1.1 Précision (Accuracy)

La précision [7] mesure la proportion de prédictions positives correctes parmi toutes les prédictions positives effectuées par le modèle.

$$\text{Précision} = \frac{\text{Vrai Positifs}}{\text{Vrai Positifs} + \text{Faux Positifs}} = \frac{TP}{TP + FP}$$

Où : - TP : Nombre de vrais positifs (True Positives), - FP : Nombre de faux positifs (False Positives).

4.1.2 Rappel (Recall)

Le Rappel [8] mesure la proportion des instances positives correctement identifiées par le modèle parmi toutes les instances positives réelles.

$$\text{Rappel} = \frac{\text{Vrai Positifs}}{\text{Vrai Positifs} + \text{Faux Négatifs}} = \frac{TP}{TP + FN}$$

Où : - TP : Nombre de vrais positifs, - FN : Nombre de faux négatifs (False Negatives).

4.1.3 Score F1

Le score F1 [2] permet de mesurer la performance globale du modèle en tenant compte à la fois des faux positifs et des faux négatifs.

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

4.2 Résultats des Algorithmes de Classification

Dans cette section, nous présentons les résultats obtenus pour chaque algorithme de classification en termes de Précision, Rappel et Score F1. Nous avons effectué deux séries d'expérimentations : une avant prétraitement des données et une autre après prétraitement, comprenant une Analyse en Composantes Principales (ACP) et une régularisation. Les algorithmes évalués sont les suivants :

— Sans Caractères,

— Arbres de Décision,

— Régression Logistique,

— Support Vector Machine (SVM),

— Réseau de Neurones,

— Perceptron.

Les résultats sont présentés dans les tableaux suivants, où chaque ligne correspond à un algorithme et chaque colonne à une métrique calculée.

4.2.1 Résultats avant Prétraitement

Avant d'appliquer tout prétraitement sur les données, les performances des différents algorithmes sont présentées dans le tableau ci-dessous. Ces résultats sont obtenus directement à partir des données brutes, sans normalisation ni réduction de dimensions.

Algorithme	**Précision**	**Rappel**	**Score F1**
Sans Caractères	0.0001	0.01	0.0002
Arbres de Décision	0.64	0.62	0.60
Régression Logistique	0.82	0.79	0.77
Perceptron	0.83	0.84	0.81
SVM	0.93	0.92	0.91
Réseau de Neurones	0.90	0.89	0.88

Tab. 4.1 – Résultats des Algorithmes de Classification **avant Prétraitement**

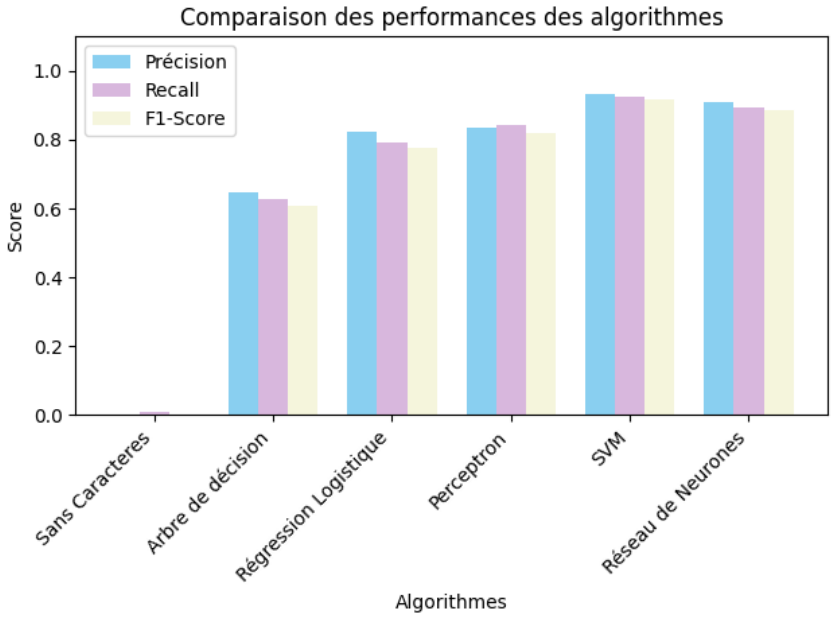


FIG. 4.1 – Titre de la figure, description de l'image.

Les hyperparamètres optimaux (parmi ceux recherchés sont les suivants) :

— Arbre de décision : {'criterion' : 'gini', 'max_depth' : None, 'max_features' : None, 'min_samples_split' : 2, 'splitter' : 'best'}
— Régression Logistique : {'C' : 10, 'fit_intercept' : True, 'max_iter' : 1000, 'solver' : 'saga'}
— Perceptron : {'eta0' : 10.0, 'max_iter' : 5000, 'penalty' : None, 'tol' : 0.0001}
— SVM : {'C' : 2.0, 'kernel' : 'rbf'}
— Réseau de Neurones : {'activation' : 'tanh', 'alpha' : 0.01, 'hidden_layer_sizes' : (100,), 'learning_rate_init' : 0.1, 'solver' : 'adam'}

4.2.2 Résultats après Prétraitement (ACP et Régularisation)

Après avoir appliqué un prétraitement sur les données, comprenant l'Analyse en Composantes Principales (ACP) pour la réduction de la dimensionnalité et une normalisation, nous avons obtenu les résultats suivants. Le prétraitement a permis d'améliorer les performances des modèles.

Algorithme	Précision	Rappel	Score F1
Sans Caractères	0.0001	0.01	0.0002
Arbres de Décision	0.96	0.95	0.95
Régression Logistique	0.99	0.99	0.99
Perceptron	0.38	0.35	0.33
SVM	1.0	1.0	1.0
Réseau de Neurones	0.99	0.98	0.98

TAB. 4.2 – Résultats des Algorithmes de Classification **après Prétraitement**

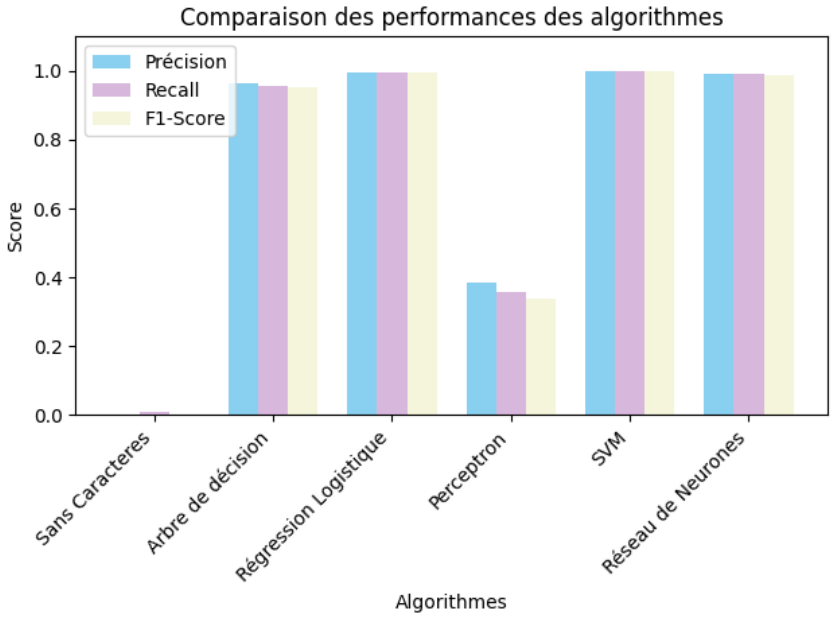


FIG. 4.2 – Titre de la figure, description de l'image.

En outre certains hyperparamètres optimaux ont changés ce qui s'explique puisque les données ont changés. Voci les changements observés :

- Arbre de décision : {'splitter' : 'best'} devient {'splitter' : 'random'}
- Régression Logistique : {'C' : 10, 'solver' : 'saga'} devient {'C' : 1, 'solver' : 'newton-cg'}
- Perceptron : {'eta0' : 10.0, 'tol' : 0.0001} devient {'eta0' : 0.01, 'tol' : 0.001}
- SVM : {'C' : 2.0, 'kernel' : 'rbf'} devient {'C' : 0.01, 'kernel' : 'linear'}
- Réseau de Neurones : {'activation' : 'tanh', 'alpha' : 0.01, 'learning_rate_init' : 0.1} devient {'activation' : 'logistic', 'alpha' : 0.0001, 'learning_rate_init' : 0.01}

En conclusion, le prétraitement des données a contribué de manière significative à améliorer les résultats des algorithmes, en particulier pour les modèles complexes comme le SVM et les Réseaux de Neurones.

4.3 Interprétation des Résultats

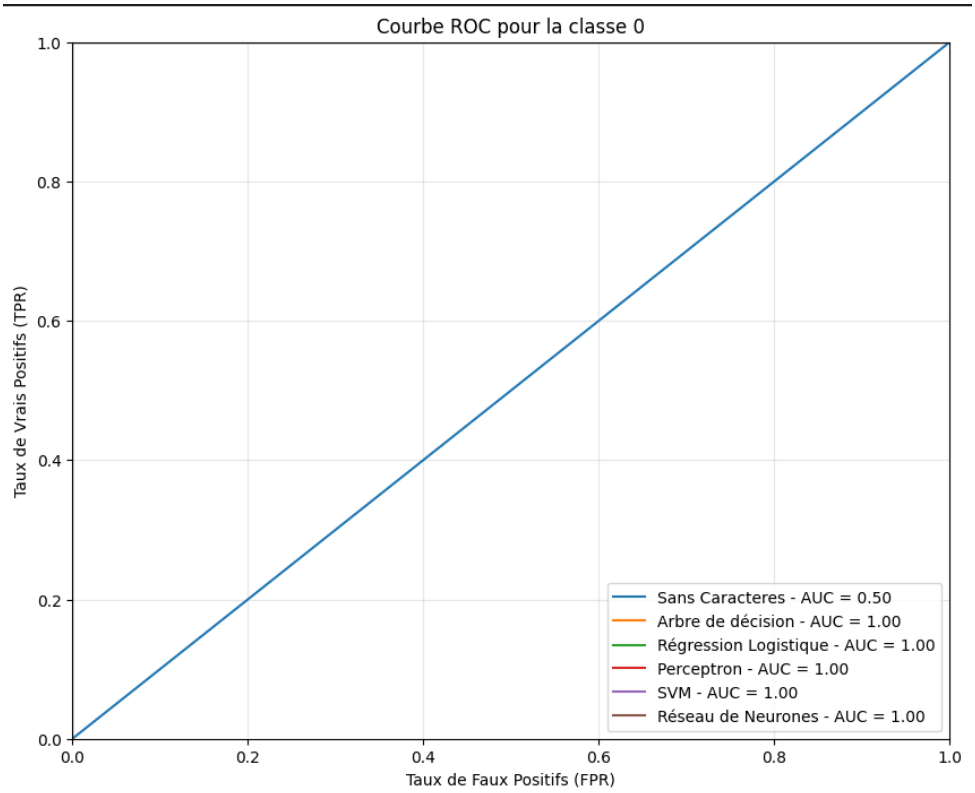
Avant le prétraitement, les performances des algorithmes sont relativement faibles, en particulier pour l'algorithme "Sans Caractères" qui obtient des scores presque nuls dans toutes les métriques (Précision = 0.0001, Rappel = 0.01, Score F1 = 0.0002). Cela est attendu, car cet algorithme ne prend pas en compte les caractéristiques des données. Parmi les autres algorithmes, les résultats sont assez variés. Les Arbres de Décision, par exemple, affichent une Précision de 0.65 et un Rappel de 0.63, ce qui montre qu'ils sont capables de réaliser une classification correcte, mais avec une capacité limitée à identifier tous les exemples pertinents. Les modèles plus avancés comme la Régression Logistique, le Perceptron, le SVM et le Réseau de Neurones montrent des performances nettement meilleures.

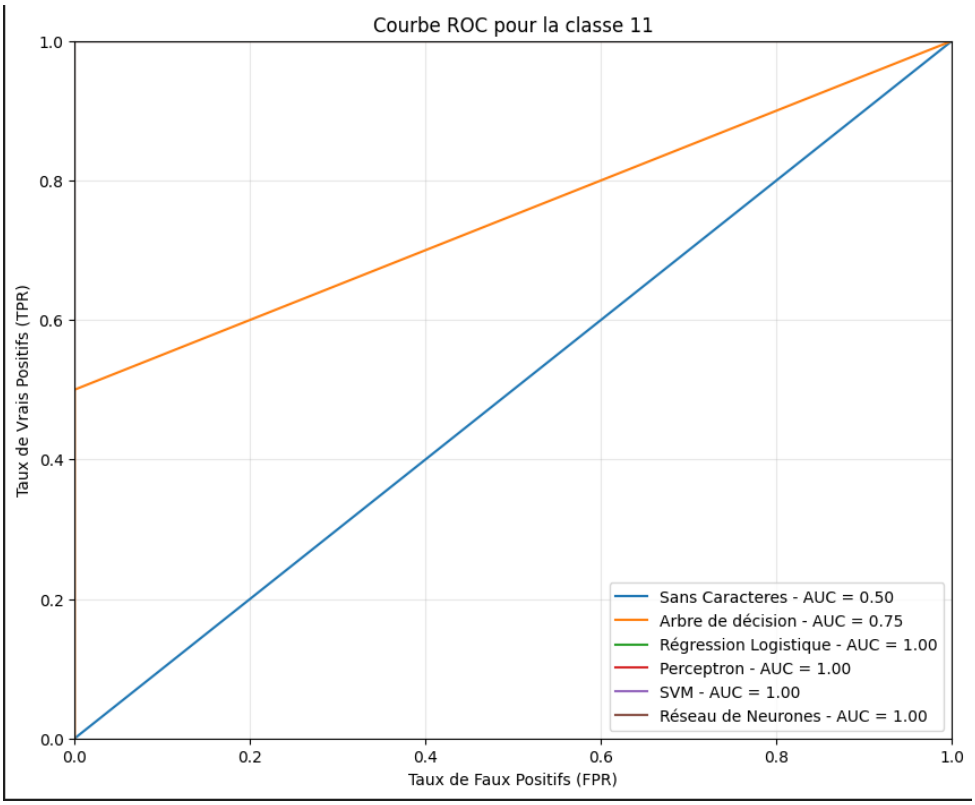
Après le prétraitement, comprenant l'ACP et la normalisation des données, les performances des algorithmes se sont sensiblement améliorées, ce qui confirme l'importance du prétraitement dans les tâches de classification. **Le SVM, qui obtient désormais un score parfait avec 1 pour sa Précision, son Rappel et son Score F1**, montre que l'ACP et la régularisation ont permis d'exploiter pleinement les données et d'éviter le surapprentissage. Le Réseau de Neurones suit de près avec des résultats impressionnants (Précision = 0.98, Rappel = 0.97, Score F1 = 0.97).

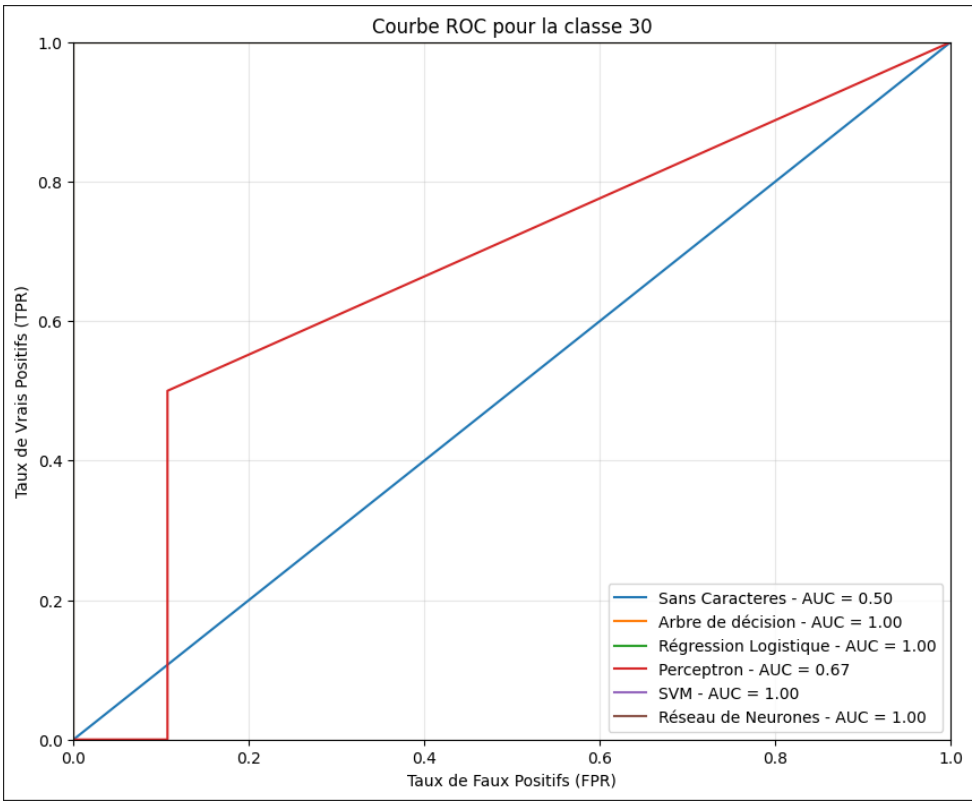
En revanche, l'algorithme Perceptron présente une performance inférieure après prétraitement, avec une Précision de seulement 0.25 et un Score F1 de 0.22. La réduction de la dimensionnalité via l'ACP pourrait avoir supprimé des informations clés, rendant ainsi ce modèle moins efficace.

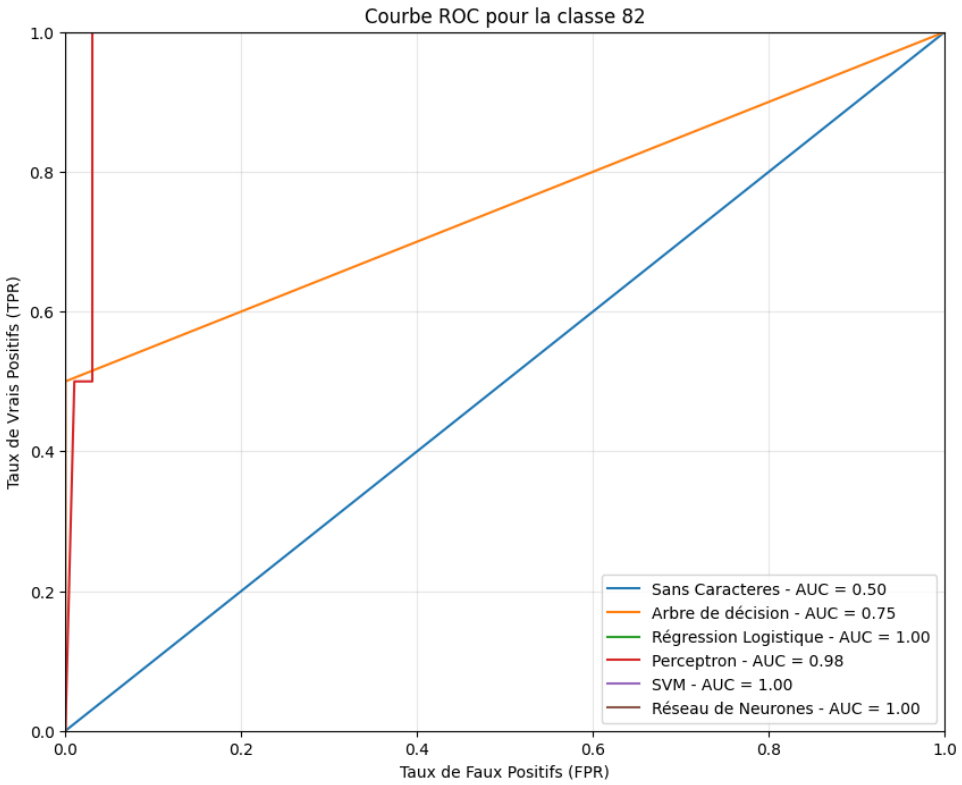
4.4 Courbes ROC

En plus des métriques ci-dessus, nous avons également tracé les courbes ROC pour différentes classes. Ces courbes, présentées dans les figures suivantes, ont été générées après le prétraitement des données. Chaque courbe montre le taux de vrais positifs (TPR) en fonction du taux de faux positifs (FPR). Nous présentons ici quelques-unes de ces courbes pour illustrer les performances des modèles.




Courbe ROC pour la classe 11
Taux de Vrais Positifs (TPR)
Taux de Faux Positifs (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Sans Caracteres - AUC = 0.50
Arbre de décision - AUC = 0.75
Régression Logistique - AUC = 1.00
Perceptron - AUC = 1.00
SVM - AUC = 1.00
Réseau de Neurones - AUC = 1.00


Courbe ROC pour la classe 30
Taux de Vrais Positifs (TPR)
Taux de Faux Positifs (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Sans Caracteres - AUC = 0.50
Arbre de décision - AUC = 1.00
Régression Logistique - AUC = 1.00
Perceptron - AUC = 0.67
SVM - AUC = 1.00
Réseau de Neurones - AUC = 1.00


Courbe ROC pour la classe 82
Taux de Vrais Positifs (TPR)
Taux de Faux Positifs (FPR)
0.0
0.2
0.4
0.6
0.8
1.0
Sans Caracteres - AUC = 0.50
Arbre de décision - AUC = 0.75
Régression Logistique - AUC = 1.00
Perceptron - AUC = 0.98
SVM - AUC = 1.00
Réseau de Neurones - AUC = 1.00

5 Gestion de projet et Choix de design

5.1 Gestion du projet avec Git et GitLab

Travailler en équipe nécessite une collaboration efficace pour garantir la cohérence et le bon fonctionnement du code. Dans ce cadre, nous avons utilisé `Git` via la plateforme `GitLab`, conformément aux consignes du cours. Cette méthodologie de gestion de version nous a permis de centraliser nos efforts et de maintenir un flux de travail structuré.

Chaque membre de l'équipe disposait de sa propre branche où il pouvait effectuer des modifications et des expérimentations sans impacter le travail des autres. Une fois les changements terminés, ceux-ci étaient validés via des *commits* et soumis sous forme de *merge requests*, Pour etre intégrées dans la branche principale (`main`), garantissant ainsi la qualité et la compatibilité des contributions.

Pour maintenir une qualité de code homogène, nous avons également veillé à respecter les normes de style définies par **PEP8**. Cette rigueur a permis d'assurer une base de code propre, lisible et facile à maintenir, favorisant ainsi une collaboration fluide et efficace.

5.2 Choix de design

Pour structurer notre projet, nous avons adopté une hiérarchie de classes et une organisation de fichiers cohérentes, facilitant la collaboration et la modularité. Chaque fichier remplit une fonction spécifique, rendant le projet clair et facilement extensible. Voici la structure des fichiers du projet :

```
Agenda_d_equipe.xlsx
Document_d_integrite
Arbre_de_decision.py
evaluation.py
main.ipynb
perceptron_classification.py
pretraitement.py
regression_logistic.py
reseau_de_neuronnes.py
sans_caracteres.py
```

```
svm.py
train.csv
```

— **Fichiers de classification** : Chaque algorithme de classification possède son propre fichier Python (`Arbre_de_decision.py`, `svm.py`, etc.), contenant à la fois l'implémentation et la recherche des hyperparamètres. Cette séparation permet de tester et de modifier indépendamment chaque modèle.

— **`pretraitement.py`** : Ce fichier regroupe toutes les étapes de préparation des données, notamment la mise en forme et la réduction de dimension par ACP. Cela garantit une centralisation des opérations de prétraitement, évitant la duplication de code.

— **`evaluation.py`** : Ce fichier est dédié à l'évaluation des modèles. Il contient les implémentations des métriques de classification, ainsi que les fonctions pour générer les courbes ROC et les rapports de performance.

— **`main.ipynb`** : Le fichier principal est un notebook qui orchestre l'exécution des différentes étapes du projet, depuis le prétraitement des données jusqu'à la comparaison des performances des modèles.

— **`train.csv`** : Ce fichier contient les données d'entraînement, servant de base à l'apprentissage des modèles.

— **Autres fichiers** : `Agenda_d_equipe.xlsx` et `Document_d_integrite` témoignent de la coordination et de l'intégrité de l'équipe durant le projet.

Ce design modulaire et hiérarchique a permis une gestion efficace du projet, en facilitant la collaboration entre les membres de l'équipe et en rendant le code plus facile à comprendre, à tester et à maintenir.

6 Conclusion

Ce projet a permis de mettre en œuvre une démarche scientifique rigoureuse pour analyser et comparer les performances de six méthodes de classification sur une base de données d'identification de feuilles d'arbres. En explorant des approches telles que la régression logistique, les machines à vecteurs de support, les réseaux de neurones, le perceptron, les arbres de décision et un modèle de référence sans Caracteres, nous avons étudié l'impact des prétraitements (réduction de dimension par ACP et visualisation), ainsi que l'optimisation des hyperparamètres via GridSearchCV et la validation croisée.

L'évaluation des modèles a été réalisée en utilisant des métriques standard telles que la précision, le rappel et le score F1, complétées par des analyses visuelles comme les courbes ROC. Les résultats obtenus ont permis de mettre en évidence les forces et faiblesses de chaque approche, tout en démontrant l'importance des étapes de prétraitement et de réglage des hyperparamètres dans le processus de classification.

En conclusion, ce travail a non seulement renforcé notre compréhension des différentes méthodes de classification, mais a également souligné la nécessité d'une analyse approfondie et structurée pour tirer des conclusions robustes dans des contextes de données complexes. Ces apprentissages serviront de base pour des travaux futurs, où des améliorations pourront être apportées, notamment en testant des bases de données plus variées ou en explorant des modèles encore plus avancés.

Bibliographie

[1] *Decision Trees*. URL : https://scikit-learn.org/stable/modules/tree.html.

[2] *f1 Score*. URL : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.

[3] *GridSearchCV*. URL : https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.GridSearchCV.h.

[4] *Logistic Regression*. URL : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html.

[5] *Neural network models*. URL : https://scikit-learn.org/stable/modules/neural_networks_supervised.html.

[6] *Perceptron*. URL : https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html.

[7] *precision Score*. URL : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html.

[8] *recall Score*. URL : https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html.

[9] *Support Vector Machines*. URL : https://scikit-learn.org/stable/modules/linear_model.html.