Rapport Projet 2IA

Régression et Classification

Réalisé par

Ebnou Abdoum* Anatole Carre* Lucas Gault*

Briag Guégan-Royan* Jacqueline Rabotchi† Alice Touzet‡

*prenom.nom@etudiant.univ-rennes.fr

†jacqueline.rabotchi.1@etudiant.univ-rennes.fr

‡alice.rignault-touzet@etudiant.univ-rennes.fr

Avril 2026

Résumé

Ce rapport présente les résultats d'un projet encadré mené sur deux jeux de données réels dans le cadre du module Introduction à l'IA. La première partie traite un problème de régression : prédire l'année de sortie d'une chanson à partir de caractéristiques de timbre audio (dataset *Year Prediction MSD*). La seconde partie aborde un problème de classification sur le dataset *NASA Exoplanets*. Pour chaque tâche, nous décrivons les choix d'organisation, l'exploration des données, les méthodes de modélisation retenues et leur évaluation comparative.

Table des matières

1. Organisation de l'équipe

1.1. Répartition en sous-groupes

Après avoir lu l'énoncé dans son ensemble, on a constaté que le projet se découpait en deux parties bien distinctes (une tâche de régression et une tâche de classification), ce qui rendait naturelle une division en deux équipes de trois travaillant en parallèle. La répartition a été décidée dès la première séance :

— **Groupe Régression** : Ebnou A., Briag G.-R., Alice T.
— **Groupe Classification** : Anatole C., Jacqueline R., Lucas G.

1.2. Cahier des charges et jalons

Un cahier des charges a été établi collectivement pour structurer les 12 heures encadrées, servant de référence commune tout au long du projet. Les grandes étapes retenues suivaient la progression méthodologique attendue :

Séance 1
: Exploration initiale : choix du dataSet, prise en main des jeux de données, premières visualisations et vérification de la qualité des données.

Séance 2
: Analyse des corrélations, sélection de variables candidates et premiers essais de modélisation.

Séance 3
: Entraînement, régularisation et évaluation comparative des modèles.

Séance 4
: Mise en commun des deux parties, rédaction finale et préparation de l'archive de rendu.

1.3. Infrastructure et outils collaboratifs

Dès la première séance, un dépôt Git commun a été initialisé avec deux branches parallèles, `regression` et `classification`, afin que chaque equipe puisse avancer de manière autonome tout en gardant une visibilité sur l'avancement de l'autre. Chaque branche est structurée en sous-dossiers (`data/`, `docs/`, `src/`) pour séparer les figures, les rapports intermédiaires et les scripts. Le dépôt est accessible à l'adresse suivante :

`https://gitlab2.istic.univ-rennes1.fr/eebnouabdoum/2ia-projet`

Note : le dépôt étant privé, merci de contacter l'un des membres du groupe pour obtenir un accès — Ebnou, Anatole, Lucas, Briag, Jacqueline ou Alice.

Les rapports intermédiaires versionnés au fil des séances assurent une traçabilité du travail effectué et évitent toute perte de scripts entre deux séances, et surtout facilitent la redaction du rendu final.

2. Partie Régression — Year Prediction MSD

Note : On attire l'attention sur les variations entre le notebook et le rapport. En effet, l'ordre d'exécution des cellules peut entraîner des résultats différents. Ainsi, les résultats de ce rapport se basent sur une exécution passée.

2.1. Présentation du problème

2.1.1. Contexte et motivation

Le dataset *Year Prediction MSD* est extrait du projet Million Song Dataset (MSD), une base de données à grande échelle sur la musique populaire. L'objectif est de prédire l'année de sortie d'une chanson à partir de descripteurs de timbre extraits de l'audio.

Ce problème présente un intérêt concret pour les systèmes de recommandation musicale : dater automatiquement une chanson permet d'organiser des catalogues, de repérer des tendances sonores à travers le temps, ou encore d'alimenter des moteurs de recommandation contextuels, Notre encadrant (M. Hellier) a d'ailleurs évoqué cette dimension lors d'un point d'avancement (Seance 2), en soulignant que les mêmes descripteurs de timbre pourraient servir à d'autres tâches (retrouver des chansons au son similaire, par exemple).

La difficulté principale vient du caractère abstrait des variables explicatives : les descripteurs de timbre sont des représentations spectrales compactes, sans correspondance directe avec des attributs musicaux interprétables. Les corrélations avec l'année de sortie restent faibles sur l'ensemble des 90 variables, ce qui rend la tâche non triviale et, par là même, pertinente.

2.1.2. Description du dataset

Propriété	**Valeur**
Source	Kaggle / UCI Repository
Dimensions	515 345 observations × 91 colonnes
Variable cible	`Year` (année de sortie, 1922–2011)
Variables explicatives	90 : `TimbreAvg1-12` et `TimbreCovariance1-78`
Valeurs manquantes	Aucune (nettoyées sur Kaggle)

Les 12 variables `TimbreAvg` correspondent aux composantes moyennes du vecteur de timbre obtenu par analyse spectrale. Les 78 variables `TimbreCovariance` correspondent

aux termes de la matrice de covariance de ce même vecteur (ce dataset a été retenu délibérément pour sa difficulté).

En savoir plus sur le dataset :

- **Kaggle** (données + description) : `www.kaggle.com/datasets/zakariaeyoussefi/song-year-prediction-msd`
- **UCI Machine Learning Repository** : `https://archive.ics.uci.edu/dataset/203/yearpredictionmsd`
- **Site officiel du Million Song Dataset** : `http://millionsongdataset.com`

2.2. Analyse exploratoire

2.2.1. Statistiques descriptives sur Year

Statistique	Valeur
Minimum	1922
Maximum	2011
Moyenne	1998.4
Médiane	2002.0
Écart-type	10.93

2.2.2. Distribution de la variable cible

La distribution de `Year` est fortement asymétrique (Figure 1) : on observe une concentration très marquée sur la période 1995–2008, une quasi-absence de chansons antérieures à 1955, et une chute après 2009 (qu'on pense liée à un biais de collecte du MSD). Cette non-normalité a eu une conséquence directe sur le choix des métriques d'évaluation : le test de Shapiro-Wilk appliqué aux résidus de tous nos modèles rejette la normalité ($p < 2.2 \times 10^{-16}$), ce qui nous a empêchés d'utiliser une ANOVA pour comparer les modèles entre eux.

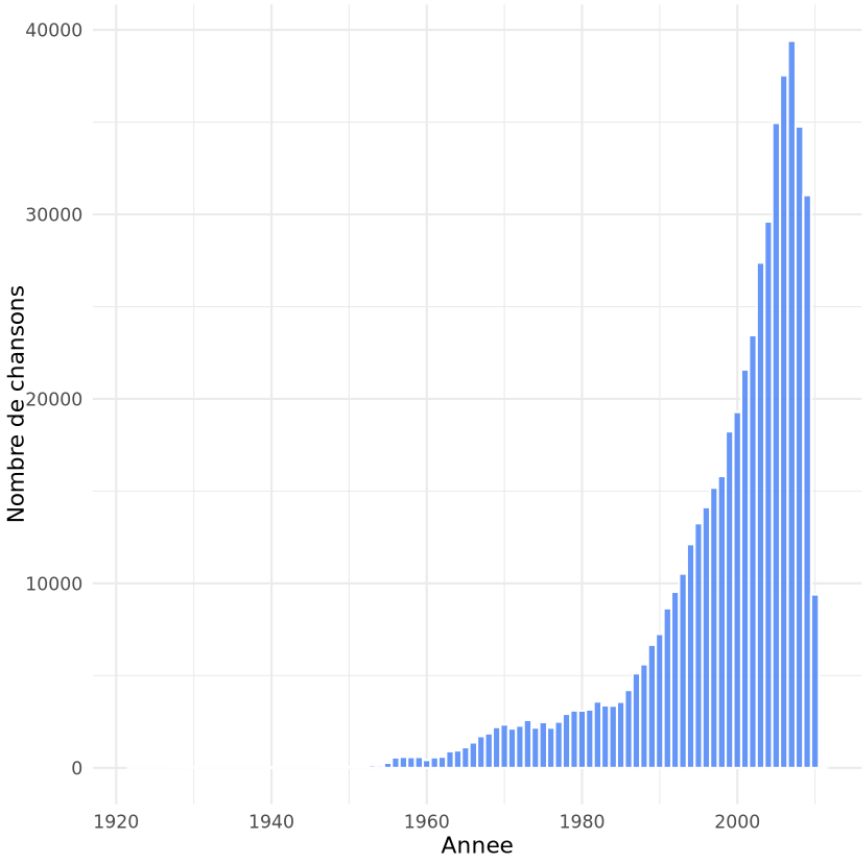

Figure 1 – Distribution du nombre de chansons par année — dataset complet (515 345 observations).

2.2.3. Choix de la taille de l'échantillon

Travailler sur la totalité des 515 345 observations s'est avéré rédhibitoire pour la sélection de variables par stepAIC. On a donc comparé plusieurs tailles d'échantillons en évaluant la fidélité de leur distribution à celle du jeu complet (Fig 2). L'échantillon de $n = 5\,000$ observations reproduit fidèlement la distribution tout en maintenant des temps de calcul raisonnables ; les tailles inférieures ($n = 500..$) déforment sensiblement la distribution et ont été écartées.

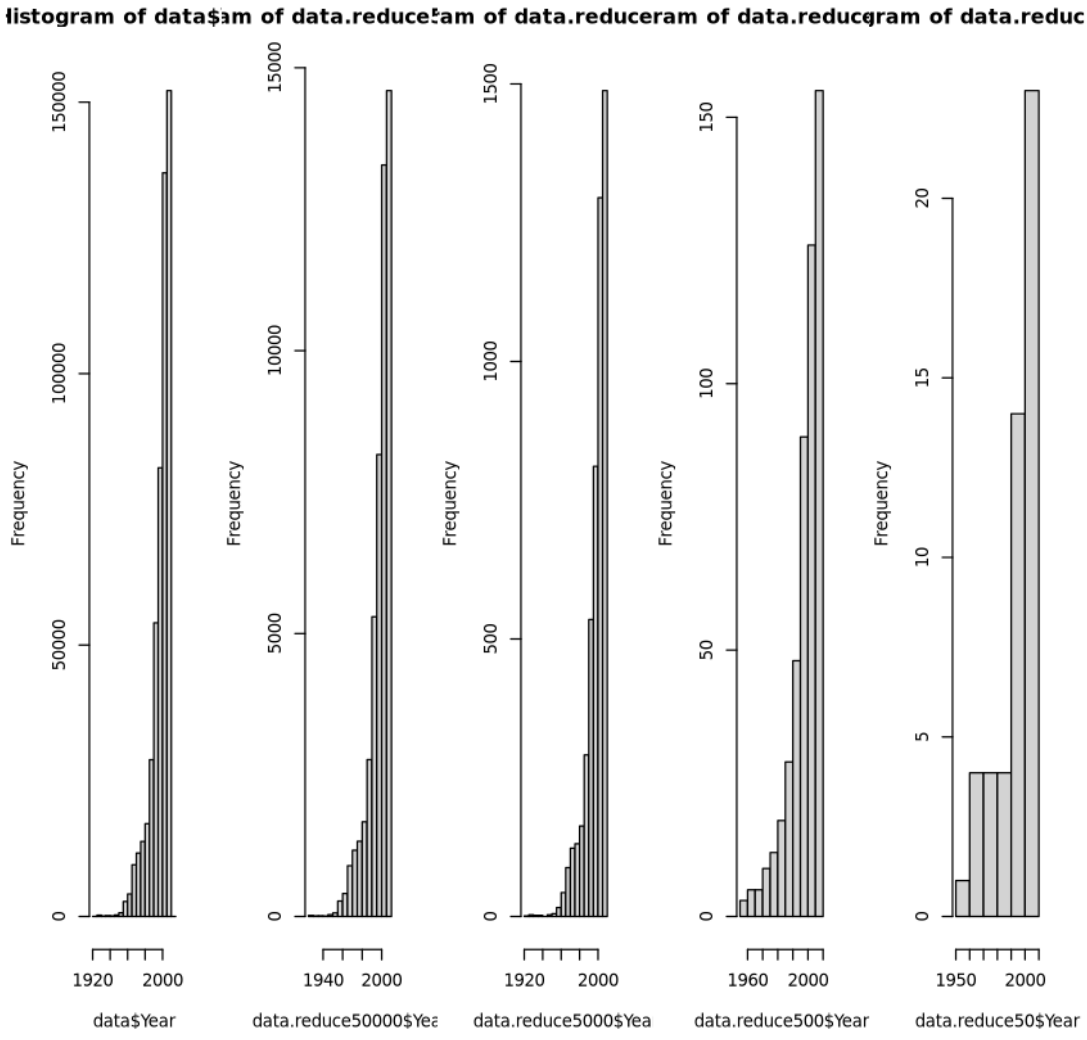

Figure 2 – Comparaison des distributions de `Year` pour les tailles d'échantillon 50 000, 5 000, 500 et 50.

2.2.4. Analyse des corrélations

L'analyse des corrélations étant explicitement mentionnée dans l'énoncé du projet comme outil attendu, nous avons cherché comment la mettre en œuvre sur nos 90 variables explicatives. Nous avons découvert et utilisé la librairie `corrplot` (https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html), qui permet de produire des représentations visuelles de matrices de corrélation.

Pour les 12 variables `TimbreAvg`, `corrplot` nous a permis de produire une heatmap annotée avec les coefficients de Pearson, parfaitement lisible sur ce nombre de variables (Figure 3).

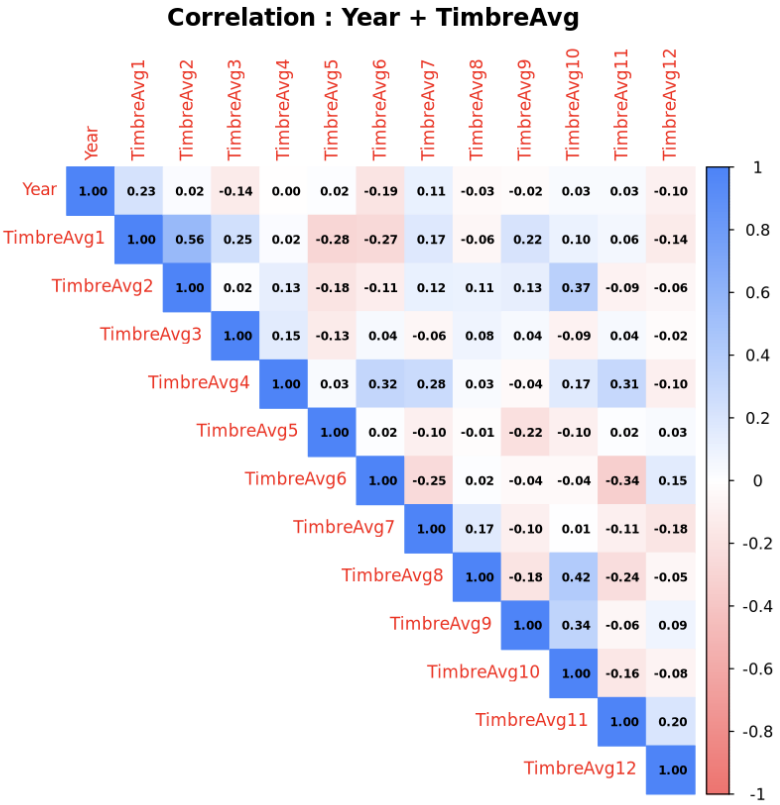

Figure 3 – Matrice de corrélation entre `Year` et les 12 variables `TimbreAvg`.

Pour les 78 variables `TimbreCovariance`, une heatmap 79×79 était illisible, même avec `corrplot`. Nous avons donc opté pour un graphique en barres horizontales classant chaque variable par sa corrélation de Pearson avec `Year` en valeur absolue décroissante. Ce choix permet d'identifier d'un coup d'œil les variables les plus liées à l'année de sortie sans surcharge visuelle. Le calcul a été réalisé comme suit :

```
cov_cols <- paste0("TimbreCovariance", 1:78)
corr_cov <- cor(data[, c("Year", cov_cols)], use = "complete.obs")
```

Parmi les `TimbreAvg`, `TimbreAvg1` présente la corrélation positive la plus forte ($r = 0{,}23$), suivie en négatif par `TimbreAvg6` ($r = -0{,}19$) et `TimbreAvg3` ($r = -0{,}14$). Parmi les `TimbreCovariance`, la plus corrélée positivement est `TimbreCovariance35` ($r \approx 0{,}09$–$0{,}10$), et la plus corrélée négativement est `TimbreCovariance51` ($r \approx -0{,}12$). Globalement, $|r| < 0{,}25$ pour l'ensemble des variables.

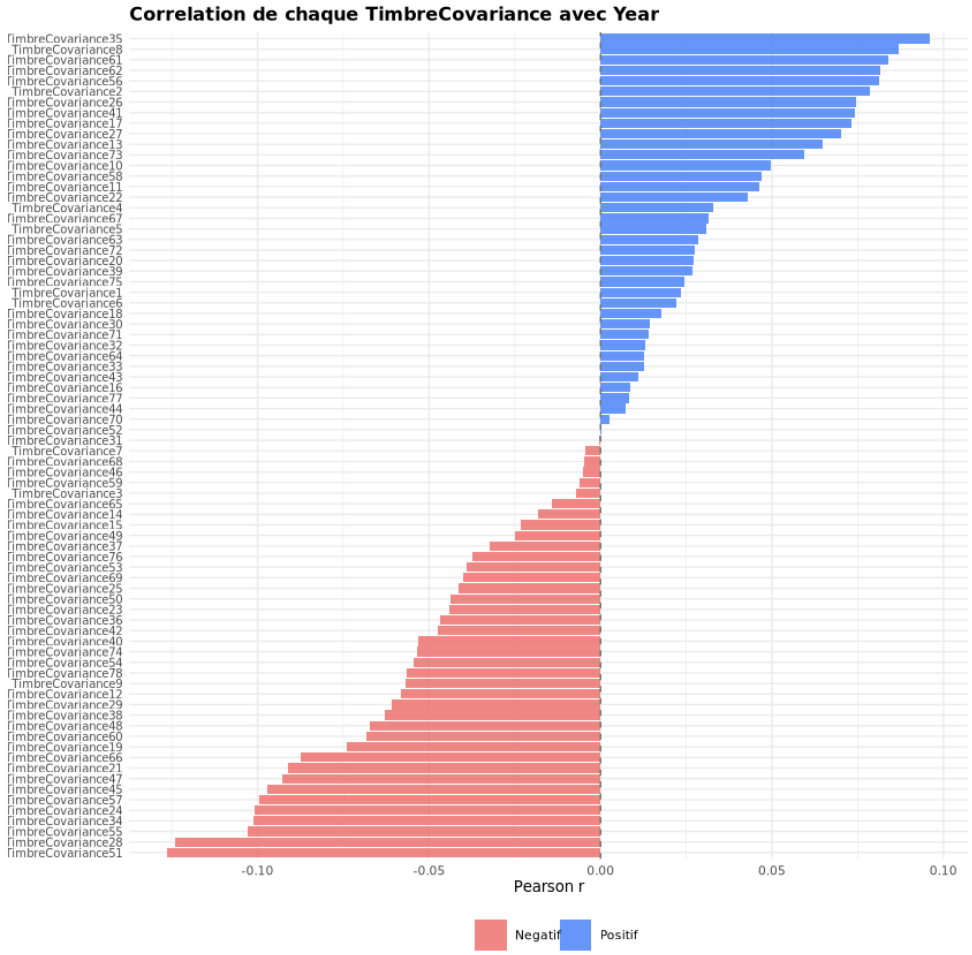

Figure 4 – Corrélations de Pearson entre `Year` et les 78 variables `TimbreCovariance`

2.3. Modélisation

2.3.1. Découpage train/test

Deux découpages ont été utilisés selon la méthode testée.

Pour les modèles nécessitant une sélection de variables (`stepAIC`), un échantillon aléatoire de 5 000 observations a été tiré avec `set.seed(42)`, puis découpé en 80 % entraînement (4 000 obs.) et 20 % test (1 000 obs.).

Pour les modèles supportant de grands volumes (LASSO, Random Forest), nous avons également testé le découpage officiel recommandé par les auteurs du dataset (https://archive.ics.uci.edu/dataset/203/yearpredictionmsd) : les 463 715 premières observations pour l'entraînement et les 51 630 dernières pour le test. Ce découpage est conçu pour éviter le biais dit « producer effect » : il garantit qu'aucun artiste n'apparaît à la fois dans le train et dans le test.

2.3.2. Modèle de référence simple

Avant d'appliquer des méthodes automatiques, on a construit un modèle simple en sélectionnant manuellement six variables parmi les plus corrélées à `Year` : `TimbreAvg1`, `TimbreAvg6`, ainsi que quatre termes de covariance (`TimbreCovariance35`, `51`, `28`, `55`). Ce choix reflète les variables les plus saillantes dans les matrices de corrélation. Ce modèle simple atteint un R^2 ajusté de 0.088, ce qui sert de ligne de base pour la comparaison.

2.3.3. Régression linéaire avec sélection stepAIC

La procédure stepAIC appliquée au modèle complet à 90 variables a retenu **44 variables**, réduisant l'AIC de 29 363,83 à 29 297,21 et le BIC de 29 942,88 à 29 586,73. Le R^2 ajusté passe de 0,213 (modèle complet) à 0,218 (modèle stepAIC).

2.3.4. Régularisation LASSO

On a préféré le LASSO à la régression Ridge pour sa capacité à annuler certains coefficients, réalisant ainsi une sélection de variables directement dans l'estimation. Le LASSO ajoute une pénalisation $\lambda \sum |\beta_j|$ à la fonction de coût : plus λ est grand, plus des coefficients sont forcés exactement à zéro.

Le paramètre optimal obtenu est $\lambda_{\min} \approx 0{,}037$. Le modèle LASSO retient 75 variables sur 90 ; la variable dominante est `TimbreAvg1` (coefficient $\approx 0{,}839$), nettement au-dessus de toutes les autres. La trajectoire des coefficients en fonction de $\log(\lambda)$ est présentée en Figure 5.

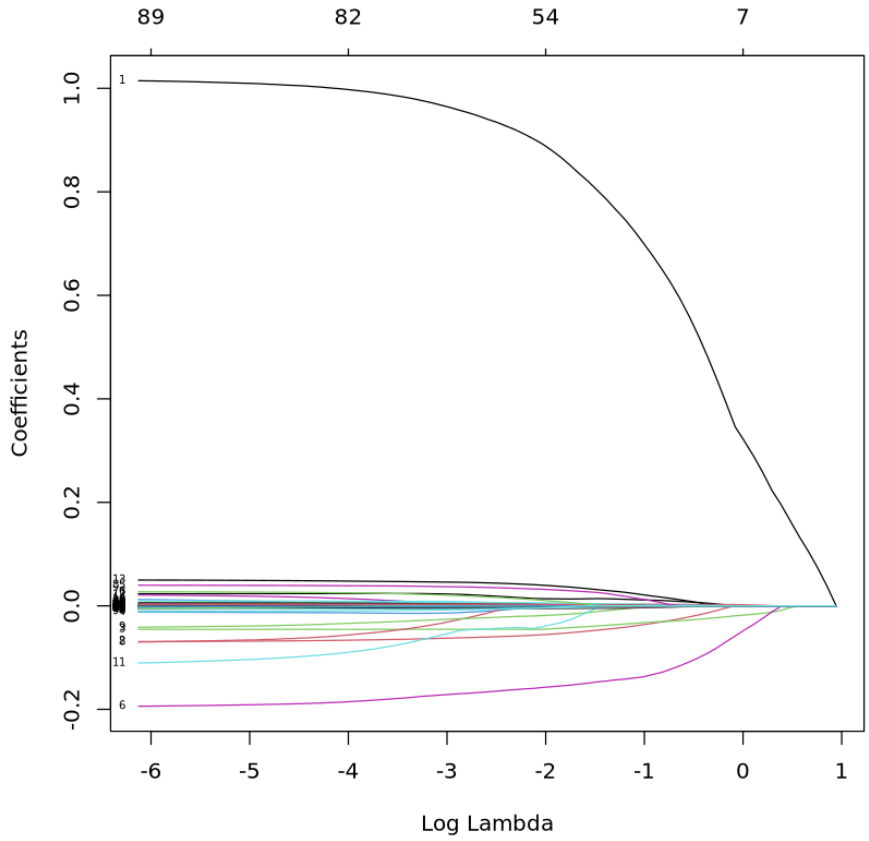

Figure 5 – Trajectoire des coefficients LASSO en fonction de $\log(\lambda)$

Nous avons appliqué le LASSO sur deux découpage afin d'évaluer la robustesse des résultats :

Découpage	**MSE**	**R² (ajuste)**
5 000 observations (aléatoire, `set.seed(42)`)	35,44	0,225
Officiel (463 715 train / 51 630 test)	90,44	0,232

Le R^2 ajusté est similaire entre les deux découpages ($\approx 0{,}23$), ce qui confirme que le plafond observé n'est pas un artefact du sous-échantillonnage. Le MSE plus élevé sur le découpage officiel s'explique par la plage d'années plus large dans le jeu de test, qui contient toutes les époques y compris les années antérieures à 1960, quasi absentes de l'entraînement.

2.3.5. Tentative d'intersection stepAIC ∩ LASSO

Nous avons calculé l'intersection des variables retenues par stepAIC et par LASSO, aboutissant à 44 variables communes. Ce modèle atteignait un R^2 ajusté de 0,202 sur l'entraînement, soit un résultat légèrement inférieur au modèle stepAIC seul (0,218). Cette dégradation s'explique par le fait que certaines variables retenues par stepAIC mais écartées

par LASSO apportaient néanmoins un signal utile. Ce modèle n'a donc pas été retenu pour la comparaison finale.

2.3.6. Random Forest

Face au plafonnement des modèles linéaires autour de $R^2 \approx 0{,}22$, et sachant que ce dataset est largement étudié dans la communauté, nous avons cherché ce qui avait déjà été tenté. Le dépôt de Dogra (2020), disponible sur GitHub (https://github.com/prakhardogra921/Year-Prediction-using-Regression), compare plusieurs méthodes de régression sur ce même jeu et confirme que les modèles linéaires (régression simple, LASSO, Ridge, Elastic Net) atteignent rapidement un plafond (voir tableau comparative, depôt git, Paper.pdf, page 4) , et que des gains significatifs nécessitent des méthodes capturant des relations non linéaires. C'est ce qui nous a conduits à explorer le Random Forest, une méthode ensembliste vue en cours. Cette approche capture des interactions non linéaires entre les 90 descripteurs sans transformation préalable.

Nous avons testé cette méthode sur les deux découpages :

— Sur l'échantillon de 5 000 observations : MSE = 36,18, $R^2 = 0{,}235$
— Sur le découpage officiel : MSE = 94,45, $R^2 = 0{,}198$

Le Random Forest apporte une amélioration marginale par rapport aux modèles linéaires (+0,01 de R^2 sur le découpage 5 000). Le graphique d'importance des variables (Figure 6) confirme la domination de `TimbreAvg1`, déjà identifiée par le LASSO, ce qui est cohérent entre les deux approches.

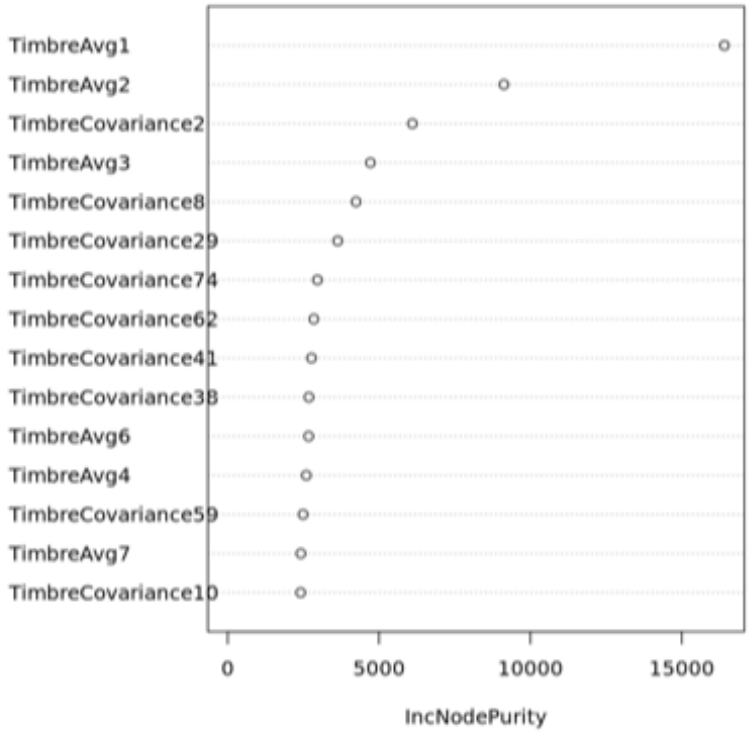

Figure 6 – Importance des 15 variables les plus influentes selon le Random Forest (critère : réduction de l'impureté des nœuds, *IncNodePurity*).

2.4. Explorations complémentaires

2.4.1. Réduction du nombre de variables

Pour tester si une réduction du nombre de variables améliorait la généralisation, on a tracé les 15 variables les plus corrélées à `Year` dans l'espace bidimensionnel (variable vs. `Year`). Les nuages de points obtenus ne présentaient aucune structure exploitable (forme de tornade verticale), confirmant l'absence de relation linéaire simple dans ce sous-espace. Le modèle entraîné sur ces 15 variables atteint un R^2 ajusté de 0,088, identique au modèle simple, ce qui n'apporte pas d'amélioration suffisante pour justifier cette reduction.

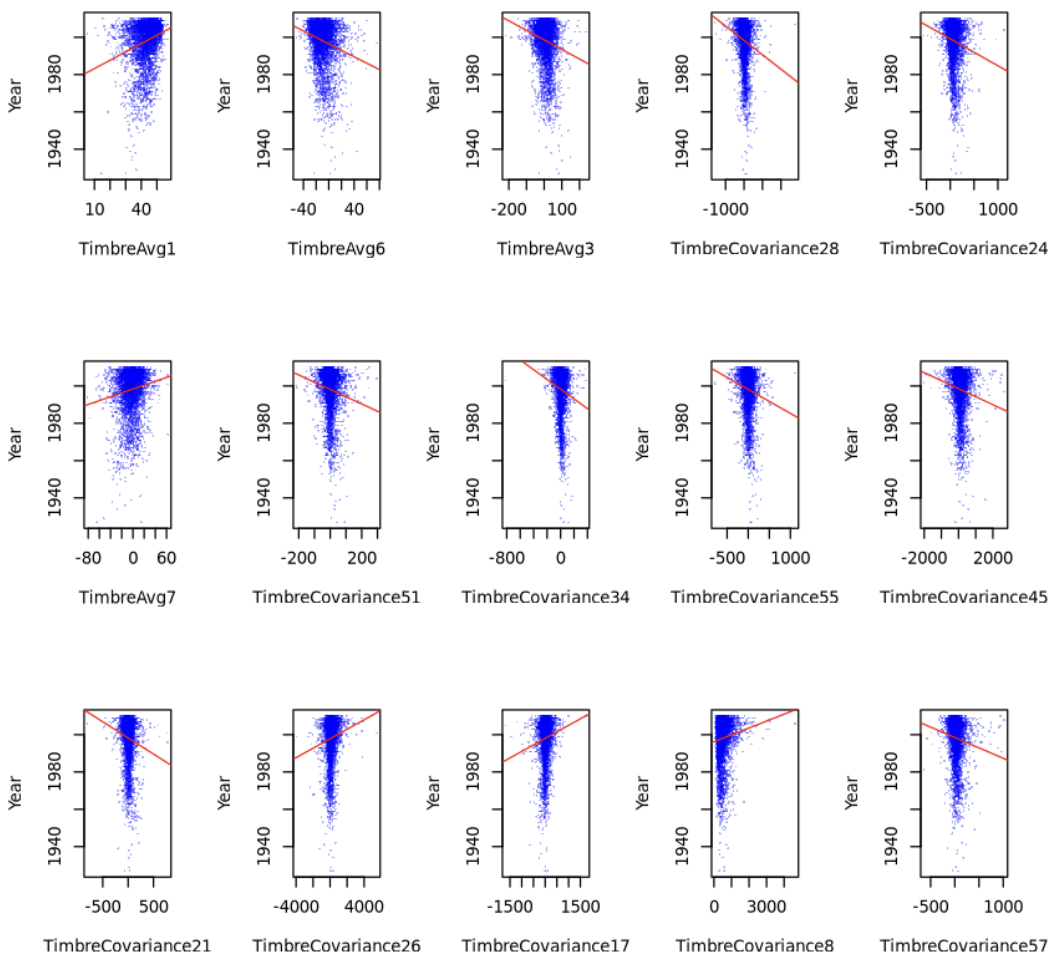

Figure 7 – Nuages de points des 15 variables les plus corréles à `Year`.

2.4.2. Réduction de la plage de prédiction

On a également exploré une réduction du problème lui-même en restreignant la plage de prédiction aux années 1980–2010, qui concentrent l'essentiel des observations. L'idée était double : limiter le risque que le modèle apprenne par cœur la surreprésentation de certaines années, et réduire la variance de la variable cible. Cependant, les gains obtenus sur les métriques sont restés quasi nuls par rapport aux modèles entraînés sur l'ensemble complet, ce qui nous a conduits à l'abandonner.

2.4.3. Tentative de réseau de neurones

Face au plafonnement des modèles linéaires, nous avons tenté d'appliquer un réseau de neurones via le package `nnet` (une couche cachée). Cependant, ce package impose une limite sur le nombre de poids : avec 90 variables en entrée, seuls des réseaux de 5 neurones maximum étaient réalisables sans erreur. Les configurations testées (size = 5,

decay = 0,001 à 0,1) donnaient un R^2 de 0,18 au mieux, inférieur aux modèles linéaires. Les configurations à size ≥ 8 divergeaient complètement (R^2 négatif). Nous avons donc abandonné cette piste, `nnet` n'étant pas adapté à ce nombre de variables sans réduction préalable de la dimension.

2.5. Synthèse

2.5.1. Évaluation des modèles

L'ensemble des modèles ont été évalués sur le jeu de test avec deux métriques : le MSE ($\frac{1}{n}\sum(y_i - \hat{y}_i)^2$) et le R^2. Remarque : comme precise avant, le test de Shapiro-Wilk ayant rejeté la normalité des résidus pour tous les modèles ($p < 2{,}2 \times 10^{-16}$), une comparaison par ANOVA n'était pas envisageable.

Modèle	**Découpage**	**MSE**	**R^2**
Simple (6 variables)	5 000 aléatoire	44,55	0,058
StepAIC (44 variables)	5 000 aléatoire	37,13	0,215
LASSO	5 000 aléatoire	35,44	0,225
Random Forest	5 000 aléatoire	36,18	0,235
LASSO	Officiel 463k/51k	90,44	0,232
Random Forest	Officiel 463k/51k	94,45	0,198

Les Figures 8 et 9 illustrent graphiquement les performances des quatre modèles principaux sur le jeu de test à 5 000 observations.

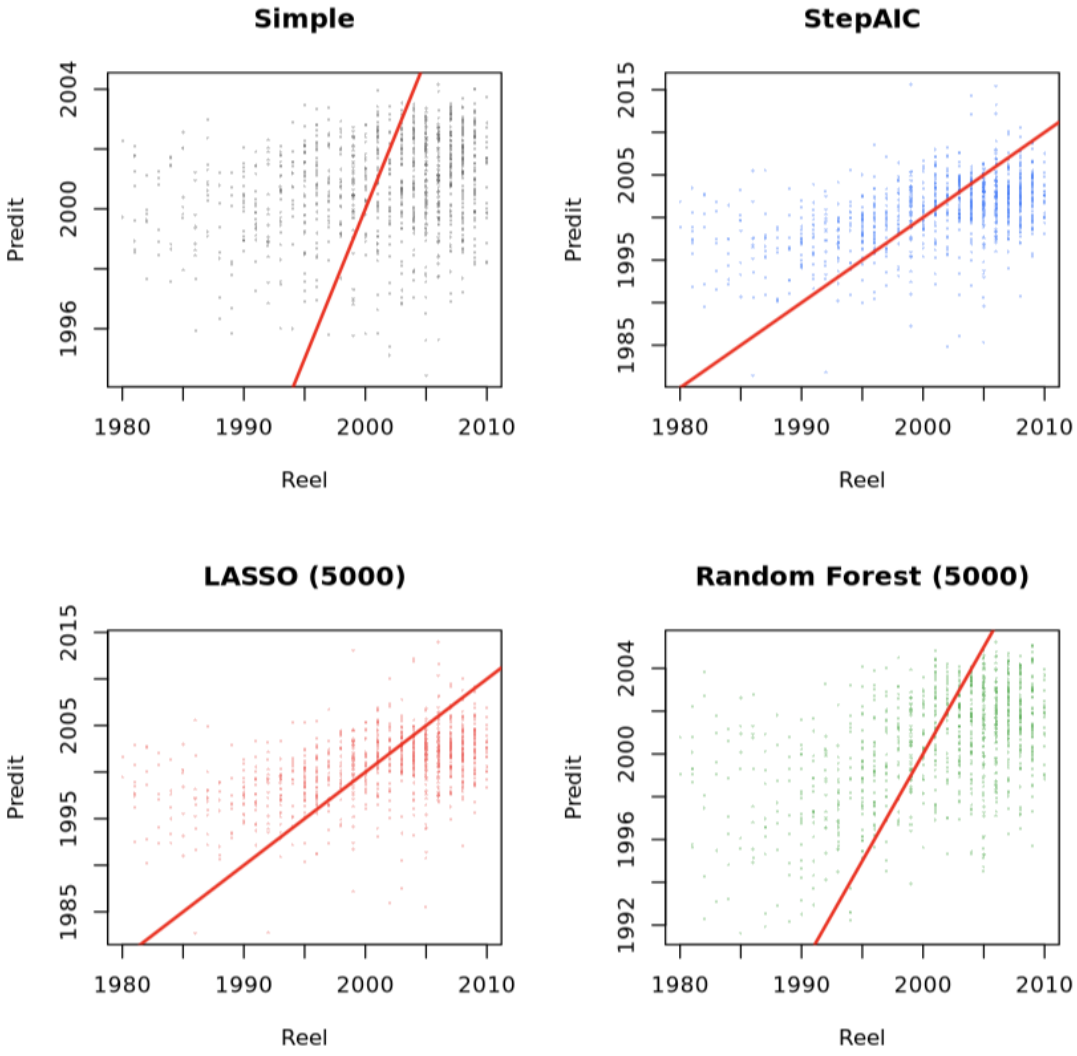

Figure 8 – Valeurs prédites vs valeurs réelles pour les quatre modèles ; jeu de test (1 000 observations). La diagonale rouge représente la prédiction parfaite.

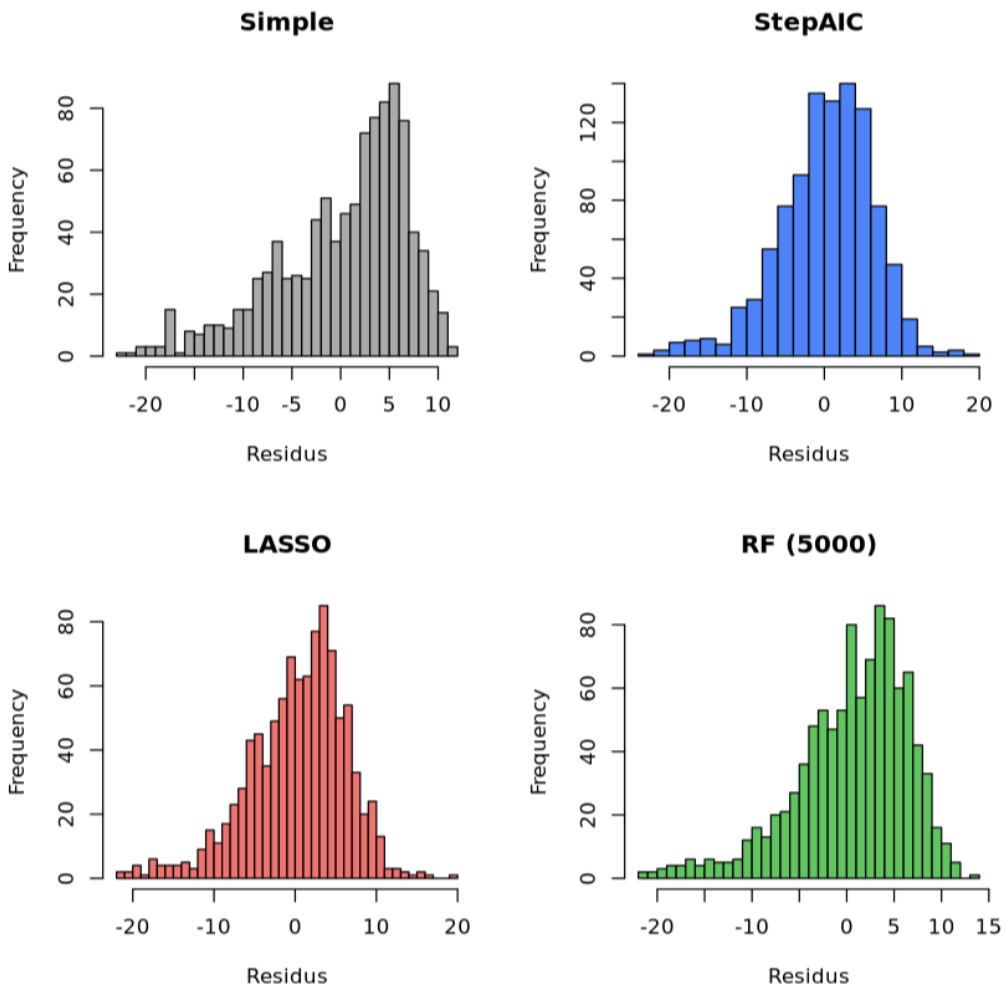

Figure 9 – Distribution des résidus pour les quatre modèles.

Les graphiques prédit vs réel révèlent un comportement commun à tous les modèles : les prédictions se concentrent autour de 1998–2005 quelle que soit l'année réelle, reflétant la surreprésentation de cette période dans les données d'entraînement. Les années antérieures à 1990 sont systématiquement mal prédites. Les distributions des résidus montrent une asymétrie vers les valeurs positives (le modèle prédit des années trop récentes pour les chansons anciennes) confirmée par le rejet de la normalité au test de Shapiro-Wilk.

2.5.2. Discussion et limites

Les résultats obtenus plafonnent autour de $R^2 \approx 0{,}23$, ce qui est cohérent avec les documents de recherche sur ce dataset. Ce plafond n'est pas un échec méthodologique mais une limite inhérente aux données : les descripteurs de timbre encodent la couleur sonore d'une chanson, pas son époque. L'évolution musicale à travers les décennies est réelle mais subtile, et masquée par une très forte hétérogénéité inter-artistes.

Le travail de Dogra (2020) sur ce même dataset, qui teste également la régression polynomiale et un réseau de neurones à une couche, aboutit aux mêmes conclusions : les

méthodes linéaires et quasi-linéaires plafonnent toutes autour de $R^2 \approx 0{,}20$–$0{,}25$. Cela confirme que la limite observée n'est pas liée à nos choix méthodologiques mais au signal disponible dans les variables.

Plusieurs limites spécifiques à notre approche méritent d'être soulignées :

— **Taille de l'échantillon.** La contrainte imposée par `stepAIC` nous a forcés à travailler sur 5 000 observations, soit moins de 1 % du dataset. Cela introduit une variance dans les résultats selon le tirage, atténuée par l'utilisation de `set.seed(42)`.

— **Découpage aléatoire.** Notre tirage aléatoire ne respecte pas le découpage officiel recommandé par les auteurs, qui garantit qu'aucun artiste n'apparaît dans les deux ensembles (« producer effect »). Nos métriques sur 5 000 obs. sont donc légèrement optimistes. Le test sur le découpage officiel ($R^2 = 0{,}232$ pour le LASSO) confirme néanmoins la cohérence des résultats.

— **Réseau de neurones.** Le package `nnet` ne supporte pas plus de quelques centaines de poids, ce qui limitait les réseaux à 5 neurones avec 90 variables en entrée. Les résultats ($R^2 \approx 0{,}18$) étaient inférieurs aux modèles linéaires. Des architectures plus profondes seraient nécessaires pour exploiter une éventuelle non-linéarité des données.

— **Méthodes non explorées.** Par contrainte de temps dans le cadre des 12 heures imparties, nous n'avons pas pu tester Ridge, la régression polynomiale. Ces pistes restent ouvertes pour une exploration future.

3. Partie Classification — NASA Exoplanets

3.1. Présentation du problème

3.1.1. Contexte

Le dataset *cleaned_520.csv*, qu'on nommera `exoplanet.csv`, est un jeu de données provenant de l'article *NASA Exoplanets* sur Kaggle. Ce jeu représente un extrait de la base de données contenant un certain nombre de caractéristiques sur les planètes découvertes à travers les différentes missions effectuées par la NASA. L'objectif de cette partie est de prédire la classification des exoplanètes par rapport au type de celles-ci (*Neptune-like*, *Gas Giant*, *Super Earth* et *Terrestrial*). Ce problème de classification peut aider et être réutilisé dans le secteur de la recherche spatiale ou encore dans la classification après la récolte massive de données brutes d'astrophysiques.

Pour aller plus loin, ce problème peut servir à prédire si une planète est habitable ou non, ce qui fait rêver pour certain(e)s et motive notre choix du jeu.

3.1.2. Description du dataset

Propriété	**Valeur**
Source	Kaggle — *NASA Exoplanets*
Dimensions initiales	5 250 observations × 13 colonnes
Variable cible	`planet_type` (4 classes)
Variables quantitatives	8 : `distance`, `stellar_magnitude`, `discovery_year`, `mass_multiplier`, `radius_multiplier`, `orbital_radius`, `orbital_period`, `eccentricity`
Variables qualitatives	5 : `name`, `planet_type`, `mass_wrt`, `radius_wrt`, `detection_method`

3.1.3. Première analyse et observation du dataset

Notre dataset n'était pas encore complètement propre. En effet, une partie de notre jeu de données possédait un label de classification nommé `Unknown`, car nombreux étaient les échantillons auxquels il manquait des données parmi leurs colonnes, comme montré en Figure 10. On pouvait voir les zones noires, montrant l'emplacement des données manquantes et donc nous permettant de trouver les échantillons incomplets. Il était essentiel pour nous de ne pas avoir cette incomplétude pour pouvoir utiliser et tester des algorithmes de classification supervisée / non supervisée.

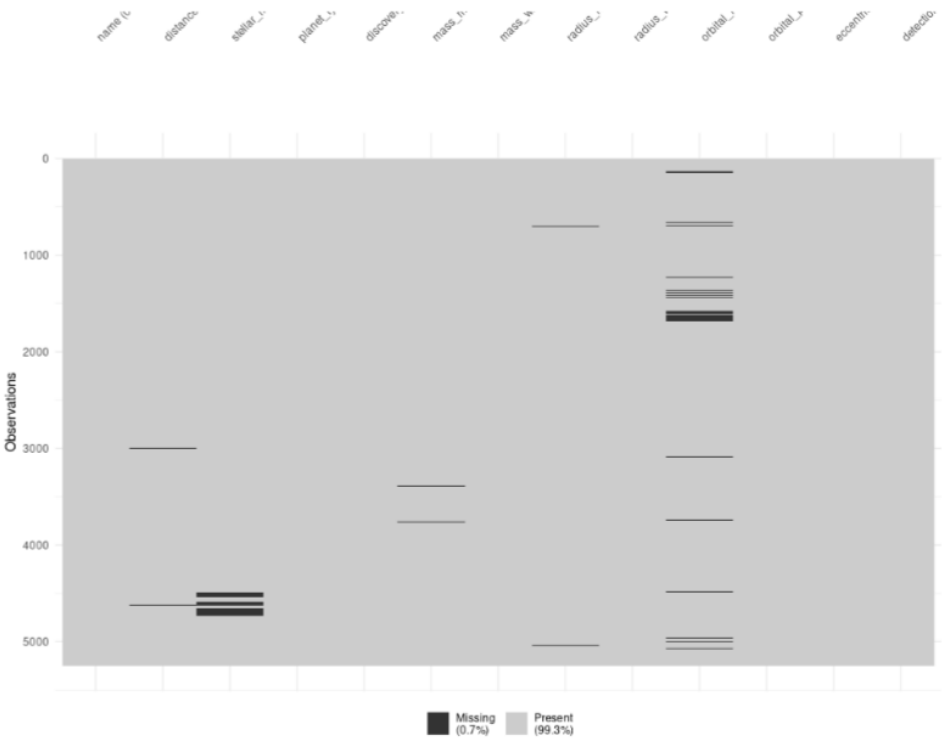

Figure 10 – Représentation de l'absence de valeurs par variable explicative (zones noires = valeurs manquantes).

3.2. Analyse exploratoire

3.2.1. Distribution temporelle des découvertes

À l'aide de la Figure 11, on pouvait déjà observer deux années pour lesquelles la découverte de planètes hors de notre système solaire fut un succès. En effet, en 2014 et 2016, la découverte de nouvelles planètes fut importante grâce en partie au télescope spatial Kepler et à la technique statistique nommée « vérification par multiplicité ».

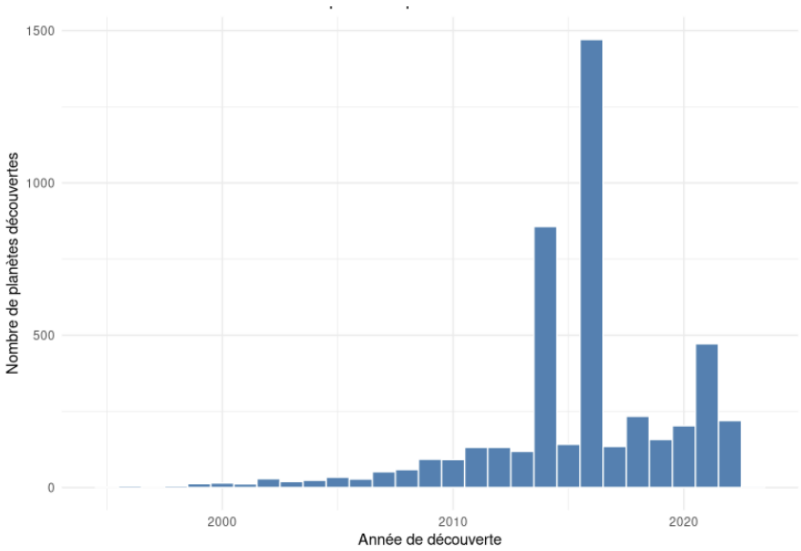

Figure 11 – Distribution du nombre de découvertes d'exoplanètes par année.

3.2.2. Répartition des classes après nettoyage

La Figure 12 montre la répartition des types de planètes après le nettoyage des données : *Neptune-like* représente environ 35 %, *Super Earth* 31 %, *Gas Giant* 30,1 % et *Terrestrial* 3,8 %.

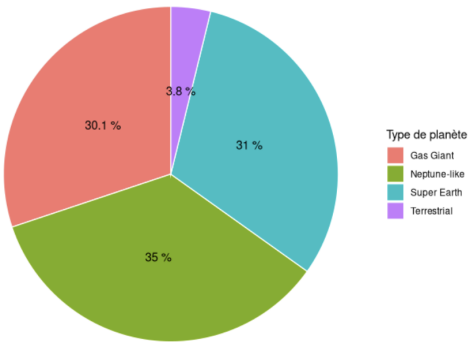

Figure 12 – Répartition du type de planètes après nettoyage des données.

NB : la signification des classes :

1. Gas Giant : Une planète géante constituée principalement d'hydrogène, d'hélium ou d'autres gaz. (ex. Jupiter, Saturn...)
2. Neptune-like : Une planète dont la taille est proche de celle de Neptune ou d'Uranus, composé également de hydrogène et hélium avec un noyau rocheux et de métaux lourds.

3. Super Earth : Une planète dont la masse varie d'environs 1 à 10 fois celle de la Terre avec une composition similaire à celle-ci (*fer, eau et autres matériaux*)
4. Terrestrial : Une planète compacte avec une surface rocheuse comme celle de la Terre (*Mercure, Venus, Terre, Mars..*)

3.3. Nettoyage et préparation du jeu de données

L'une des premières étapes avant même de traiter ce jeu de données était de le nettoyer et le préparer pour pouvoir utiliser et appliquer les multiples algorithmes de classification et d'observation. Dans le cas de notre dataset, plusieurs problématiques ont été trouvées pouvant mettre en difficulté les algorithmes.

3.3.1. L'absence de certaines données

L'une des premières actions réalisées sur le jeu de données était l'observation des valeurs manquantes dans plusieurs colonnes et variables explicatives. Pour ce faire, on a utilisé la fonction `vis_miss(data)` de la librairie `naniar`, nous permettant d'observer en noir l'absence de ces valeurs.

Plusieurs choix étaient possibles dans la résolution de ce problème, comme la complétion de ces valeurs par des estimations ou des calculs à l'aide d'autres variables. L'un des soucis, si l'on solutionnait comme cela, aurait pu être la création de biais dans nos futures prédictions de classification. Dans notre cas, nous avons opté pour une solution plus radicale avec l'élimination des échantillons pour lesquels il manquait des valeurs dans une de leurs colonnes, à l'aide de la fonction `na.omit(data)` supprimant les N/A dans notre jeu.

Après la suppression de ces valeurs, nous avons pu remarquer que la proportion de planètes classées comme `Unknown` est passée à 0 %. Nous créant par la même occasion un problème avec un type de planète vide. En effet, c'est un problème car dans certains procédés algorithmiques comme les arbres de décisions ou encore les Random Forest, avoir une classe vide empêche le bon déroulement de l'algorithme. Pour ce faire, après plusieurs recherches dans la documentation de R, on a trouvé la fonction `droplevels()` permettant de supprimer les facteurs dits « non-utilisés ».

3.3.2. Préparation et création de variables explicatives

Maintenant que notre jeu de données était propre, il nous restait une problématique majeure : nos échantillons n'étaient pas tous à la même échelle. En effet, la masse et le rayon des planètes étaient mesurés soit à l'échelle de Jupiter, soit à celle de la Terre. Pour ce faire, on a décidé de mettre l'ensemble de nos données à l'échelle de la Terre à l'aide de deux coefficients :

— 1 rayon jovien = 11,2 rayons terrestres

— 1 masse jovienne = 317,8 masses terrestres

Ces deux coefficients ont été calculés à l'aide de deux rapports : le premier entre la masse de Jupiter ($\approx 1{,}898 \times 10^{27}\,\text{kg}$) et celle de la Terre ($\approx 5{,}97 \times 10^{24}\,\text{kg}$), et le deuxième de même avec les valeurs des rayons. Après la création de nos deux colonnes `mass_earth` et `radius_earth`, on a supprimé les variables `mass_multiplier`, `mass_wrt`, `radius_multiplier` et `radius_wrt`.

3.3.3. Échelles des données

Après quelques traitements de nos données, une problématique est revenue plusieurs fois : celle de l'échelle dans les données d'astrophysique. On n'a pas toujours les mêmes échelles utilisées entre les exoplanètes, et ça peut devenir problématique notamment pour certains algorithmes ou procédés, notamment par exemple l'utilisation des modèles linéaires.

On a donc décidé de créer un nouveau dataset en utilisant la fonction logarithme sur les colonnes de notre jeu de données déjà préparé : `mass_earth`, `radius_earth`, `distance`, `orbital_period`, `orbital_radius`. Nous n'avons pas eu besoin d'appliquer cette transformation à la magnitude stellaire car elle a déjà une fonction logarithme appliquée, ni à l'excentricité car elle est déjà comprise entre 0 et 1, ni à l'année de découverte qui n'est pas vraiment pertinente.

3.3.4. Normalisation des données

La normalisation des données est essentielle pour l'utilisation de plusieurs algorithmes se basant sur la distance comme K-NN, SVM et d'autres. Mais on a aussi pris le soin de garder un jeu de données non normalisé pour les algorithmes ou procédés ne demandant pas nécessairement un jeu normalisé.

3.4. Réduction de dimension et observation visuelle

La première étape pour essayer de classifier nos exoplanètes était d'essayer d'observer visuellement des potentiels clusters lors d'une projection en 2 dimensions lors d'une réduction de dimensions. En effet, notre jeu de données propre est en dimension 8, mais il était difficile d'obtenir directement une première intuition sur nos données et de voir si on pouvait obtenir un nombre de clusters, pouvant par la même occasion être utilisé pour notre algorithme K-means.

3.4.1. L'analyse en composantes principales (ACP)

L'ACP peut être une première approche malgré le peu de dimensions de notre jeu. Mais malgré un centrage et une réduction des données, l'ACP, essayant de trouver une

variance globale maximum, a échoué à cause de certaines valeurs extrêmes du jeu, et de nombreux points ont été « écrasés » par les extremum, comme montré en Figure 13. On atteignait déjà la limite de l'ACP avec cette projection qui ne permettait pas d'obtenir de résultat pertinent avec la représentation de clusters.

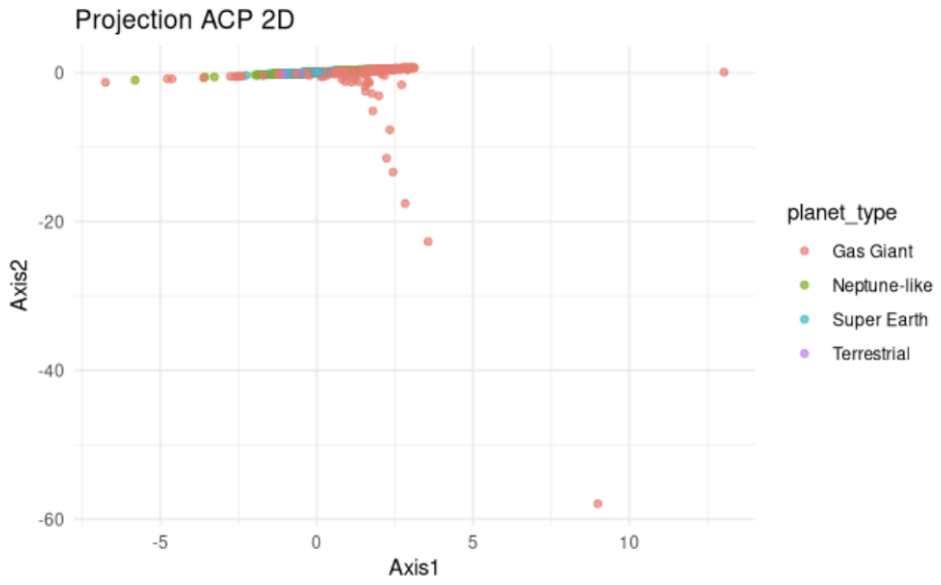

Figure 13 – Exoplanètes projetées en 2D — méthode ACP.

3.4.2. L'algorithme t-SNE

Contrairement à l'ACP, le t-SNE est une méthode de réduction de dimension non linéaire qui cherche à conserver les distances locales, autrement dit deux points proches dans l'espace d'origine resteront proches dans l'espace réduit. Cette méthode, se déroulant en deux étapes, calcule la probabilité qu'une paire de points soit voisine en suivant une distribution gaussienne, puis la modélise dans un espace réduit (2D ou 3D) en appliquant la distribution de Student.

La perplexité, représentant le nombre de voisins pris en considération pour chaque point, permet de capturer les groupes locaux (si faible) ou globaux (si élevée). Nous avons fait varier cet hyperparamètre afin de trouver la projection offrant la meilleure séparation visuelle des classes pour notre jeu de données (Figure 14).

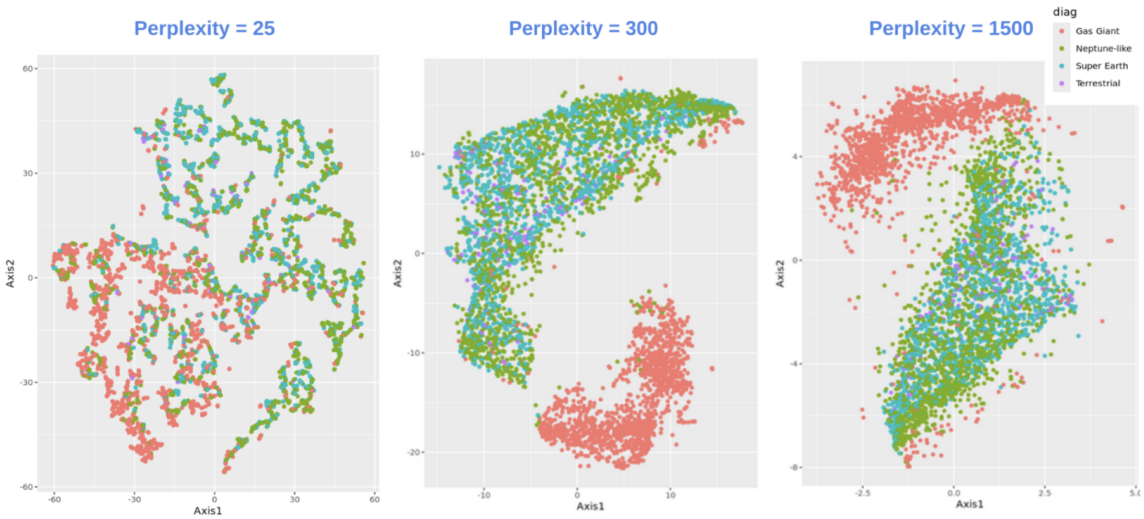

Figure 14 – Projection 2D par t-SNE pour trois valeurs de perplexité (25, 300, 1500).

À partir d'une perplexité de 300, le t-SNE révèle deux ensembles séparés : les *Gas Giants* forment un cluster bien distinct contre le reste des classes mélangées. Les planètes *Terrestrial* apparaissent surtout par points distincts sans cluster concret, sûrement dû à leur faible présence dans le jeu de données (3,8 %), mais tout de même assez proches des *Super Earth* car ils partagent une caractéristique commune (voir analyse de l'arbre de décision).

Avec une perplexité de 1500, les *Gas Giants* restent séparés mais les autres classes restent mélangées. Cette projection apporte moins d'information qu'à 300 car la perplexité trop élevée dépasse le nombre de voisins possibles, ce qui explique pourquoi les *Gas Giants* sont plus dispersés.

Sans les labels, l'application du t-SNE sur notre jeu nous permettrait de séparer les points en au moins deux classes : *Gas Giants* contre les 3 classes restantes. Cette méthode n'est pas suffisante pour séparer *Neptune-like*, *Super Earth* et *Terrestrial*. Ces méthodes, ACP et t-SNE, ne permettent pas de prédire directement les labels, mais nous donnent des indications utiles pour la suite. Nous avons aussi envisagé la méthode de l'autoencodeur pour réduire la dimension en espace latent, mais dans notre cas nous avions trop peu de dimensions pour obtenir quelque chose de significatif.

3.5. Modélisation et classification

Dans cette partie de classification, il existe plusieurs approches possibles. En effet, on a essayé de montrer les différents procédés possibles comme le K-means, SVM, modèles linéaires, arbres de décisions, les forêts aléatoires ou encore une dernière approche plus puissante avec les réseaux de neurones. Pour la majorité de nos algorithmes, on a utilisé le principe de la validation croisée en séparant notre jeu de données en deux sous-ensembles

(80/20 %), l'un nommé `data.train` et l'autre `data.test` : on a réalisé l'apprentissage de notre modèle sur le premier ensemble et on l'a testé sur le second. On a évalué nos performances sur `data.test` pour pouvoir comparer à la fin entre chaque algorithme.

3.5.1. Modèles linéaires

Pour commencer par une méthode simple à mettre en œuvre et à interpréter, nous avons choisi d'expérimenter la classification par la méthode des modèles linéaires. Celle-ci consiste à estimer la probabilité d'appartenir à une classe à partir d'une combinaison linéaire de features choisies (e.g. $\mathbb{P}(Y = 1) = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n$). Concrètement, on ajuste un hyperplan (ie. une droite) sur une variable cible binaire 0/1 de sorte à minimiser la variance résultante.

Pour rappel, afin de mener à bien l'application de cette méthode à notre dataset, nous avons précédemment nettoyé celui-ci de ses trous, supprimé les classes de planète "Unknown" et mis à l'échelle de la Terre les masses des planètes et leurs rayons respectifs.

Comme visualisable dans le graphique ci-dessous, nous avons vérifié à l'aide d'une échelle log10 que certaines features - ici la masse et le rayon des exoplanètes - suivaient une relation approximativement linéaire, validant notre ambition d'appliquer la prédiction de classe par modèle linéaire sur celles-ci.

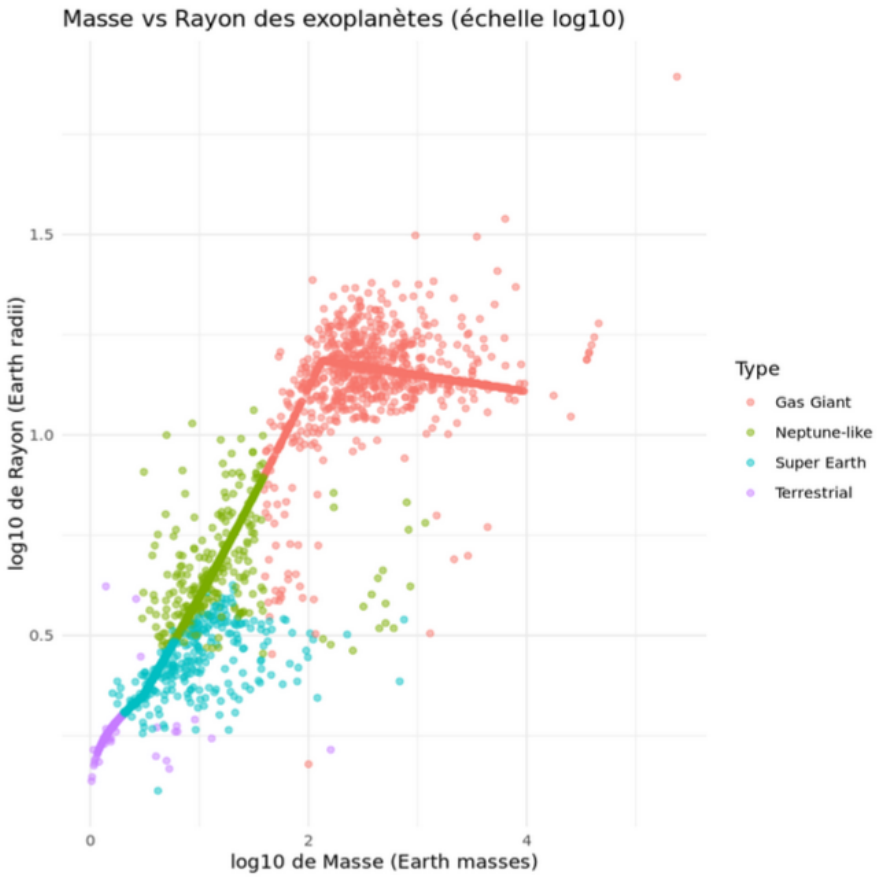

Figure 15 – Représentation du rayon d'une exoplanète en fonction de sa masse.

De plus, puisque cette méthode nécessite d'avoir une cible de prédiction binaire, nous avons rajouté une colonne par classe de prédiction (ie. type de planète), valant 1 si la planète est du type indiqué, 0 sinon (is_gas_giant, is_super_earth, ...). Cet encodage a été produit par le code suivant :

```
data$is_gas_giant   <- as.numeric(data$planet_type == "Gas Giant")
data$is_super_earth <- as.numeric(data$planet_type == "Super Earth")
data$is_neptune     <- as.numeric(data$planet_type == "Neptune-like")
data$is_terrestrial <- as.numeric(data$planet_type == "Terrestrial")
```

Figure 16 – Fonctions utilisées pour les nouvelles variables

Les pré-requis étant atteint, nous avons pu entraîner nos modèles linéaires grâce à la fonction lm(classe ~features, data = data.train) de R. Pour chaque type de planète encodée binairement, nous avons choisit de le prédire en fonction des variables explicatives suivantes : *distance, stellar_magnitude, discovery_year, orbital_radius, orbital_period, eccentricity, mass_earth, radius_earth*. Remarquons que l'on n'utilise pas prediction_method ici, car c'est un biais observationnel, pas une propriété intrinsèque de la planète.

Nos modèles étant entrainés, nous avons pu réaliser des tables de confusion ci-dessous pour chacun, permettant de mesurer leurs performances concrètes sur le jeu de test. Nous

y constatons de bonnes performances des modèles de prédiction de Gas Giant (14/953 erreurs) et Terrestrial (37/953 erreurs), mais celles de Super Earth (223/953 erreurs) et Neptune-like (354/953 erreurs) restent à désirer.

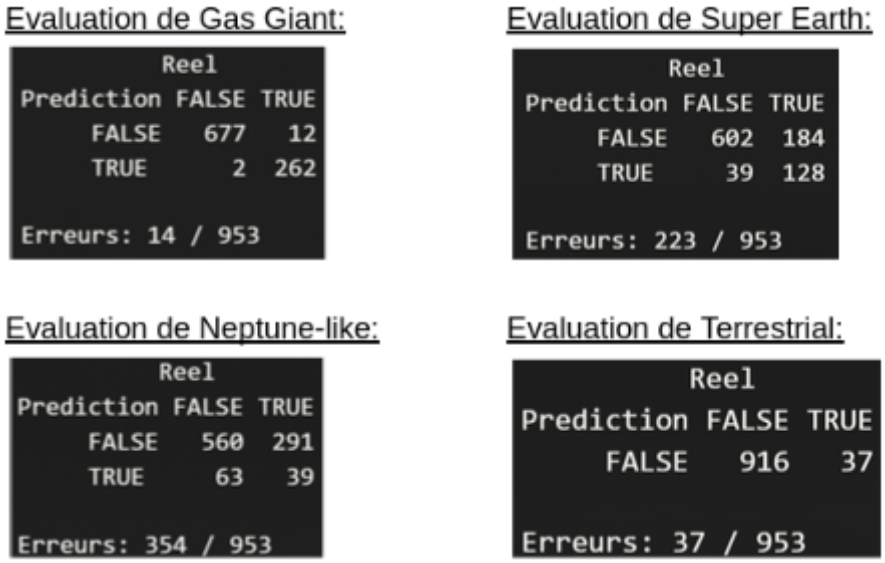

Figure 17 – Résultat des prédictions en fonction des vraies classification

Afin de tenter d'améliorer les performances de cette méthode, on étend la dimension du jeu de données en y ajoutant des colonnes quadratiques permettant de capturer les relations non-linéaires. Nous avons choisit les termes quadratiques suivants, puis avons re-séparé nos données en un jeu d'entraînement et un jeu de test :

```
# Termes quadratiques
data$mass2       <- data$mass_earth^2
data$radius2     <- data$radius_earth^2
data$mass_radius <- data$mass_earth * data$radius_earth
```

Figure 18 – Ajouts de nouvelles variables quadratiques

De la même manière que nos modèles non-étendus, nous avons entraîné nos modèles linéaires étendus sur le jeu d'entraînement, puis avons recueilli leurs performances sur le jeu de test.

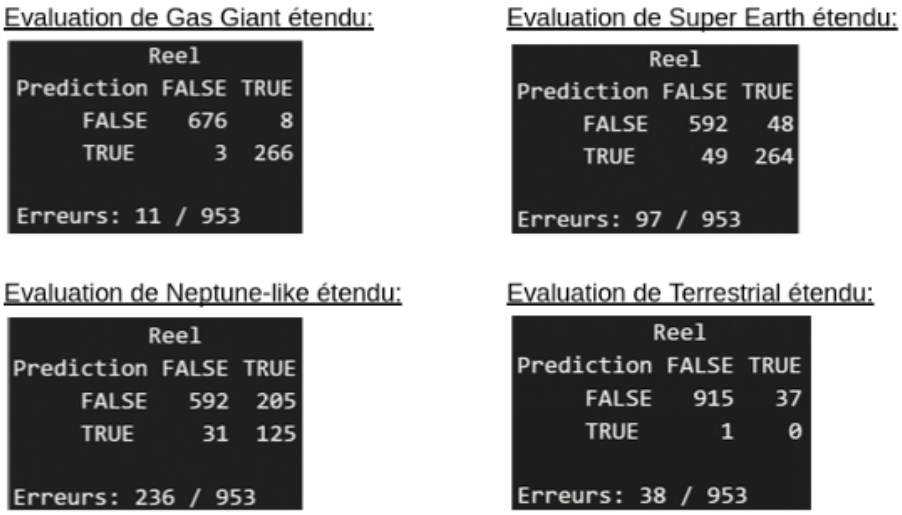

Figure 19 – Résultat des prédictions en fonction des vraies classification (avec l'ajout des variables quadratiques)

Les résultats sont meilleurs ici (ie. taux d'erreur nettement plus faible sur certains modèles étendus par rapport à leurs homologues non-étendus), démontrant l'intérêt de l'extension de dimension avec des colonnes quadratiques.

```
--- Erreurs classe simple vs étendue ---
Gas Giant    : Simple 28 | Etendue 20
Super Earth  : Simple 186 | Etendue 104
Neptune-like : Simple 378 | Etendue 301
Terrestrial  : Simple 37 | Etendue 37
```

```
--- Accuracy par classe: simple vs étendue ---
Gas Giant    : Simple 97.06 % | Etendue 97.9 %
Super Earth  : Simple 80.48 % | Etendue 89.09 %
Neptune-like : Simple 60.34 % | Etendue 68.42 %
Terrestrial  : Simple 96.12 % | Etendue 96.12 %
```

Figure 20 – Observation et comparaison des erreurs et de la précision (avant et après l'ajout des variables quadratiques)

Bilan : L'utilisation des modèles linéaire fut une première bonne approche pour essayer de classer nos planètes, mais on voit quand même pour chaque type de planètes à prédire, un taux d'erreur trop élevé, malgré de bons résultats uniquement pour les planètes gazeuses (*Gas Giant*) du à sa facilité à être isolé. L'ajout des variables quadratiques fut une bonne idée avec une amélioration générale des résultats, mais gardant encore un taux d'erreur trop élevé.

3.5.2. K-NN

L'un des algorithmes de classification les plus connus et utilisés est celui du K-NN (*k-nearest neighbors*), une méthode non paramétrique utilisée pour la classification et la régression. Son fonctionnement est très simple : on imagine essayer de classifier un nouveau point. On commence par prendre une valeur de K, nous permettant de prendre les K voisins les plus proches, on calcule donc les distances entre tous les points et celui à définir. Après avoir récupéré nos voisins, on récupère la classe majoritaire parmi eux et on l'assigne à notre point.

Dans notre situation, pour appliquer notre algorithme, on a besoin de normaliser les données pour faire nos calculs de distances à l'aide de la fonction *scale()*, au risque d'avoir des variables pas à la même échelle écrasant d'autres variables. L'une des difficultés du K-NN est le choix de la valeur de K pour obtenir la meilleure performance. Plutôt que de le tester à la main, on a utilisé une boucle de 1 à 30, avec un pas de 2 à chaque itération car K doit être impair pour éviter les cas où les voisins du point à classer présentent une répartition égale des classes. À chaque tour de boucle, on a calculé la précision (nombre de bonnes prédictions parmi le total de prédictions) stockée dans un vecteur R, puis on a affiché un graphique pour observer la précision en fonction de la valeur de K (Figure 22).

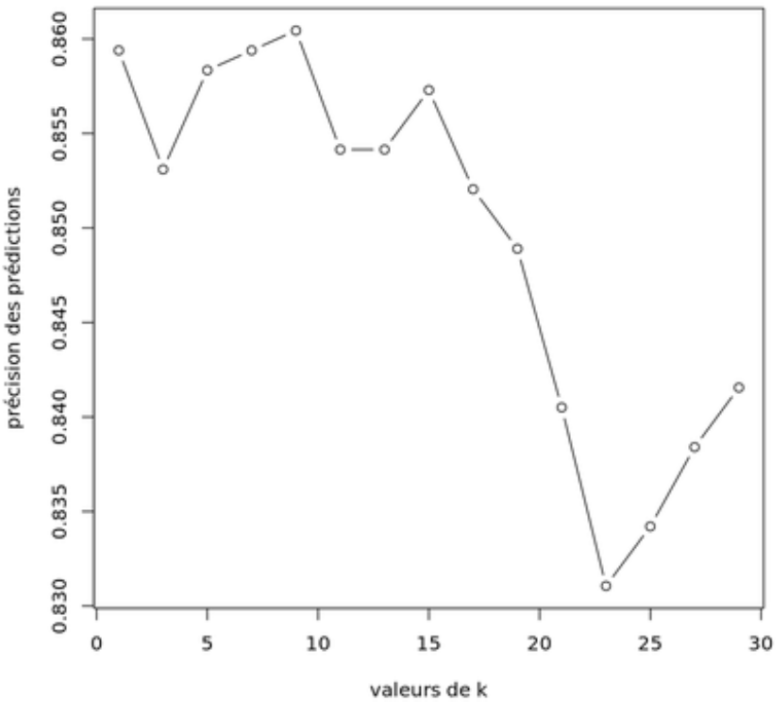

Figure 21 – Précision du K-NN sur l'ensemble de test en fonction de la valeur de K.

Attention, dû au choix aléatoire de nos ensembles d'apprentissage et de test, le graphique n'est jamais le même. Mais, sur celui-ci, on observe bien le phénomène de sur-apprentissage

avec un K faible et de sous-apprentissage avec un K élevé. Dans notre cas, la valeur 9 semble la meilleure, pour notre algorithme. Voici donc une table de confusion :

predictions	Gas Giant	Neptune-like	Super Earth	Terrestrial
Gas Giant	269	5	0	0
Neptune-like	7	270	43	0
Super Earth	0	52	271	25
Terrestrial	0	0	1	10

Figure 22 – Matrice de confusion - K-NN avec K = 9

Bilan : On observe une prédiction correcte avec une précision d'environ 87%. Mais, avec cette méthode, on peut voir que les planètes avec pour classes "*Super Earth*" et "*Neptune-like*" ont du mal à bien être différencié avec 52 "*Neptune-like*" classé comme des "*Super Earth*" ou encore 43 "*Super Earth*" défini comme des "*Neptune-like*". Mais par contre les planètes gazeuses arrivent à être bien classées avec seulement 7 erreurs de planètes prédites comme des planètes de type "*Neptune-like*", cela dit, c'est plutôt logique lorsqu'on a observé précédemment les graphiques de la méthode T-SNE.

3.5.3. SVM

Comme le pour le K-NN, Le SVM (*Support Vector Machine*) est un algorithme qui cherche à séparer les données en classes en trouvant un hyperplan qui maximise la marge entre les points les plus proches de chaque classe. Si aucune séparation n'est possible, le SVM les projette dans une dimension supérieure où cette séparation devient possible, en utilisant un noyau qui existe en différentes variantes. Comme nous avons 4 classes et que le SVM ne supporte que des classifications binaires, l'algorithme SVM utilise la stratégie *One vs One* en créant 16 paires de classification (*Gas Giant* vs *Neptune-like*, *Gas Giant* vs ...). La classe prédite est celle qui à les meilleures performances.

Noyau linéaire ($C = 1$). Le modèle a trouvé 1 200 vecteurs de support, ce qui montre que les points sont difficilement séparables. Le SVM a du mal à définir des marges claires entre les classes et s'appuie sur beaucoup de points. L'accuracy de 55,4 % est faible pour 4 classes. Le noyau linéaire n'est pas suffisant pour séparer les classes, ce que le t-SNE nous avait déjà confirmé.

La matrice de confusion (Figure 24) montre que le SVM linéaire classe bien les *Gas Giants* (367/401 corrects) avec des erreurs de classification pour *Neptune-like* et *Super Earth* entre elles (212 *Neptune-like* classés en *Super Earth*). Quant à *Terrestrial*, ils sont très mal classés avec seulement 16/192 corrects.

Figure 23 – Résultats des prédictions pour les classes

— **Gas Giant** : sensibilité de 91,5 % et précision de 99,7 % — le SVM détecte bien les *Gas Giants* et se trompe rarement lorsqu'il prédit cette classe.
— **Terrestrial** : sensibilité de 100 % mais précision de seulement 8,2 % — toutes les vraies *Terrestrial* sont détectées, mais le modèle prédit très souvent à tort cette classe, générant des faux positifs. Nous reviendrons plus tard sur pourquoi elle est souvent prédite comme *Super Earth* (voir arbre de décision).
— **Super Earth** : sensibilité de 26,7 % et précision de 22,9 % — le SVM n'arrive pas à distinguer cette classe de *Neptune-like*.

Le taux d'erreur global de ce modèle est de 44,59 %, ce qui reste élevé.

pred_test	Gas Giant	Neptune-like	Super Earth	Terrestrial	Unknown
Gas Giant	367	1	0	0	0
Neptune-like	29	81	0	0	0
Super Earth	4	212	64	0	0
Terrestrial	1	2	176	16	0
Unknown	0	0	0	0	0

[1] "Accuracy : 55.4039874081847 %"

Figure 24 – Matrice de confusion — SVM à noyau linéaire ($C = 1$, accuracy = 55,4 %).

Noyau radial ($C = 1$). Comme le jeu de données était toujours difficile à séparer linéairement, nous avons appliqué le noyau radial avec des séparations non linéaires. Le modèle a trouvé 1 674 vecteurs de support avec une amélioration de l'accuracy à 71,98 %, soit un taux d'erreur de 28,02 %.

Les *Super Earths* sont beaucoup mieux classées en passant de 64 correctement classées avec le noyau linéaire à 238 avec le noyau radial, mais restent souvent confondues avec les *Neptune-like* (210 sur 464). Les *Terrestrial* sont classées à 0 de correcte et se retrouvent sur la ligne de la classe *Super Earth*, ce qui s'explique par leur faible représentation dans le jeu de données et par le fait que la frontière du SVM radial ne parvient pas à les isoler des autres classes (Figure 25).

```
pred_test      Gas Giant Neptune-like Super Earth Terrestrial Unknown
  Gas Giant          365           3           0           0       0
  Neptune-like        31          83           2           0       0
  Super Earth          5         210         238          16       0
  Terrestrial          0           0           0           0       0
  Unknown              0           0           0           0       0

[1] "Accuracy : 71.9832109129066 %"
```

Figure 25 – Matrice de confusion — SVM à noyau radial ($C = 1$, accuracy = 71,98 %).

3.5.4. Arbre de décision

Un arbre de décision est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. C'est une suite de questions simples qui permet de prendre une décision finale, prenant la forme d'un arbre avec des nœuds (une question, ex. `mass_earth` ≥ 38), des branches (la réponse vrai/faux) et des feuilles (la décision, classe ou valeur). Le choix de la meilleure séparation utilise l'indice de Gini ou l'entropie, et répète le processus récursivement jusqu'au point où les données sont bien séparées ou qu'un point d'arrêt soit atteint (`tree_depth` ou `min_n`).

Pour notre jeu de données, nous avons laissé l'arbre se construire sans contrainte de profondeur, obtenant l'arbre suivant (Figure 26).

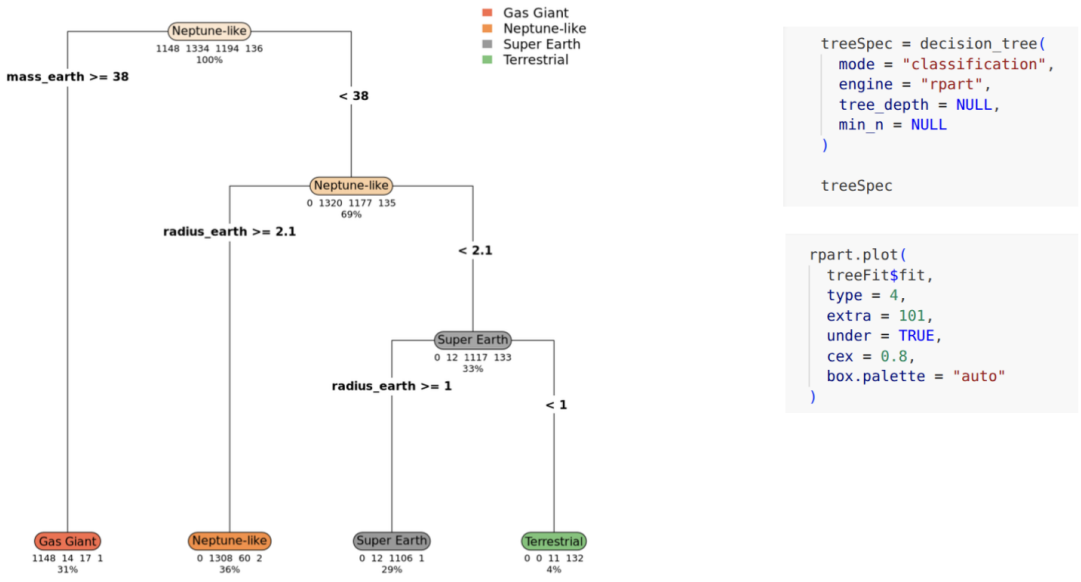

Figure 26 – Arbre de décision obtenu sans contrainte de profondeur.

On observe 3 suites de questions dont la première coupure se fait sur `mass_earth` ≥ 38, qui classe la planète comme *Gas Giant*. Puis pour les planètes moins massives (< 38), les

questions sont focalisées sur `radius_earth` : si $\geq 2{,}1$ alors c'est une *Neptune-like*, sinon parmi les restantes, si ≥ 1 alors c'est une *Super Earth*, sinon une *Terrestrial*. On en tire la conclusion que **la masse et le rayon sont les deux variables les plus discriminantes** pour classer les planètes.

	Gas Giant	Neptune-like	Super Earth	Terrestrial
Gas Giant	287	3	3	0
Neptune-like	0	331	14	0
Super Earth	0	2	266	0
Terrestrial	0	0	0	47

```
[1] "Accuracy : 97.691500524659 %"
```

Figure 27 – Matrice de confusion - Arbre de décision (*sans limite de profondeur - accuracy = 97,69 %*)

La matrice de confusion montre une classification presque parfaite avec une accuracy de 97,69 % (*on peut noter que ceci est plutôt peu courant*). Les seules erreurs restent entre *Neptune-like* classé comme *Super Earth* (14/345) et inversement (2/268), ainsi que quelques *Gas Giants* classés comme *Neptune-like* ou *Super Earth* (6/293). La seule classe parfaitement classée (47/47) est *Terrestrial*.

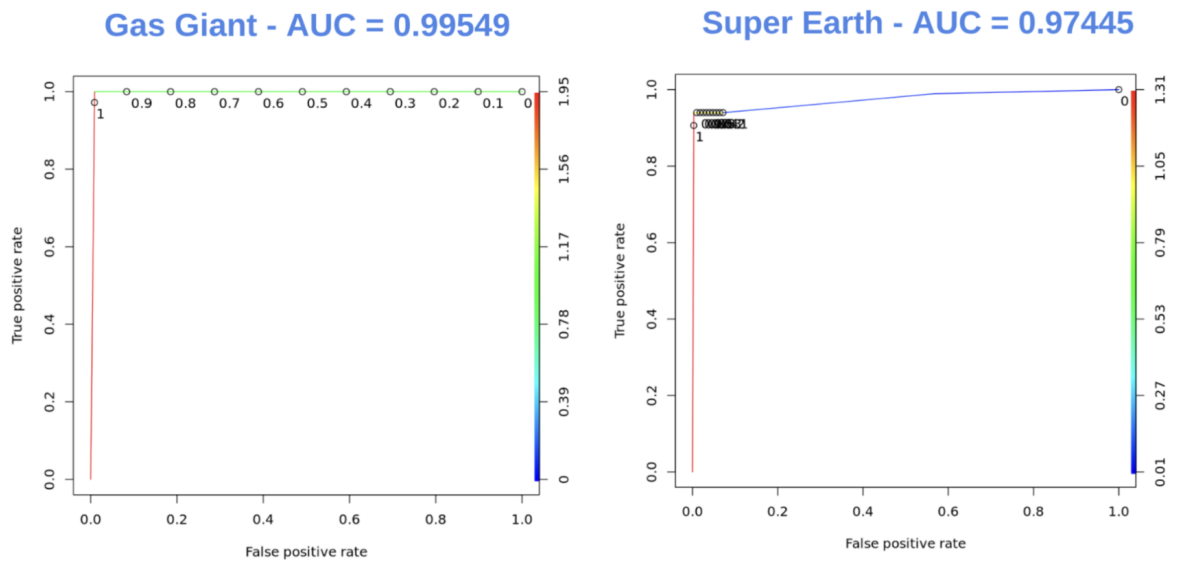

Figure 28 – Courbes ROC de l'arbre de décision — *Gas Giant* (AUC = 0,995) et *Super Earth* (AUC = 0,974).

On observe des courbes ROC presque parfaites pour *Gas Giant* et *Super Earth* avec des AUC proches de 1 (Figure 28). Contrairement au SVM, l'arbre de décision est plus

performant avec une accuracy de 97,7 %, reposant sur la masse et le rayon pour classer les planètes, ce qui est cohérent avec les caractéristiques physiques réelles des exoplanètes.

3.5.5. Random Forest

Comme on a pu observer dans la partie précédente, les arbres de décision sont faciles à utiliser, mais ils ont une principale contrainte et limitation, le sur-apprentissage. En effet, ils marchent bien sur les données de l'ensemble d'apprentissage mais ont du mal à généraliser sur l'ensemble de test. On va donc introduire et utiliser les forêts aléatoires pour passer cette limitation.

Une forêt aléatoire (ou *RandomForest*) se représente par un ensemble de plusieurs arbres de décision chacun indépendant entre eux. Pour chaque nouvel arbre, on va récupérer un jeu d'apprentissage différent avec des tirages avec remise dans le jeu d'apprentissage d'origine, on appelle cette étape : le *bootstrap*.
Une fois nos arbres de décision construits et entraînés grâce au Bootstrapping, chaque nouvel échantillon est présenté à chacun des arbres et la prédiction finale est celle-qui reviens le plus fréquemment (choix majoritaire), on appelle cette étape : *Aggregating*.

Contrairement aux simples arbres de décisions, K-NN ou le SVM, on utilisera donc notre jeu de données complet ("data"), dû au principe de bootstrapping et d'Aggregating.

On décide donc d'appliquer l'algorithme et la fonction RandomForest() sur tous notre jeu, en enlevant simplement la colonne donnant le nom de nos exoplanètes, car cela n'est pas très pertinent lors de la créations de nos arbres et du coup de nos règles mathématiques.

On affiche donc la fonction : fitRF <- randomForest(planet_type ~ , data = data)

```
Call:
 randomForest(formula = planet_type ~ ., data = data)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 3

        OOB estimate of  error rate: 2.01%
Confusion matrix:
             Gas Giant Neptune-like Super Earth Terrestrial class.error
Gas Giant         1429            2           3           1 0.004181185
Neptune-like         8         1639          23           0 0.018562874
Super Earth          0           45        1429           3 0.032498307
Terrestrial          1            2           8         172 0.060109290
```

Figure 29 – Résultat de l'algorithme pour la forêt aléatoire (*nbtree = 500*)

La première chose qu'on peut remarquer est qu'avec nos données OOB (*Out of Bag*),

on a un taux d'erreur faible avec 2.01 $\dot{\%}$ sur nos prédictions. On peut donc estimer qu'à partir de nouvelles exoplanètes, on aura 97.99 $\dot{\%}$ de chance de prédire correctement les planètes avec leur bonne classe. Par contre, on peut observer que les planètes gazeuses sont très bien prédite et classées avec seulement un taux d'erreur égal à 0.4 $\dot{\%}$, grâce à leurs caractéristiques physiques plutôt unique par rapport aux autres (Masse et diamètre). Ensuite, avec notre matrice de confusion, on voit que la majorité de nos erreurs est la frontière entre les planètes de type "*Super Earth*" et "*Neptune-like*" :

— 45 "*Super Earth*" on été classées comme des "Neptune like" (~1.8 % d'erreurs)
— 23 "*Neptune-like*" on été classées comme des "Super Earths" (~3.2 % d'erreurs)

Dans une forêt aléatoire, on retrouve naturellement la notion d'indice de Gini permettant de mesurer la pureté d'un nœud dans un arbre de décision. Le **Mean Decrease Gini**, de la figure ci-dessous, nous permet de voir à quel point une variable explicative a permis de faire diminuer l'impureté dans les 500 arbres.

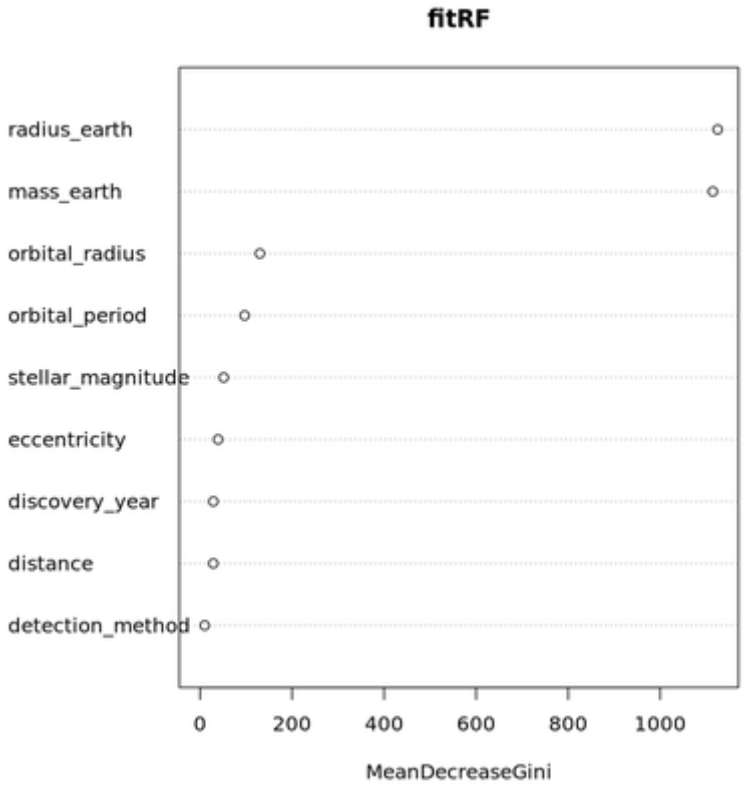

Figure 30 – Importance des variables explicatives à l'aide du *MeanDecreaseGini*

Le graphique au-dessus révèle et montre à quel point la masse et le rayon sont les variables les plus importantes pour caractériser le type d'une planète. On peut malgré tout noter que les variables "orbital_radius" et "orbital_period", donc les caractéristiques orbitales d'une planète, ont un faible rôle dans la séparation des classes.

L'un des problèmes de l'algorithme RandomForest est son potentiel coût de calcul dû

au nombre d'arbres. Pour ce faire, on peut illustrer l'évolution du taux d'erreur OOB par rapport aux nombres d'arbres comme illustré ci-dessous :

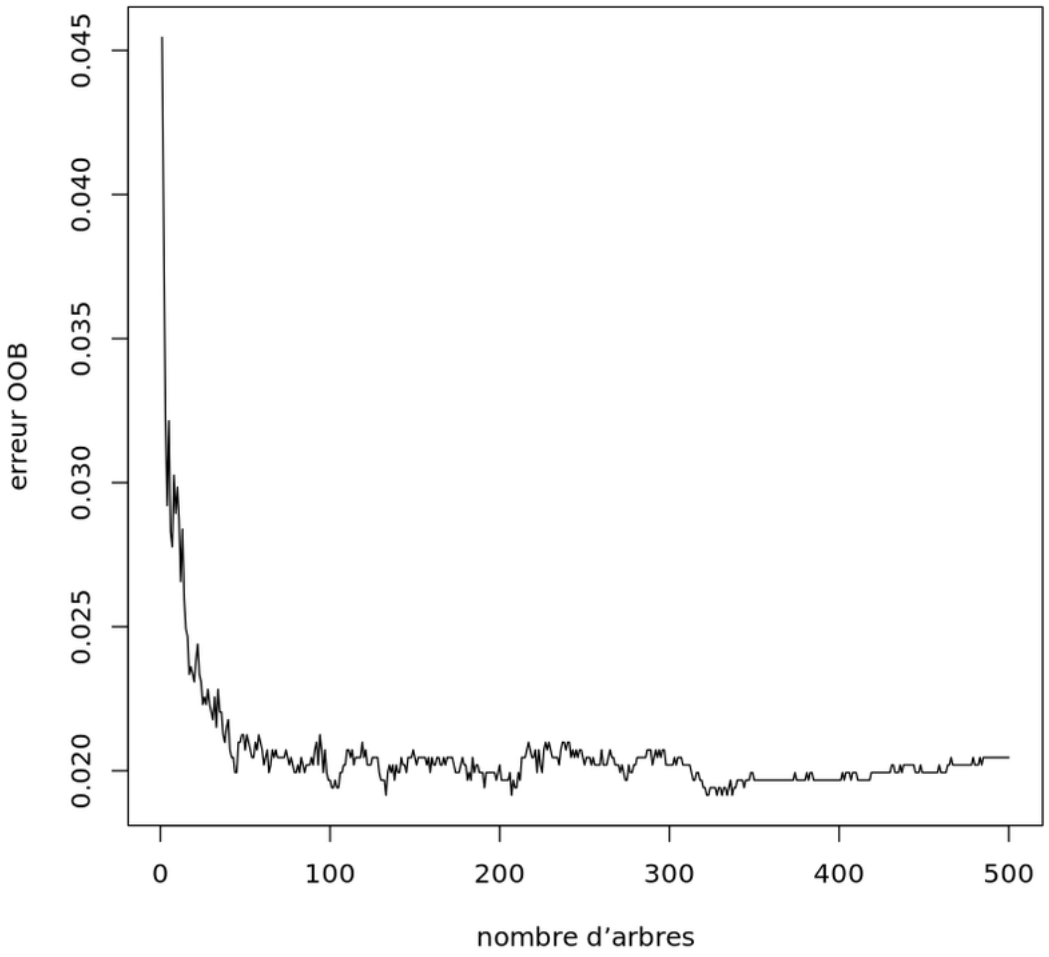

Figure 31 – Illustration du taux d'erreur OOB (*Out of Bag*) en fonction du nombre d'arbres

Comme le montre la courbe, on voit que le taux d'erreur OOB chute drastiquement et se stabilise à un seuil aux alentours de 2 %. Initialement la fonction RandomForest() initialise le nombre d'arbres à 500 avec paramètre "*ntree*" qui est une bonne valeur, mais on pourrait choisir une valeur plus proche des 350. A l'inverse, mettre le paramètre a une valeur plus grande que 500 serait finalement inutile et coûteux pour une amélioration presque absente.

Bilan : Pour conclure, les forêts aléatoires sont une bonne solution. En effet, avec un taux d'erreur de 2 %, on peut facilement affirmer que c'est notre meilleure solution à l'instant t . De plus, on peut remarquer une bonne classification pour les planètes dite : "*Terrestrial*" avec un faible taux d'erreur d'environ 6 %, d'après la figure 29, alors que les

méthodes précédentes (*K-NN,SVM...*) avaient beaucoup plus de mal à les prédire.

3.5.6. Réseau de neurones

Passons à une méthode supervisée plus complexe, typique du machine learning : les réseaux de neurones, dont l'un de ses objectifs est l'approximation de n'importe quelle fonction continue.

Explication du processus : En pratique, on exploite une architecture de couches de neurones successives (*eg. entrée -> couche(s) cachée(s) -> sortie*), interconnectées entre couches, où l'information circule dans un seul sens. À chaque neurone, l'information est transformée selon des poids, revenant ainsi à calculer une somme pondérée des entrées du réseau en y ajoutant un biais : c'est la prédiction. Le réseau minimise une fonction de coût (erreur prédiction/réalité) en ajustant ses poids par descente de gradient et rétropropagation, de la sortie vers l'entrée.

L'une des problématique aux réseaux de neuronnes est son côté "*boîte noire*" empêche d'expliquer les prédictions, pouvant nuire à la confiance. De plus, elle est non déterministe (initialisation aléatoire des poids), chaque entraînement pouvant donner des résultats différents.

Application à notre jeu de données : On commence par procéder comme avec les modèles linéaires avec l'encodage binaire de chaque classe à prédire. Dans notre cas de classification, comme pour d'autres méthodes, nous savons, grâce au graphique exprimant à l'échelle log10 le rayon en fonction de la masse de chaque planète, qu'il existe une relation approximativement linéaire permettant de caractériser les types de planète. De plus, on ne doit pas oublier de normaliser les valeurs, pour ce faire, on applique une fonction à l'aide de *min()* et *max()* dont la formule est : *(x - min(x)) / (max(x) - min(x))*.

```r
# Normalisation des jeux
features <- c("distance", "stellar_magnitude", "discovery_year", "orbital_radius", "orbital_period", "eccentricity", "mass_earth",
"radius_earth")
normalize <- function(x) { (x - min(x)) / (max(x) - min(x)) }

data.train.norm <- as.data.frame(lapply(data.train[, features], normalize))
data.test.norm  <- as.data.frame(lapply(data.test[, features], normalize))
```

Figure 32 – Fonction programmé en R pour la normalisation

Les pré-requis étant rassemblés, on a pu débuter cette méthode par des entraînements de réseau de neurones, à 1 couche cachée, pour chaque variable cible binaire (*is_gas_giant, is_super_earth, ...*). Une fois de plus, on n'utilise pas la feature prediction_method ici, car

c'est un biais observationnel, pas une propriété intrinsèque de la planète. Nous pouvons visualiser les réseaux ci-dessous :

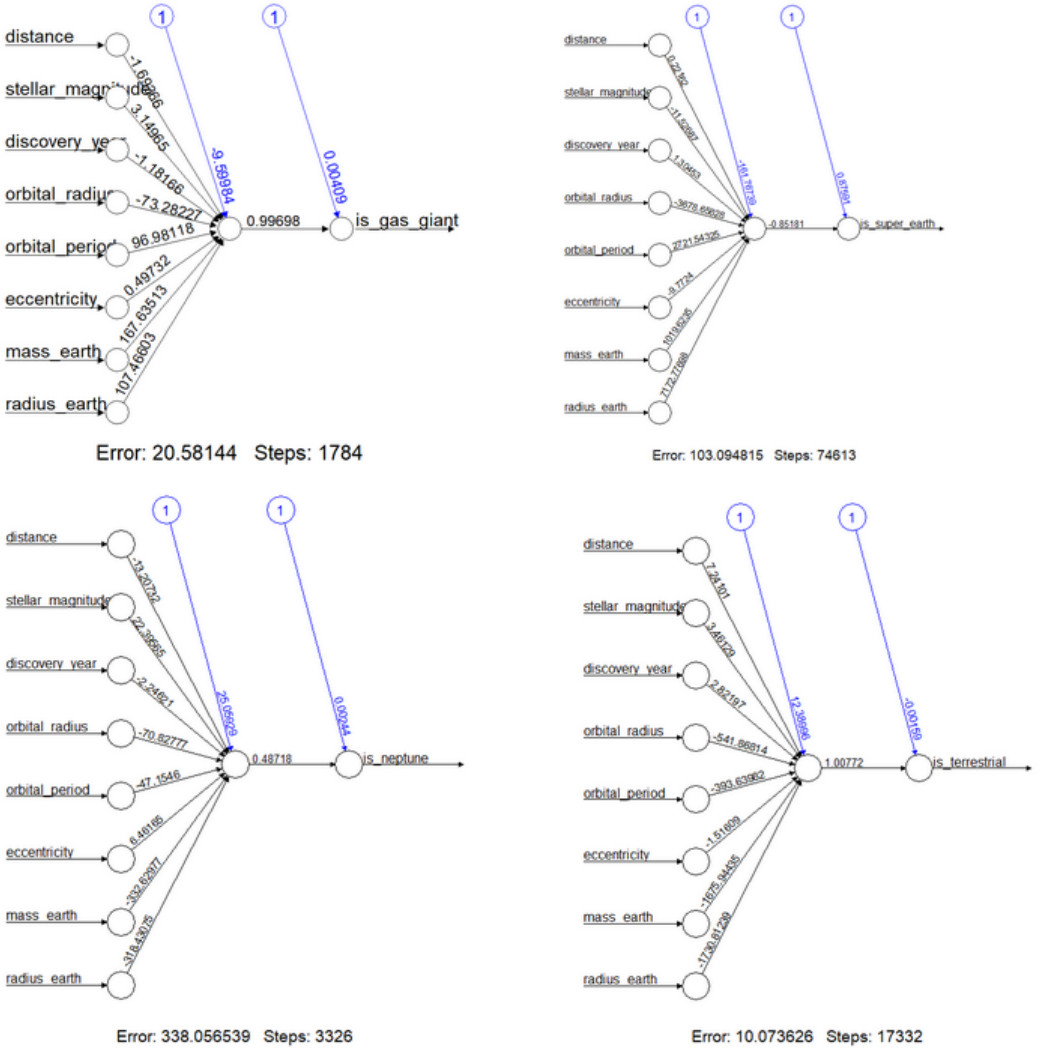

Figure 33 – Illustration des réseaux de neurones pour chaque classe

Voici les tables de confusion correspondantes sont les suivantes :

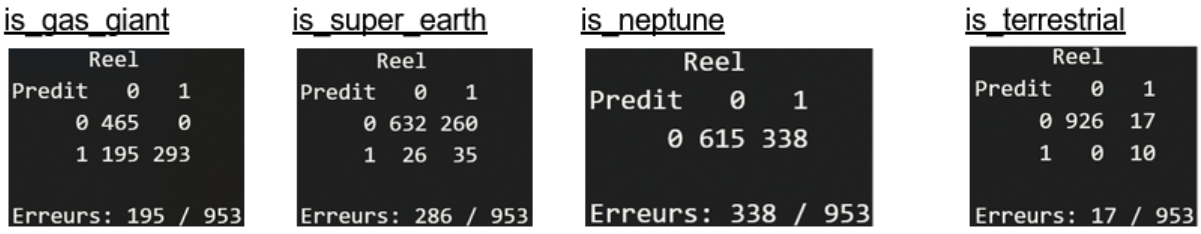

Figure 34 – Tables de confusions pour chaque classe

Celles-ci nous montrent qu'une unique couche cachée n'est pas suffisante pour prédire nos types de planète. On observe des performances faibles avec en particulier le cas pour

les "*Neptune-like*", où le réseau dédié n'arrive à en prédire aucune. On sait que les réseaux de neurones à une couche cachée sont souvent plus puissants que les modèles linéaires, mais dans notre cas, cela n'est pas le cas. Nous avons donc opté pour un second groupe de réseaux de neurones avec 3 couches cachées. L'entraînement fut plus long et a donné les tables de confusions suivantes :

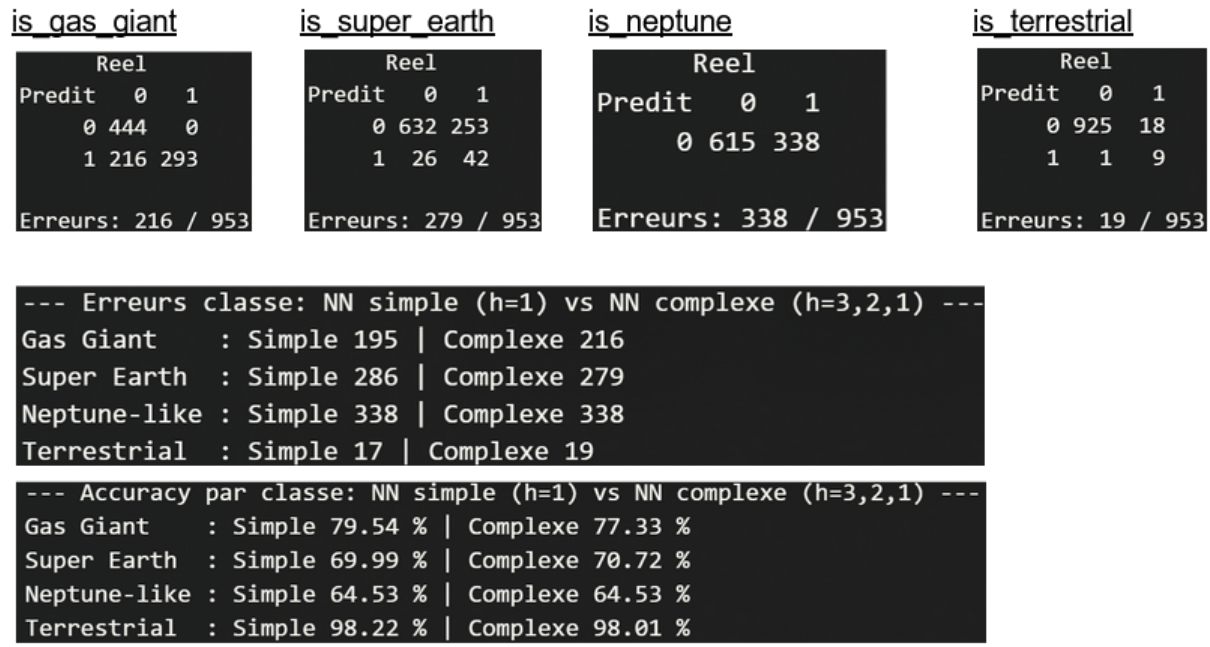

Figure 35 – Tables de confusions pour chaque classe (avec réseaux de neurones à 3 couches cachées)

Par rapport aux réseaux simples, ces résultats nous montrent une légère dégradation pour le cas *is_gas_giant* et de la stagnance pour *is_super_earth*, *is_neptune* et *is_terrestrial*.

Bilan : En conclusion, nous convergeons vers le fait que les réseaux à 1 couche généralisent mieux que les réseaux à 3 couches, à cause d'un sur-apprentissage possible des réseaux multi-couches dans le cas de notre dataset. Remarquons qu'entrainer des réseaux avec plus de couches, à plus grande dimension, aurait peut-être aidé à prédire avec plus de précision nos classes, malheureusement nous ne disposions pas du temps nécessaire à l'entraînement de telles complexités de modèle.

3.6. Évaluation des modèles

On peut maintenant rappeler les résultats de l'ensemble des méthodes utilisées à la classification. On mentionnera donc les quelques caractéristiques propres à leur configuration de leur algorithme et bien sur la précision de chacun.

Modèle	**Configuration**	**Accuracy**
Modèle Linéaire	Basique	$\approx 63{,}0\,\%$
Modèle Linéaire	Termes quadratiques	$\approx 85{,}0\,\%$
K-NN	$K = 9$	$\approx 87{,}0\,\%$
SVM	Noyau Linéaire ($C = 1$)	$\approx 55{,}4\,\%$
SVM	Noyau Radial ($C = 1$)	$\approx 72{,}0\,\%$
Réseaux de neurones	1 à 3 couches cachées	$\approx 77{,}65\,\%$ (modèle complexe)
Arbre de décision	Profondeur libre	$\approx 97{,}7\,\%$
Random Forest	500 arbres	$\approx 98{,}0\,\%$

Comme l'illustre le tableau, la croissance de nos scores d'Accuracy illustre à chaque fois une nouvelle méthode plus performante que la précédente :

— **Les premières limites linéaires :** Les faibles performances du SVM à noyau linéaire (55,4 %) et du modèle linéaire de base démontrent que les lois physiques séparant les types de planètes ne peuvent pas être résolues par de simples hyperplans.

— **Les méthodes utilisant la distance et leur point faible :** Les algorithmes qui utilisent la notion de distances entre points (K-NN à 87 %) ou sur la séparation avec des courbes (SVM Radial à 72 %) montrent de meilleures performances, mais lorsque l'on étudie leurs matrices de confusion, on observe un problème majeur de ces méthodes : on sacrifie les classes minoritaires par rapport aux majoritaires. Dans notre situation, les planètes "*Terrestrial*" sont du coup très mal prédite, à cause de leur faible présence dans le dataset.

— **Les réseaux de neurones (1 et 3 couches) :** Les réseaux de neurones (à 1 ou 3 couches) sont théoriquement capables d'approximer une fonction continue mais malgré cela, on pu montrer des performances décevantes, notamment pour prédire la classe *Neptune-like*. Ce résultat montre que malgré la puissance qu'est le réseau de neurones, on n'arrive pas toujours à trouver de bons résultats. Pour ce faire, on aurait eu besoin de plus de temps pour faire travailler des modèles complexes.

— **Les méthodes ensemblistes :** Les méthodes basées sur des règles de décision ont obtenu de bons résultats. Malgré que l'arbre de décision simple ait obtenu un très bon score (97,7 %), nous avons testé la méthode sans limite de profondeur, mais du coup l'exposant à un risque critique de sur-apprentissage. Le **Random Forest** s'impose avec le meilleur résultat pour la précision de prédiction. En effet, on atteint une précision de $\approx 98\,\%$ tout en parvenant à identifier la classe "*Terrestrial*" minoritaire, qui était encore là, difficile à prédire.

3.7. Discussion et limites

Notre objectif de classification a atteint d'excellents scores de prédiction avec 98 % d'accuracy en utilisant la Forêt Aléatoire. Le succès de notre partie classification s'explique par la nature même du jeu de données. En effet, les classes d'exoplanètes reposent sur des frontières physiques comme la masse et le rayon, que les algorithmes ensemblistes ont parfaitement réussi à capter.

Dans ce projet, on a eu l'occasion de trouver certaines limites à notre jeu de données :

— **Le biais d'observation :** En effet, le jeu de données souffre d'un déséquilibre massif. Comme montré au début de la partie de classification, les planètes "*Terrestrial*" représentent moins de 4 % du dataset, contre plus de 30 % pour les "*Gas Giant*". C'est bien sur normal du à la faible masse des planètes telluriques. Ce déséquilibre a fortement pénalisé les modèles basés sur les distances (K-NN, SVM), qui ont eu des difficultés à les prédire.

— **La frontière physique entre les classes :** La majorité des erreurs de notre meilleur modèle (Random Forest) se situe entre les "*Super Earths*" et les "*Neptune-like*". C'est finalement plutôt logique, car notre modèle se basait principalement sur les propriétés physiques comme la masse ou le rayon de la planète. Le souci se présente pour nos 2 classes, car les plus grosses planètes "*Super Earths*" possèdent une masse et un rayon proche de celle des plus petites planètes dite : "*Neptune-like*".

4. Conclusion générale

Au terme de ce projet, nous retenons des leçons techniques. Le travail sur un jeu de données conséquent, non nécessairement propre, prolonge les méthodes vues en travaux pratiques mais nous expose à la difficulté de mise en œuvre et d'interprétation, et nous pousse à explorer la littérature existante. Même si nous ne travaillons pas souvent dans un groupe si nombreux (6 personnes), l'organisation et la communication ont été fluides malgré la durée limitée.

Sources

— Bertin-Mahieux, T., Ellis, D. P. W., Whitman, B., et Lamere, P. (2011). *The Million Song Dataset*. Proceedings of the 12th International Society for Music Information Retrieval Conference (ISMIR).
http://millionsongdataset.com

— Dogra, P. (2020). *Year Prediction using Regression* ; implémentation et comparaison de plusieurs algorithmes de régression sur le dataset Year Prediction MSD.
https://github.com/prakhardogra921/Year-Prediction-using-Regression

— UCI Machine Learning Repository — Year Prediction MSD Dataset.
https://archive.ics.uci.edu/dataset/203/yearpredictionmsd

— NASA Exoplanets - A Comprehensive Catalogue of Exoplanets Discovered by NASA Missions
https://www.kaggle.com/datasets/adityamishraml/nasaexoplanets/data

— Documentation R `corrplot` :
https://cran.r-project.org/web/packages/corrplot/vignettes/corrplot-intro.html

— Documentation R `droplevels` :
https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/droplevels

— Documentation R `na.omit` :
https://www.rdocumentation.org/packages/photobiology/versions/0.14.0/topics/na.omit

— Documentation R `vis_miss` :
https://www.rdocumentation.org/packages/visdat/versions/0.6.0/topics/vis_miss

— Documentation R `SVM` :
https://www.rdocumentation.org/packages/e1071/versions/1.7-17/topics/svm