Université de Sherbrooke

IFT 603-712 : Techniques d'apprentissage

Projet de Session

Automne 2024

Ce travail est réalisé par:

Nom	**Courriel**	**Matricule**
Nathan Beaujean	nathan.beaujean@usherbrooke.ca	24 150 669
Fatima-Ezzahra Bour	fatima-ezzahra.bour@usherbrooke.ca	24 171 487
Aya Errafik	Aya.Errafik@USherbrooke.ca	24 171 441
Selly Bill-Gate Medewou	selly.bill-gate.medewou@usherbrooke.ca	24 172 663

Faculté des Sciences,
Département d'informatique

Table des figures

Table des matières

1 Introduction

Dans un contexte scientifique où la compréhension et la classification des objets célestes jouent un rôle clé, les techniques d'apprentissage automatique offrent des outils puissants pour analyser de vastes ensembles de données astronomiques. Ce projet s'inscrit dans cette dynamique en utilisant la base de données **Stellar Classification Dataset SDSS17**, issue des relevés du Sloan Digital Sky Survey.

L'objectif principal est de comparer les performances de différents algorithmes de classification implémentés via la bibliothèque **Scikit-learn**, dans la tâche de catégorisation des objets célestes en étoiles, galaxies et quasars. En nous appuyant sur une démarche méthodologique rigoureuse, nous explorons diverses métriques pour évaluer l'efficacité des modèles, tout en discutant leurs forces et leurs limites.

Le document se structure comme suit : après une description détaillée des données utilisées, nous présentons la conception du projet et la démarche suivie pour comparer les modèles. Enfin, les résultats obtenus sont discutés avant de conclure sur les perspectives et les enseignements tirés de cette étude.

2 Description des données

La base de données **Stellar Classification Dataset SDSS17** provient du célèbre relevé astronomique Sloan Digital Sky Survey (SDSS). Elle contient des informations sur différents types d'objets célestes, tels que des étoiles, des galaxies et des quasars, répartis en plusieurs catégories selon leurs caractéristiques spectroscopiques et photométriques.

Les données se composent des caractéristiques suivantes :

— **u, g, r, i, z** : Les magnitudes photométriques dans différentes bandes spectrales (de l'ultraviolet à l'infrarouge).

— **redshift** : Le décalage spectral, qui fournit des informations sur la vitesse et la distance des objets.

— **spec_class** : La classe spectrale de l'objet, qui constitue la variable cible (étoile, galaxie ou quasar).

— **plate, mjd, fiberid** : Identifiants techniques liés à l'observation astronomique.

La répartition des objets selon leur classe est illustrée dans la Figure 2.1. On observe que les galaxies représentent la majorité des objets (**59.4%**), suivies par les étoiles (**21.6%**) et les quasars (**19.0%**).

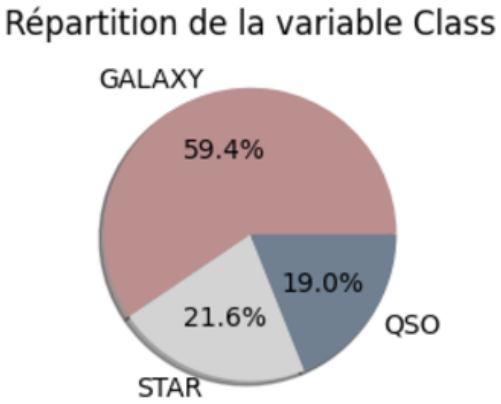


Fig. 2.1 – Répartition de la variable `Class`

Avec plus de **100 000 échantillons**, cette base de données constitue un ensemble riche et varié pour expérimenter différents algorithmes de classification. Elle permet de comparer leurs performances sur une tâche réelle et scientifiquement significative.

3 Design du code

Le projet repose sur une approche systématique de prétraitement et de visualisation des données afin de mieux comprendre et préparer le jeu de données pour des analyses ou des modèles de machine learning. Le design du projet a été divisé en trois modules principaux, chacun ayant une responsabilité distincte mais complémentaire.

3.1 Module de Prétraitement des Données

Le module `Preprocessing` gère les étapes essentielles de préparation des données avant toute analyse ou modélisation. Il se divise en plusieurs sous-processus pour assurer une gestion optimale du jeu de données :

- **Chargement et Exploration des Données :** Le fichier CSV est chargé dans un DataFrame Pandas, et des méthodes permettent de visualiser les premières lignes, les types de données et les statistiques descriptives du jeu de données.
- **Traitement des Valeurs Manquantes et Duplicates :** Ce module détecte et gère les valeurs manquantes et les lignes dupliquées. Les valeurs manquantes sont supprimées et les doublons sont également éliminés.
- **Encodage des Variables Catégorielles :** Des colonnes catégorielles sont identifiées et encodées en utilisant des `LabelEncoder`, afin de convertir les données qualitatives en valeurs numériques.
- **Analyse des Corrélations :** Un mécanisme d'identification des paires de variables fortement corrélées est mis en place. Si des corrélations supérieures à un seuil défini (par défaut, 0.7) sont trouvées, une ou les deux variables corrélées sont supprimées pour éviter la multicolinéarité.
- **Normalisation des Données :** Un processus de normalisation est appliqué aux colonnes numériques pour garantir que les données soient sur une échelle comparable, ce qui est essentiel pour les modèles sensibles à l'échelle comme les réseaux de neurones ou les SVMs.
- **Division des Données en Ensembles d'Entraînement et de Test :** Le module sépare ensuite le jeu de données en ensembles d'entraînement et de test en utilisant la fonction `train_test_split` de Scikit-learn, avec un échantillonnage stratifié basé sur la cible.

3.2 Module de Visualisation des Données

Le module `Visualize` est conçu pour fournir des représentations graphiques permettant d'explorer visuellement les données et les relations entre les différentes variables :

- **Matrice de Corrélation :** Une heatmap de la matrice de corrélation est générée pour les variables sélectionnées, ce qui permet de visualiser les relations linéaires entre les caractéristiques et d'identifier les variables fortement corrélées.
- **Histogrammes et KDE :** Le module permet de visualiser la distribution des variables à l'aide d'histogrammes et de graphiques de densité (KDE). Cette étape est utile pour comprendre la distribution des données et détecter les éventuelles anomalies.
- **Boxplots :** Des boxplots sont utilisés pour visualiser les valeurs aberrantes (outliers) dans les colonnes numériques. Cela permet de détecter les valeurs extrêmes et d'évaluer si un traitement est nécessaire avant la modélisation.
- **Outliers :** Un graphique montre le pourcentage d'outliers pour chaque variable, ce qui peut aider à prendre des décisions sur la gestion de ces valeurs extrêmes (par exemple, suppression ou transformation).
- **Relations entre Variables :** Un graphique KDE permet de visualiser la distribution des variables en utilisant la transformation logarithmique, permettant d'examiner les relations entre certaines caractéristiques.

3.3 Module d'Entraînement des Modèles

Le module `Training` gère l'entraînement, l'évaluation des performances et l'optimisation des modèles de machine learning. Il prend en charge plusieurs modèles de classification, dont `LogisticRegression`, `DecisionTreeClassifier`, `RandomForestClassifier`, `KNeighborsClassifier`, `SVC`, `AdaBoostClassifier` et des pipelines pour la régression polynomiale.

- **Initialisation des Modèles :** Les modèles sont initialisés avec des paramètres par défaut, mais peuvent être ajustés par la suite.
- **Entraînement des Modèles :** Chaque modèle peut être entraîné sur les données d'entraînement avec la méthode `train_model`, qui enregistre la précision sur l'ensemble de test.
- **Entraînement avec Validation Croisée :** La méthode `train_with_cross_validation` permet d'effectuer une validation croisée `k`-fold et d'évaluer la performance sur l'ensemble de test.
- **Affichage des Matrices de Confusion :** Les matrices de confusion des modèles peuvent être visualisées grâce à `plot_confusion_matrix`.
- **Courbes d'Erreur :** La méthode `plot_error_curves` permet de comparer l'erreur des différents modèles.
- **Courbes d'Apprentissage :** Les courbes d'apprentissage sont tracées pour évaluer l'évolution de la performance en fonction de la taille de l'échantillon d'entraînement via `plot_learning_curve`.

— **Optimisation des Hyperparamètres :** `GridSearchCV` est utilisé pour optimiser les hyperparamètres des modèles.
— **Évaluation des Modèles :** Les performances des modèles sont évaluées avec des métriques telles que la précision, le rappel, le score F1 et l'accuracy.

3.4 Interaction et Flexibilité

Les modules sont conçus pour être exécutés à partir d'un notebook, offrant une interface interactive où l'utilisateur peut tester différentes étapes du prétraitement, explorer les visualisations des données et observer les résultats immédiatement. La possibilité d'exclure certaines colonnes dans l'analyse de corrélation ou de personnaliser les colonnes utilisées dans les visualisations est intégrée pour permettre une grande flexibilité d'analyse.

4 Démarche scientifique

La démarche scientifique que nous avons suivie dans ce projet repose sur une série d'étapes méthodiques visant à garantir la qualité et la pertinence des données pour une analyse approfondie. L'objectif principal est de préparer et explorer les données, puis de les analyser de manière à en extraire des informations utiles pour la modélisation ou toute autre forme de traitement avancé. Ce processus inclut la gestion des données manquantes, la visualisation des relations entre les variables, et la transformation des données brutes en une forme plus exploitable.

4.1 Collecte et Exploration des Données

La première étape du projet consiste à charger le dataset et à en réaliser une exploration initiale. Nous avons importé les données sous forme de fichier CSV, puis nous avons utilisé différentes fonctions pour avoir un aperçu rapide du contenu : `head()`, `info()`, et `describe()`. Ces premières explorations nous ont permis de vérifier la structure du dataset, d'identifier les types de données dans chaque colonne, et de repérer les éventuelles valeurs manquantes ou les anomalies.

Cette étape est cruciale pour comprendre les caractéristiques des données, comme la répartition des valeurs, les relations évidentes entre certaines variables et la présence d'éventuels problèmes comme des duplications de lignes ou des valeurs incohérentes.

4.2 Prétraitement des Données

Une fois l'exploration réalisée, le prétraitement des données est essentiel pour préparer un dataset propre et exploitable. Cette phase comprend plusieurs étapes clés :

— **Gestion des Valeurs Manquantes et Duplications** : En identifiant et supprimant les valeurs manquantes ou les doublons, nous assurons que le dataset soit cohérent et ne soit pas biaisé par des données absentes ou redondantes.

— **Encodage des Variables Catégorielles** : Puisque notre dataset contient des variables catégorielles, nous avons utilisé des techniques d'encodage (comme le `LabelEncoder`) pour les convertir en valeurs numériques. Cela permet aux modèles de machine learning de les traiter de manière plus efficace.

```
Quantity of null values: obj_ID         0
alpha           0
delta           0
u               0
g               0
r               0
i               0
z               0
run_ID          0
rerun_ID        0
cam_col         0
field_ID        0
spec_obj_ID     0
class           0
redshift        0
plate           0
MJD             0
fiber_ID        0
dtype: int64

Quantity of duplicated values: 0
```

FIG. 4.1 – Résultat de la vérification des valeurs manquantes.

4.3 Visualisation des Données

Après le prétraitement, il est crucial de visualiser les données sous différents angles pour mieux comprendre leurs relations et détecter d'éventuelles anomalies. Nous avons utilisé plusieurs outils de visualisation pour explorer la distribution des variables et les corrélations entre elles.

— **Histograms et KDE (Kernel Density Estimation)** : Pour chaque variable numérique, nous avons tracé des histogrammes et des courbes KDE afin de visualiser la répartition des données. Cela nous a permis de repérer d'éventuelles distributions non uniformes ou des anomalies dans les données.

— **Boxplots** : Les boxplots ont été utilisés pour détecter les valeurs aberrantes (outliers). En identifiant ces valeurs extrêmes, nous avons pu décider si des traitements

(comme la transformation ou la suppression) étaient nécessaires pour éviter qu'elles n'influencent négativement l'analyse.

— **Matrices de Corrélation** : Une fois les données prétraitées, nous avons exploré les relations entre les variables à travers des matrices de corrélation. Cela nous a permis de mieux comprendre les interactions entre les différentes caractéristiques et d'identifier celles qui étaient fortement liées entre elles.

4.4 Suppression des Variables Corrélées

Dans cette phase, nous avons analysé les corrélations entre les variables afin de détecter les relations fortes entre certaines caractéristiques. Lorsqu'une corrélation est jugée trop élevée (par exemple, au-dessus de 0.7), nous avons opté pour la suppression de l'une des variables afin de réduire les risques de multicolinéarité, qui pourrait perturber certaines analyses statistiques ou modèles.

Nous avons supprimé les variables `['run_ID', 'obj_ID', 'u', 'g', 'z', 'rerun_ID']` après une analyse approfondie de leur pertinence pour le modèle. Voici le détail de cette suppression :

4.4.1 Analyse de `run_ID` et `obj_ID`

Pour les variables `run_ID` et `obj_ID`, nous avons :

— Tracé les histogrammes de distribution, montrant des distributions similaires. Voir la figure 4.2 et la figure 4.3.

— Vérifié leur linéarité avec un scatter plot (figure 4.4), qui indique une relation linéaire.

— Analysé la matrice de corrélation (figure 4.5), montrant des valeurs très faibles (proches de 0), indiquant une très faible corrélation avec la variable cible.

Conclusion : Ces deux variables présentent une faible relation ou une relation inexistante avec la variable cible `class`, ce qui justifie leur suppression.

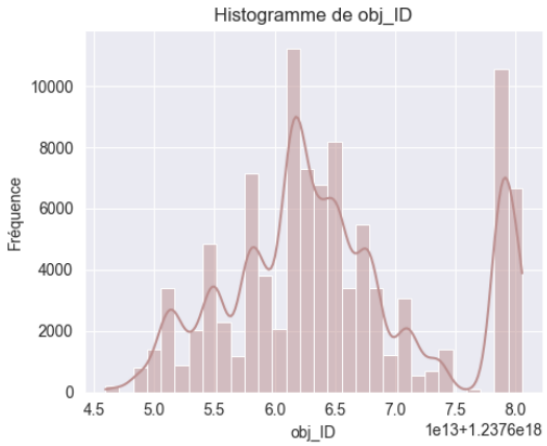


FIG. 4.2 – Histogramme de distribution de `obj_ID`.

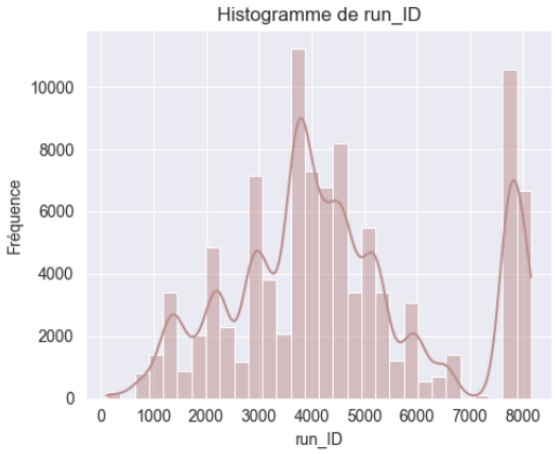


FIG. 4.3 – Histogramme de distribution de `run_ID`.

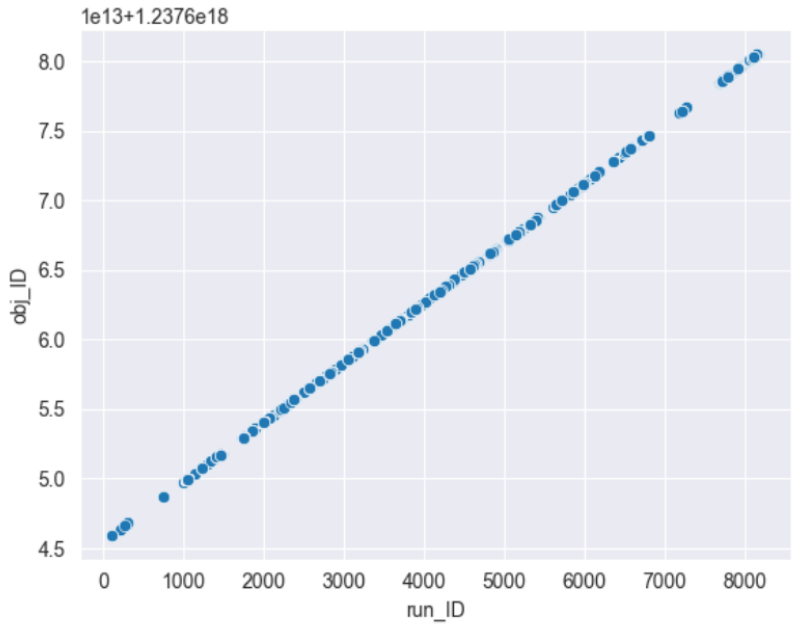


FIG. 4.4 – Scatter plot entre `run_ID` et `obj_ID`.

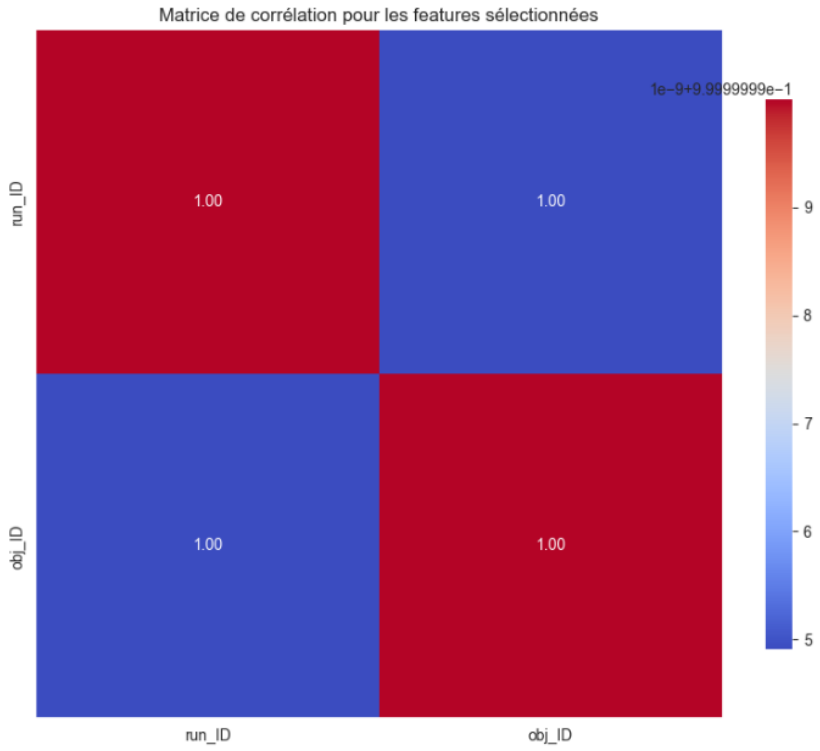


FIG. 4.5 – Matrice de corrélation pour `run_ID` et `obj_ID`.

4.4.2 Analyse des Variables `u`, `g`, `z`

Pour les variables `u`, `g`, et `z`, nous avons :

— Tracé des scatter plots entre les paires de variables (`u, g`, `g, z`, et `u, z`) (**??**). Ces graphes montrent une densité insuffisante pour conclure sur une relation.

— Calculé la matrice de corrélation (figure 4.6), montrant des valeurs proches de 1 entre les variables, ce qui suggère une forte redondance d'information.

— Calculé la corrélation avec la variable cible (`class`), avec des valeurs très faibles (proches de 0) comme suit :

```
u : -0.024645
g : -0.020066
z : -0.001614
```

Conclusion : Ces variables présentent une faible relation ou une relation inexistante avec la variable cible et une redondance entre elles, ce qui justifie leur suppression.

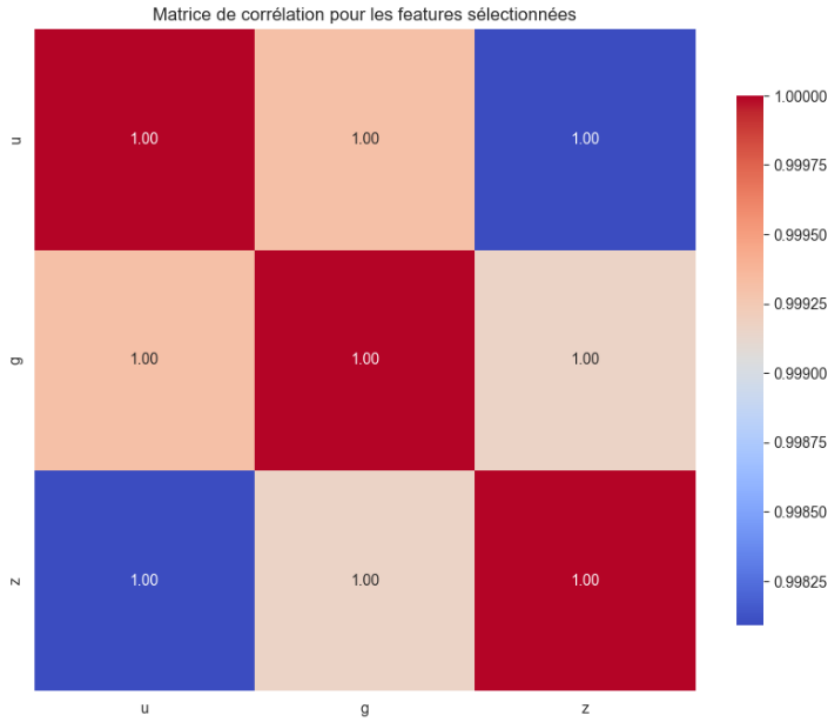


FIG. 4.6 – Matrice de corrélation pour les variables `u`, `g`, et `z`.

4.5 Modèles Utilisés et Justification du Choix

Dans ce projet, plusieurs modèles de machine learning ont été utilisés pour effectuer la classification des données. Chaque modèle a été choisi en fonction de ses caractéristiques spécifiques et de sa pertinence pour le dataset. Voici une présentation détaillée des modèles employés :

4.5.1 Régression Logistique (Logistic Regression)

La régression logistique est un modèle de classification linéaire qui estime la probabilité d'appartenance à une classe à partir d'une combinaison linéaire des caractéristiques. Ce modèle est simple, rapide à entraîner et très efficace pour les problèmes où les relations entre les variables sont linéaires.

Pourquoi ce modèle ? La régression logistique a été choisie pour sa simplicité et son efficacité. Elle est idéale lorsque les relations entre les variables sont linéaires, et elle offre également une bonne interprétabilité grâce à ses coefficients. Cela permet de mieux comprendre l'impact de chaque caractéristique sur la prédiction.

4.5.2 Arbre de Décision (Decision Tree)

Un arbre de décision est un modèle non linéaire qui divise les données en fonction de critères qui maximisent l'information sur la classe cible. Il est particulièrement adapté

pour gérer des relations complexes entre les variables, qu'elles soient numériques ou catégorielles.

Pourquoi ce modèle ? L'arbre de décision a été choisi car il permet de capturer des relations complexes et non linéaires. De plus, il est facile à interpréter et à visualiser, ce qui le rend utile pour comprendre les décisions prises par le modèle. Ce modèle est également très flexible et peut être utilisé sur différents types de données.

4.5.3 Forêt Aléatoire (Random Forest)

La forêt aléatoire est un modèle d'ensemble qui combine plusieurs arbres de décision pour améliorer la robustesse du modèle. En construisant plusieurs arbres sur des sous-ensembles aléatoires des données, la forêt aléatoire réduit la variance et le sur-apprentissage, tout en capturant des relations complexes.

Pourquoi ce modèle ? La forêt aléatoire a été sélectionnée pour sa capacité à combiner plusieurs arbres de décision et à réduire le risque de sur-apprentissage, tout en maintenant une bonne précision. Elle est particulièrement adaptée aux jeux de données avec de nombreuses caractéristiques et interactions complexes entre elles.

4.5.4 Machine à Vecteurs de Support (SVC - Support Vector Classifier)

Le SVC est un modèle basé sur des hyperplans qui séparent les données en différentes classes. En utilisant un noyau (kernel), ce modèle peut gérer des séparations non linéaires et est particulièrement efficace pour des espaces de données de haute dimension.

Pourquoi ce modèle ? Le SVC a été choisi pour sa capacité à trouver des frontières de décision complexes dans des espaces à haute dimension. Il est particulièrement utile lorsque les classes ne sont pas linéairement séparables, et son efficacité face aux données bruitées le rend adapté pour des problèmes de classification difficiles.

4.5.5 AdaBoost (Adaptive Boosting)

AdaBoost est une méthode d'assemblage qui combine plusieurs classificateurs faibles pour créer un classificateur plus puissant. En ajustant les poids des instances mal classées, AdaBoost améliore la performance générale du modèle.

Pourquoi ce modèle ? AdaBoost a été choisi car il permet d'améliorer les performances des modèles faibles en les combinant de manière efficace. C'est un modèle puissant qui, même avec peu d'exemples d'entraînement, peut produire de bons résultats en réduisant les erreurs de classification.

4.5.6 K-Plus Proches Voisins (KNN - K-Nearest Neighbors)

Le KNN est un modèle de classification basé sur la proximité des points de données. Pour chaque instance, le modèle regarde les k points les plus proches et prédit la classe

la plus fréquente parmi eux.

Pourquoi ce modèle ? Le KNN a été choisi pour sa simplicité et son efficacité lorsqu'il s'agit de classer des données sans faire d'hypothèses sur leur distribution. Ce modèle est particulièrement utile lorsque les relations entre les variables sont complexes et difficiles à modéliser directement. De plus, il ne nécessite pas de phase d'entraînement, ce qui peut être un avantage dans certains cas.

4.6 Recherche d'Hyperparamètres avec GridSearchCV

Pour optimiser les performances des classifieurs utilisés dans ce projet, nous avons effectué une recherche exhaustive des hyperparamètres à l'aide de la méthode `GridSearchCV`. Cette méthode permet d'explorer un espace d'hyperparamètres afin de trouver la combinaison optimale qui maximise les performances du modèle. Pour chaque modèle, nous avons défini un ensemble d'hyperparamètres possibles, que nous avons testés via une validation croisée. Les résultats obtenus ont ensuite été utilisés pour sélectionner le meilleur modèle pour la classification.

4.6.1 Méthodologie

Le processus de recherche des hyperparamètres repose sur la grille de recherche (`grid search`) qui explore toutes les combinaisons possibles des paramètres spécifiés. Nous avons utilisé `GridSearchCV` de la bibliothèque `scikit-learn`, qui permet de tester différentes configurations tout en utilisant une validation croisée pour évaluer la performance de chaque combinaison de paramètres. Cette méthode garantit une recherche exhaustive tout en minimisant le risque de sur-ajustement (overfitting) grâce à la validation croisée.

4.6.2 Classifieurs et Hyperparamètres Testés

Nous avons testé les modèles suivants avec différentes valeurs pour leurs hyperparamètres, accompagnées des justifications correspondantes :

Classifieur	**Hyperparamètre**	**Justification**
Régression Logistique	C : {0.01, 0.1, 1, 10}	Contrôle la régularisation en ajustant l'intensité du terme de pénalisation : des valeurs faibles favorisent un modèle plus régulier pour éviter le sur-apprentissage, tandis que des valeurs élevées permettent une meilleure adaptation aux données complexes.
	solver : {'liblinear', 'lbfgs', 'newton-cg'}	Ces solveurs sont adaptés à des contextes différents : *liblinear* pour les petits ensembles de données, *lbfgs* et *newton-cg* pour des scénarios plus complexes ou des grands ensembles de données.

Classifieur	Hyperparamètre	Justification
Arbre de Décision	criterion : {'gini', 'entropy'}	Le critère gère la précision et la performance du modèle. Gini est standard et rapide, tandis qu'Entropy fournit des informations plus riches sur l'impureté des nœuds.
	max_depth : {5, 10, 15, None}	Ce paramètre contrôle la profondeur maximale de l'arbre pour limiter le sur-ajustement dans les ensembles de données complexes ou petits.
	min_samples_split : {2, 5, 10}	Contrôle le nombre minimal d'échantillons nécessaires pour diviser un nœud, réduisant le risque de divisions inutiles ou de sur-ajustement.
Forêt Aléatoire	n_estimators : {50, 100, 200}	Définit le nombre d'arbres dans la forêt. Plus il y a d'arbres, meilleure est la stabilité du modèle, au prix d'une augmentation du coût de calcul.
	max_depth : {10, 20, 30, None}	Contrôle la profondeur maximale des arbres individuels pour éviter une complexité inutile et un sur-ajustement.
	min_samples_split : {2, 5, 10}	Assure qu'un nœud possède un minimum d'échantillons pour être divisé, ce qui évite la sur-segmentation et améliore la généralisation.
SVC	C : {0.1, 1, 10}	Définit un compromis entre la maximisation de la marge et la minimisation des erreurs de classification. Une faible valeur favorise une marge plus large (moins de sur-ajustement).
	kernel : {'linear', 'rbf', 'poly'}	Le noyau *linear* est adapté pour des données linéairement séparables, tandis que *rbf* et *poly* gèrent les non-linéarités complexes des données.
	gamma : {'scale', 'auto'}	Ajuste l'influence des points de données sur la séparation dans les noyaux non-linéaires. *Scale* est recommandé pour des données standardisées.
Adaboost	n_estimators : {50, 100, 200}	Définit le nombre de classifieurs faibles à utiliser. Un plus grand nombre améliore la robustesse mais augmente le temps de calcul.
	learning_rate : {0.01, 0.1, 1}	Ajuste l'importance accordée à chaque classifieur faible lors de la construction du modèle final.

Classifieur	Hyperparamètre	Justification
KNN	n_neighbors : {3, 5, 7}	Définit le nombre de voisins à prendre en compte pour le vote majoritaire ou la moyenne des distances, influençant ainsi la précision du modèle.

4.6.3 Résultats et Sélection du Meilleur Modèle

Chaque modèle a été évalué en utilisant la validation croisée et le score de précision (accuracy) comme critère de performance. Une fois la grille d'hyperparamètres explorée, le modèle avec la meilleure combinaison d'hyperparamètres a été sélectionné pour l'entraînement final. Cette étape permet d'améliorer les performances des modèles en ajustant des paramètres clés tels que la régularisation, la profondeur des arbres, ou le nombre de voisins.

En optimisant ces hyperparamètres, nous avons pu garantir que chaque classifieur fonctionne à son potentiel maximal, ce qui contribue à la robustesse et à la précision du modèle final.

5 Résultats et analyses

Dans cette section, nous présentons les résultats obtenus après l'entraînement de nos modèles de classification avec les meilleurs hyperparamètres identifiés grâce à la recherche exhaustive d'hyperparamètres (GridSearchCV). Nous avons effectué deux expérimentations distinctes : une sur des données déséquilibrées et une autre sur des données équilibrées.

Cependant, après avoir comparé les résultats des deux expérimentations, nous avons observé que les résultats sur les données équilibrées ne différaient pas de manière significative, ce qui a conduit à la conclusion que travailler avec les données déséquilibrées est une approche appropriée pour notre tâche.

5.1 Définition des Métriques

Les performances des modèles ont été évaluées en utilisant plusieurs métriques courantes dans les tâches de classification :

— **Exactitude (Accuracy)** : Proportion des prédictions correctes parmi toutes les prédictions effectuées. Elle est calculée comme suit :

$$\text{Exactitude} = \frac{\text{True Positives} + \text{True Negatives}}{\text{Total des Prédictions}}$$

— **Précision (Precision)** : Proportion des prédictions positives correctes parmi toutes les prédictions positives effectuées. Elle est calculée comme suit :

$$\text{Précision} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

— **Rappel (Recall)** : Proportion des éléments positifs correctement identifiés parmi tous les éléments positifs présents. Elle est calculée comme suit :

$$\text{Rappel} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

— **F1-Score** : Moyenne harmonique de la précision et du rappel, utile pour évaluer les modèles lorsque les classes sont déséquilibrées. Il est calculé comme suit :

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

5.2 Entraînement sans optimisation des hyperparamètres exhaustive

Avant d'utiliser GridSearchCV, nous avons d'abord entraîné nos modèles directement sur les données. Les résultats obtenus sont les suivants :

Modèle	Précision
Random Forest	0.9385
Decision Tree	0.6106
KNN	0.8582
Logistic Regression	0.9225
Polynomial Regression	0.9238
Neural Network	0.9413
AdaBoost	0.7114
Bagging	0.6895

Tab. 5.1 – Précision des modèles avant l'utilisation de la validation croisée.

Ensuite, nous avons appliqué la validation croisée avec 3 et 5 plis (`cv=3` et `cv=5`) pour évaluer la performance des modèles de manière plus robuste. Les résultats de la validation croisée avec `cv=3` et `cv=5` sont présentés dans les tableaux suivants.

5.2.1 Validation Croisée avec 3 Plis (`cv=3`)

Modèle	Précision (Validation Croisée)
Random Forest	0.9601
Decision Tree	0.9372
KNN	0.8430
Logistic Regression	0.9181
Polynomial Regression	0.9236
Neural Network	0.9523
AdaBoost	0.6975
Bagging	0.9592

Tab. 5.2 – Précision des modèles avec validation croisée à 3 plis (`cv=3`).

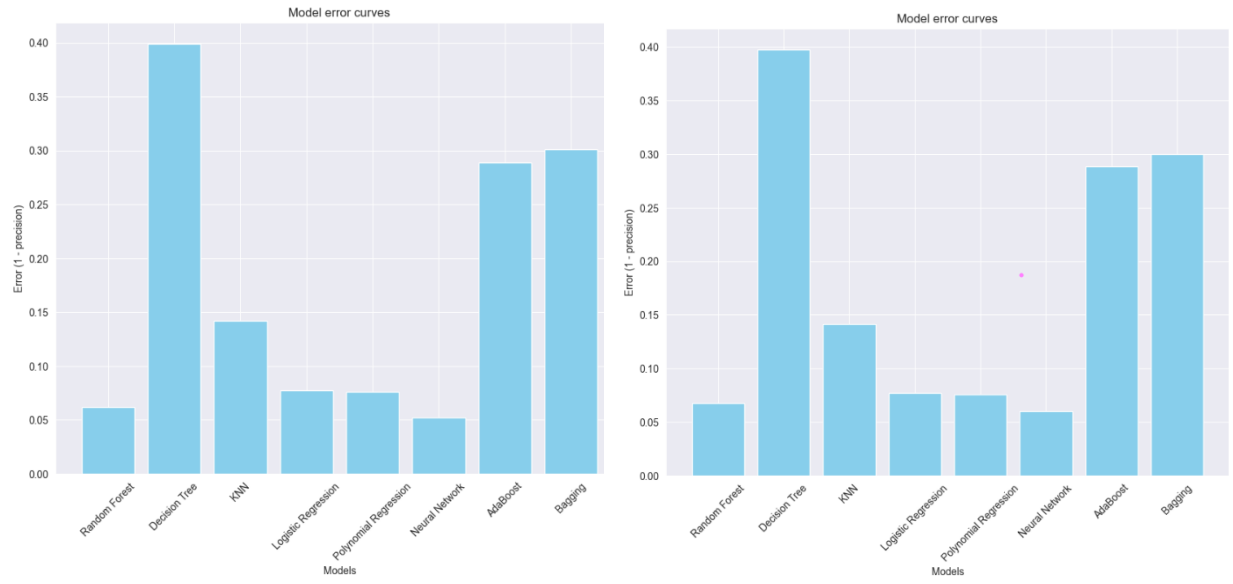
5.2.2 Validation Croisée avec 5 Plis (`cv=5`)

Modèle	Précision (Validation Croisée)
Random Forest	0.9610
Decision Tree	0.9386
KNN	0.8505
Logistic Regression	0.9208
Polynomial Regression	0.9248
Neural Network	0.9524
AdaBoost	0.7118
Bagging	0.9597

TAB. 5.3 – Précision des modèles avec validation croisée à 5 plis (`cv=5`).

Comme on peut le constater, l'utilisation de la validation croisée a permis d'améliorer la précision de plusieurs modèles. Par exemple, le *Random Forest* a montré une amélioration de sa précision, passant de 0.9385 à 0.9601 avec `cv=3` et à 0.9610 avec `cv=5`. De même, le modèle *Decision Tree* a vu sa précision augmenter de 0.6106 à 0.9372 avec `cv=3` et à 0.9386 avec `cv=5`. En revanche, d'autres modèles comme *AdaBoost* et *Bagging* ont montré une légère amélioration de leur précision, mais avec une différence moins marquée.

Les courbes d'erreur obtenues lors de la validation croisée peuvent également être utilisées pour observer les variations de la performance du modèle en fonction du nombre de plis. Elles peuvent être visualisées comme suit :



(a) Courbes d'erreur avec validation croisée à 3 folds

(b) Courbes d'erreur avec validation croisée à 5 folds

FIG. 5.1 – Comparaison des courbes d'erreur pour les modèles avec validation croisée.

En conclusion, la validation croisée a fourni des résultats plus fiables, en particulier pour des modèles comme *Random Forest* et *Decision Tree*, qui ont montré des perfor-

mances améliorées par rapport à l'entraînement direct sans validation. C'est pourquoi, afin d'optimiser davantage nos modèles, nous allons utiliser la méthode de recherche en grille (Grid Search) pour affiner les hyperparamètres et améliorer les performances des modèles de manière plus systématique.

5.3 GridSearchCV : Meilleurs Hyperparamètres

Les meilleurs hyperparamètres trouvés pour chaque modèle sont présentés dans le tableau ci-dessous :

Modèle	Meilleurs Paramètres
Logistic Regression	C: 10, solver: lbfgs
Decision Tree	criterion: entropy, max_depth: 10, min_samples_split: 10
Random Forest	max_depth: None, min_samples_split: 5, n_estimators: 200
SVC	C: 10, gamma: scale, kernel: poly
AdaBoost	learning_rate: 0.1, N_estimators: 50
KNeighbors Classifier	n_neighbors: 5, p: 1, weights: distance

Tab. 5.4 – Meilleurs paramètres de GridSearchCV pour chaque modèle

5.4 Évaluation des Modèles

Afin d'améliorer les performances de nos modèles, nous avons utilisé la méthode GridSearchCV pour effectuer une recherche exhaustive des meilleurs hyperparamètres pour chaque classifieur. Cette approche nous permet de trouver la combinaison optimale d'hyperparamètres en fonction des performances sur les données d'entraînement, tout en évitant le surapprentissage.

5.4.1 Logistic Regression

Les meilleurs paramètres trouvés pour le modèle de régression logistique sont :

— **C** : 10

— **Solver** : lbfgs

Les résultats d'évaluation pour ce modèle sont les suivants :

— **Exactitude** : 93.71%

— **Précision** : 93.89%

— **Recall** : 93.71%

— **F1-score** : 93.57%

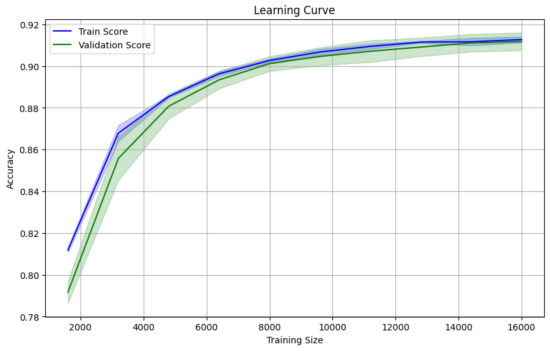


FIG. 5.2 – Courbe d'apprentissage pour Logistic Regression

5.4.2 Decision Tree

Les meilleurs paramètres trouvés pour le modèle de l'arbre de décision sont :

- **Critère** : entropy
- **Max depth** : 10
- **Min samples split** : 10

Les résultats d'évaluation pour ce modèle sont les suivants :

- **Exactitude** : 71.58%
- **Précision** : 67.31%
- **Recall** : 71.58%
- **F1-score** : 63.15%

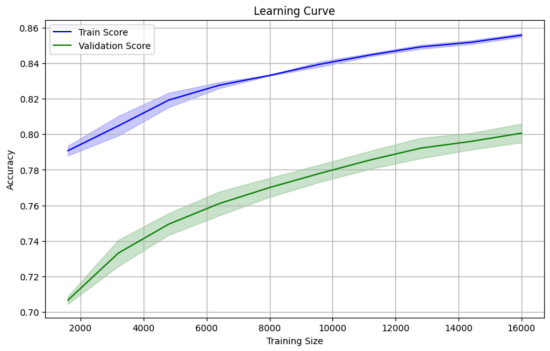


FIG. 5.3 – Courbe d'apprentissage pour Decision Tree

5.4.3 Random Forest

Les meilleurs paramètres trouvés pour le modèle de forêt aléatoire sont :

- **Max depth** : None
- **Min samples split** : 5
- **N estimators** : 200

Les résultats d'évaluation pour ce modèle sont les suivants :

- **Exactitude** : 93.15%

— **Précision** : 93.23%
— **Recall** : 93.15%
— **F1-score** : 93.08%

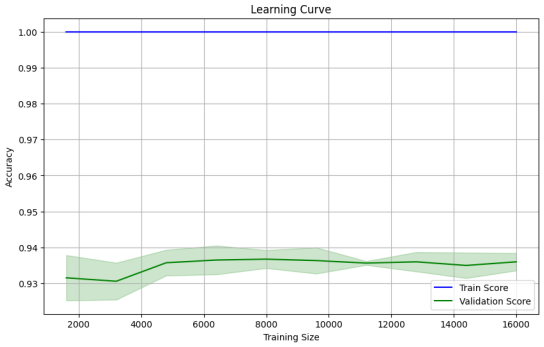


Fig. 5.4 – Courbe d'apprentissage pour Random Forest

5.4.4 SVC (Support Vector Classifier)

Les meilleurs paramètres trouvés pour le modèle SVC sont :

— **C** : 10
— **Gamma** : scale
— **Kernel** : poly

Les résultats d'évaluation pour ce modèle sont les suivants :

— **Exactitude** : 93.75%
— **Précision** : 93.95%
— **Recall** : 93.75%
— **F1-score** : 93.54%

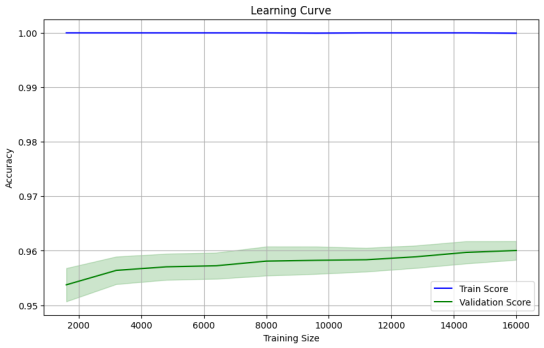


Fig. 5.5 – Courbe d'apprentissage pour SVC

5.4.5 AdaBoost

Les meilleurs paramètres trouvés pour le modèle AdaBoost sont :

— **Learning rate** : 0.01
— **N estimators** : 50

Les résultats d'évaluation pour ce modèle sont les suivants :

— **Exactitude** : 80.45%
— **Précision** : 65.99%
— **Recall** : 80.45%
— **F1-score** : 72.29%

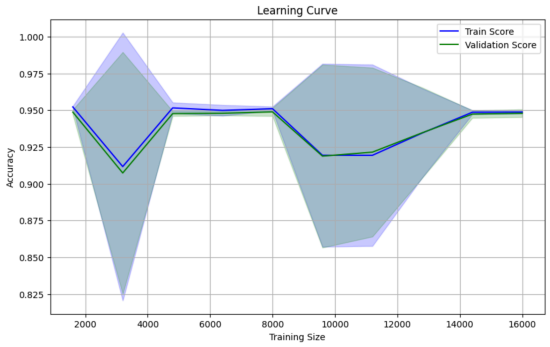


FIG. 5.6 – Courbe d'apprentissage pour AdaBoost

5.4.6 K-Nearest Neighbors (KNN)

Les meilleurs paramètres trouvés pour le modèle KNN sont :

— **N neighbors** : 5
— **P** : 1
— **Weights** : distance

Les résultats d'évaluation pour ce modèle sont les suivants :

— **Exactitude** : 87.85%
— **Précision** : 87.92%
— **Recall** : 87.85%
— **F1-score** : 87.69%

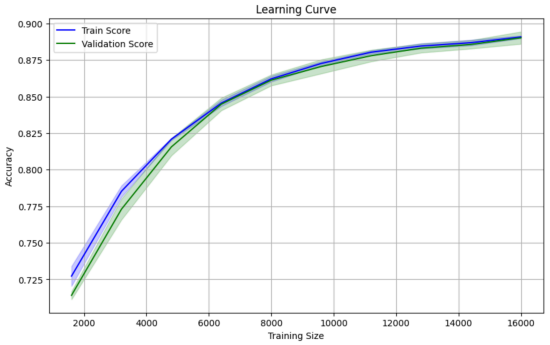


FIG. 5.7 – Courbe d'apprentissage pour KNN

5.5 Analyse des Résultats

En conclusion, l'application de **GridSearchCV** pour l'optimisation des hyperparamètres a permis d'améliorer significativement les performances des modèles par rapport à

un entraînement direct sans validation croisée. Les résultats obtenus pour chaque modèle sont les suivants :

- **Régression Logistique** : Une exactitude de 93.71%, avec des scores de précision, recall et F1 presque égaux, indiquant une bonne performance et un équilibre entre les deux classes.
- **Arbre de Décision** : Bien que l'exactitude soit de 71.58%, les scores de précision (67.31%) et de F1 (63.15%) sont relativement faibles, ce qui pourrait suggérer un problème de sur-apprentissage. La validation croisée a permis d'optimiser certains hyperparamètres, mais une amélioration est possible.
- **Forêt Aléatoire** : Le modèle a donné de très bons résultats avec une exactitude de 93.15% et des scores proches de 93% pour la précision, le recall et le F1, ce qui indique une très bonne capacité à généraliser sur les données.
- **SVC (Support Vector Classifier)** : Avec une exactitude de 93.75%, un score de précision de 93.95% et un recall de 93.75%, le SVC s'est avéré être très performant, notamment grâce à l'optimisation des hyperparamètres comme $C = 10$ et un noyau polynomial.
- **AdaBoost** : Ce modèle a montré des résultats plus faibles avec une exactitude de 80.45% et une précision de 65.99%, bien que le recall soit relativement élevé (80.45%). Cela suggère qu'il pourrait être amélioré avec un meilleur réglage des hyperparamètres.
- **KNN (K-Nearest Neighbors)** : Avec une exactitude de 87.85%, des scores de précision, recall et F1 très proches de 88%, ce modèle a montré une bonne capacité à généraliser, grâce à l'optimisation des paramètres comme $n\_neighbors = 5$ et $p = 1$.

Les courbes d'apprentissage pour chaque modèle ont montré une tendance à la convergence, ce qui est un bon indicateur que les modèles se sont bien adaptés aux données d'entraînement. Toutefois, quelques modèles comme l'arbre de décision et AdaBoost peuvent encore bénéficier d'un meilleur réglage des hyperparamètres pour optimiser leur performance.

En résumé, la validation croisée s'est avérée être un outil crucial pour évaluer la performance des modèles de manière plus fiable, surtout lorsqu'il y a des risques de sur-apprentissage. Ces résultats suggèrent qu'il serait judicieux d'effectuer des recherches supplémentaires pour affiner davantage ces modèles et maximiser leur performance globale.

Conclusion

Dans ce projet, nous avons utilisé le dataset *Stellar Classification Dataset (SDSS17)* pour classifier les étoiles en différentes catégories, en fonction de leurs caractéristiques spectrales. Après avoir nettoyé et préparé les données, nous avons appliqué plusieurs modèles de classification, notamment la régression logistique, l'arbre de décision, la forêt aléatoire, le SVC, AdaBoost, et KNN. Nous avons ensuite effectué une recherche exhaustive des hyperparamètres pour chaque modèle en utilisant `GridSearchCV` afin d'optimiser leurs performances.

Les résultats obtenus ont montré que les modèles tels que la régression logistique, la forêt aléatoire, et le SVC offraient de bonnes performances, avec des précisions et des scores F1 élevés, tandis que des modèles comme AdaBoost et l'arbre de décision ont montré des performances moins satisfaisantes. L'utilisation de la validation croisée a permis de renforcer la fiabilité des résultats, en particulier pour les modèles comme la forêt aléatoire et le SVC.

Les courbes d'apprentissage pour chaque modèle ont montré une bonne convergence des performances à mesure que l'on augmentait les données d'entraînement, suggérant que nos modèles sont bien adaptés à la tâche de classification des étoiles. En particulier, les courbes d'AUC-ROC ont confirmé la capacité des modèles à bien séparer les différentes classes.

Dans les prochaines étapes, il serait intéressant d'explorer l'utilisation de techniques d'optimisation supplémentaires comme l'optimisation bayésienne ou d'autres techniques avancées pour encore améliorer les performances des modèles, notamment pour les modèles qui ont montré des résultats moins bons. De plus, il serait pertinent de mener une analyse plus approfondie des variables et d'explorer d'autres techniques de prétraitement des données pour améliorer encore la classification.

En conclusion, ce projet a démontré l'importance de l'optimisation des hyperparamètres et de la validation croisée pour obtenir des modèles de classification robustes et performants.

Bibliographie

[1] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., et al. (2011). Scikit-learn : Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. https://scikit-learn.org/

[2] Chen, T., Guestrin, C. (2016). XGBoost : A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (pp. 785-794). ACM. https://xgboost.readthedocs.io/

[3] Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)* (pp. 1137-1143).

[4] Fawcett, T. (2006). An introduction to ROC analysis. *Pattern Recognition Letters*, 27(8), 861-874. https://doi.org/10.1016/j.patrec.2005.10.010

[5] Soriano, F. (2020). Stellar Classification Dataset (SDSS17). *Kaggle*. https://www.kaggle.com/datasets/fedesoriano/stellar-classification-dataset-sdss17

[6] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1), 5-32. https://doi.org/10.1023/A:1010933404324

[7] CART, L. (1986). Classification and Regression Trees. *Wadsworth and Brooks/Cole*.

[8] Hosmer, D. W., Lemeshow, S. (2000). *Applied Logistic Regression* (2nd ed.). Wiley-Interscience.

[9] Cover, T., Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27. https://doi.org/10.1109/TIT.1967.1053964