École Nationale Supérieure d'Informatique
pour l'Industrie et l'Entreprise

MOST
Projet 2024
Rapport

Prédiction des passagers du Titanic transportés vers une autre dimension

---

Élèves :
Colin Coërchon
Maël Daunas
Mohamed Koudiraty
Théo Taillefumier

Enseignants :
Blaise Hanczar
Jean-Christophe Janodet

4 avril 2024

Table des matières

1 Introduction

1.1 Un peu de contexte

Nous sommes en l'**an 2912** et le *Spaceship Titanic*, un paquebot interstellaire transportant 13 000 passagers vers de nouvelles exoplanètes, heurte une anomalie spatio-temporelle. L'incident rappelle évidemment le tragique destin de son homonyme d'il y a 1000 ans.

Suite au choc, **la moitié des passagers a été transportée dans une dimension alternative**. Des données corrompues du vaisseau spatial offrent un seul espoir : identifier les victimes de cette anomalie et les sauver.

Grâce à ces données, et en utilisant des techniques de science des données, nous avons pour objectif de percer ce mystère cosmique et contribuer à un sauvetage historique !

FIGURE 1 – Le *Spaceship Titanic*

1.2 L'objectif de notre travail

D'une façon plus concrète, notre travail s'articule autour d'un projet "Kaggle" qui consiste, à partir d'un jeu de données fourni, de prédire si tel ou tel passager sera transporté vers une autre dimension. Voici un aperçu des principales phases de notre démarche, décomposée comme suit :

- **Découverte et préparation des données :** Nous allons d'abord explorer le jeu de données pour comprendre sa structure et ses caractéristiques. Cette étape inclut le nettoyage des données, le traitement des valeurs manquantes, et la préparation des variables pour l'analyse.

- **Analyse exploratoire :** À travers une série d'analyses statistiques et de visualisations, nous identifierons les tendances, les anomalies et les relations clés au sein des données.
- **Modélisation prédictive :** Nous appliquerons ensuite divers algorithmes de machine learning pour prédire les chances de transport interdimensionnel des passagers. Cette phase impliquera la sélection du modèle, le réglage des paramètres, et l'évaluation de la performance.
- **Évaluation et optimisation :** Les modèles les plus performants seront finement ajustés et évalués pour maximiser leur précision et leur fiabilité.
- **Interprétation des résultats :** Enfin, nous interpréterons les résultats obtenus, en extrayant des insights pertinents qui pourraient éclairer les efforts de sauvetage.

Notre quête est ambitieuse : utiliser la science des données pour naviguer à travers cette catastrophe sans précédent et, espérons-le, **sauver des milliers de vies**.

2 Découverte et préparation des données

2.1 Exploration des données

Un ensemble de données (légèrement corrompu pour certaines) a ainsi été récupéré depuis le vaisseau, voici leur rapide description :

- `train.csv` - Ces données contiennent environ 2/3 ($\approx$ 8700) des passagers, à utiliser comme données d'entraînement.
  - ▷ `PassengerId` - **Un identifiant unique** pour chaque passager, se composant d'un code de groupe suivi d'un numéro au sein de ce groupe.
  - ▷ `HomePlanet` - La planète d'origine du passager, représentant souvent leur lieu de résidence permanent.
  - ▷ `CryoSleep` - Indique si le passager a opté pour une mise en sommeil cryogénique durant le voyage, les confinant à leur cabine.
  - ▷ `Cabin` - Le numéro de la cabine occupée par le passager, sous le format pont/numéro/côté, où le côté peut être Port (P) ou Starboard (S).
  - ▷ `Destination` - La planète vers laquelle le passager se dirige.
  - ▷ `Age` - L'âge du passager au moment du voyage.
  - ▷ `VIP` - Indique si le passager bénéficie de services VIP spéciaux pendant le voyage.
  - ▷ `RoomService`, `FoodCourt`, `ShoppingMall`, `Spa`, `VRDeck` - La somme dépensée par le passager dans les différentes commodités de luxe proposées par le *Spaceship Titanic*.

- ▷ `Name` - Les prénoms et noms du passager.
- ▷ `Transported` - Indique si le passager a été transporté vers une autre dimension, constituant notre **variable cible** à prédire.

- `test.csv` - Les données concernant le tiers restant ($\approx$ 4300) des passagers, destinées à être utilisées comme ensemble de test.

  Notre mission consiste donc à prédire l'état de `Transported` pour ces passagers.

2.2 Analyse des données

Nous venons donc d'avoir une description brute des différentes variables de nos données. Ensuite, il est primordial de télécharger les jeux de données, et de commencer à s'imprégner de lui en visualisant quelques exemples :

PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP
0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False
0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False
0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True
0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False
0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False

RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Name
0.0	0.0	0.0	0.0	0.0	Maham Ofracculy
109.0	9.0	25.0	549.0	44.0	Juanna Vines
43.0	3576.0	0.0	6715.0	49.0	Altark Susent
0.0	1283.0	371.0	3329.0	193.0	Solam Susent
303.0	70.0	151.0	565.0	2.0	Willy Santantines

Transported
False
True
False
False
True

TABLE 1 – Extrait des données du *Spaceship Titanic*

On a ainsi un beau premier exemple de nos données.

Maintenant, l'objectif est d'étudier nos données en s'intéressant précisément à chacune des variables. Commençons premièrement à vérifier que nous avons bien *environ* la moitié des personnes qui ont été transportées vers une autre dimension.

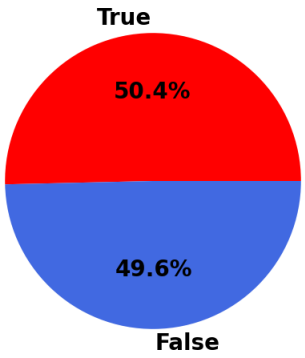


FIGURE 2 – Distribution des personnes transportées

Il y a donc bien une personne sur deux qui a été transportée suite au choc. Détaillons maintenant les autres variables.

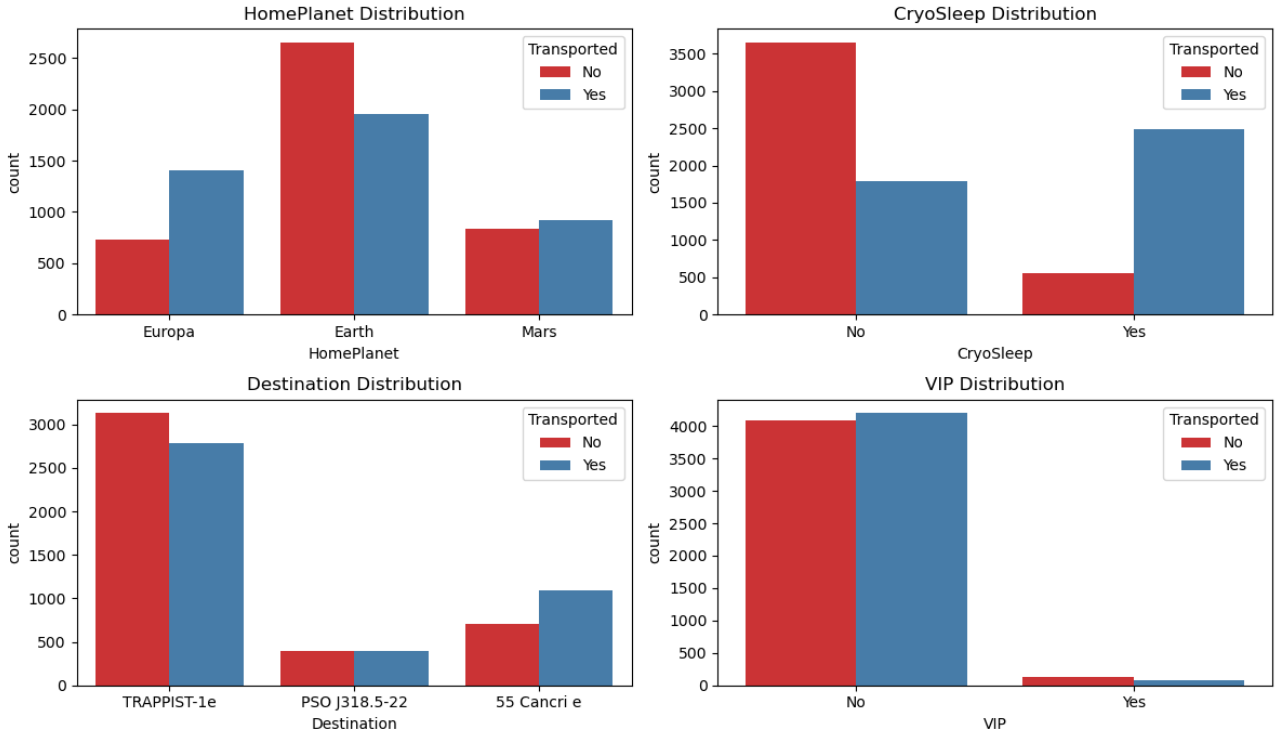


FIGURE 3 – Distribution des transportés d'après leur planète de départ et d'arrivée, et les options `VIP` et `CryoSleep`

Nous pouvons observer que la plupart des passagers viennent de la Terre, mais que les passagers de la Terre sont comparativement moins transportés, et que **les passagers d'Europe sont fortement transportés**.

Pour ce qui est des destinations, nous pouvons observer que la plupart des passagers sont transportés vers "Trappist-1e". Cette variable ne semble pas avec un réel impact sur les passagers transportés. D'un autre côté, **l'option `CryoSleep` semble influencer grandement la possibilité d'être transporté**.

Dans la fonctionnalité `VIP`, nous pouvons observer qu'une catégorie domine totalement une autre catégorie. Cela ne semble donc pas être une fonctionnalité utile car elle pourrait conduire à un surajustement dans notre modèle.

Concentrons-nous pour finir sur **l'âge** des passagers :

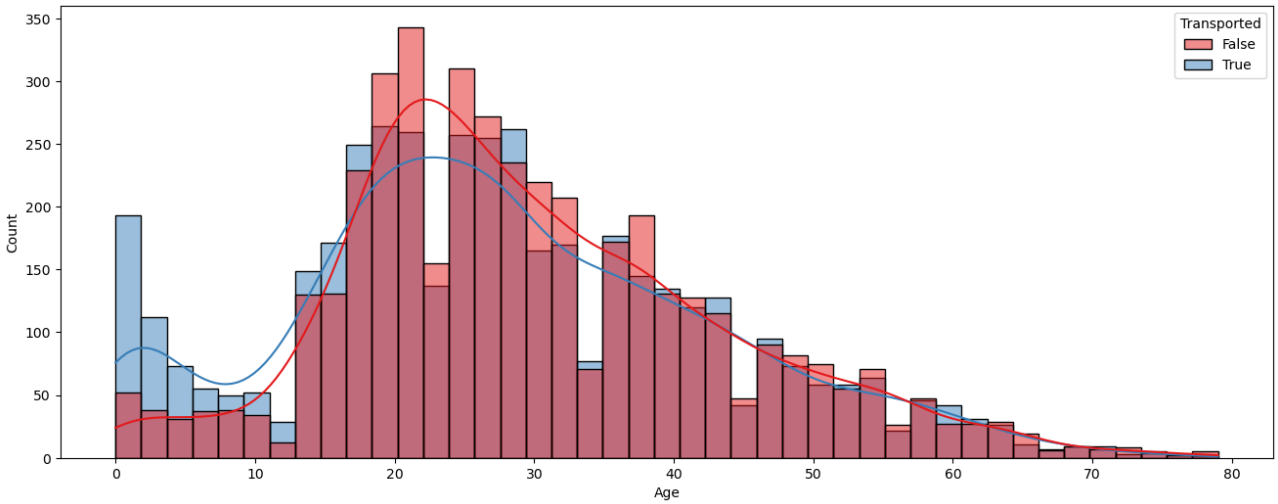


FIGURE 4 – Distribution en fonction de l'âge des passagers

- La plupart des passagers étaient âgés de 18 à 32 ans.
- Les personnages d'un âge de 0 à 18 ans sont hautement transportés, surtout les nouveau-nés
- Les passagers âgés de 18 à 32 ans sont comparativement moins transportés que dans les autres catégories d'âge.
- Au delà de 32 ans, l'âge ne semble pas influencer le résultat.

Nous avons également comparé les données relatives aux dépenses dans le vaisseau, et on peut globalement observer que :

- La plupart des passagers ne semblent pas dépenser d'argent.
- Étant donné que la plupart des dépenses sont nulles, les valeurs avec des dépenses plus élevées sont en quelque sorte des valeurs aberrantes dans nos données.
- D'une manière générale, les passagers ayant moins de dépenses sont plus susceptibles d'être transportés que les passagers ayant des dépenses élevées.

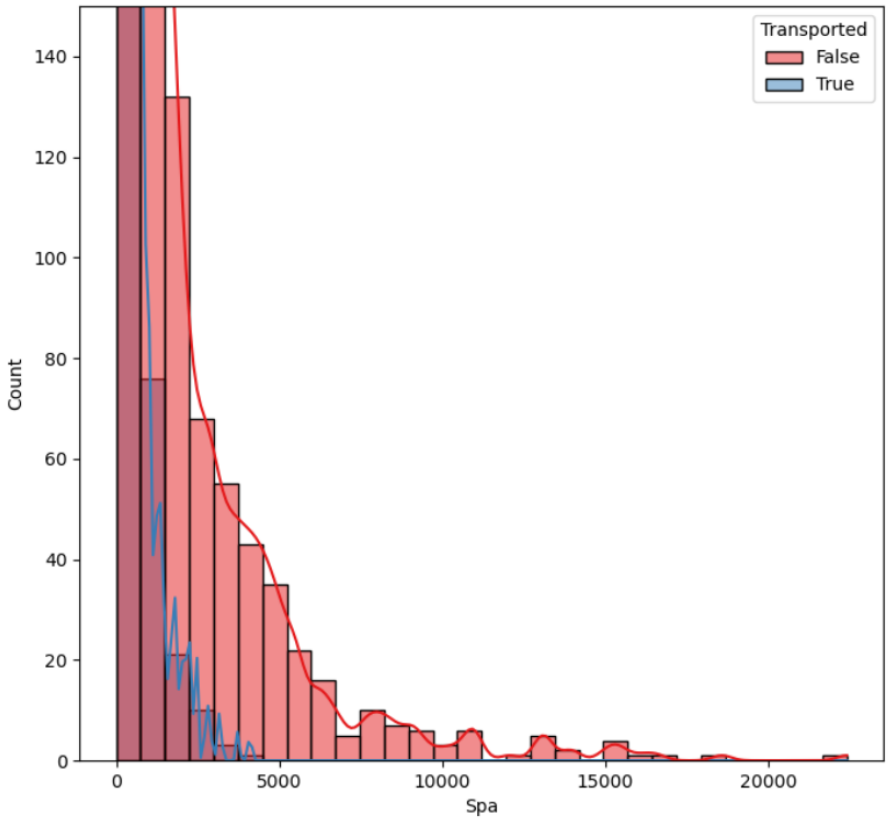


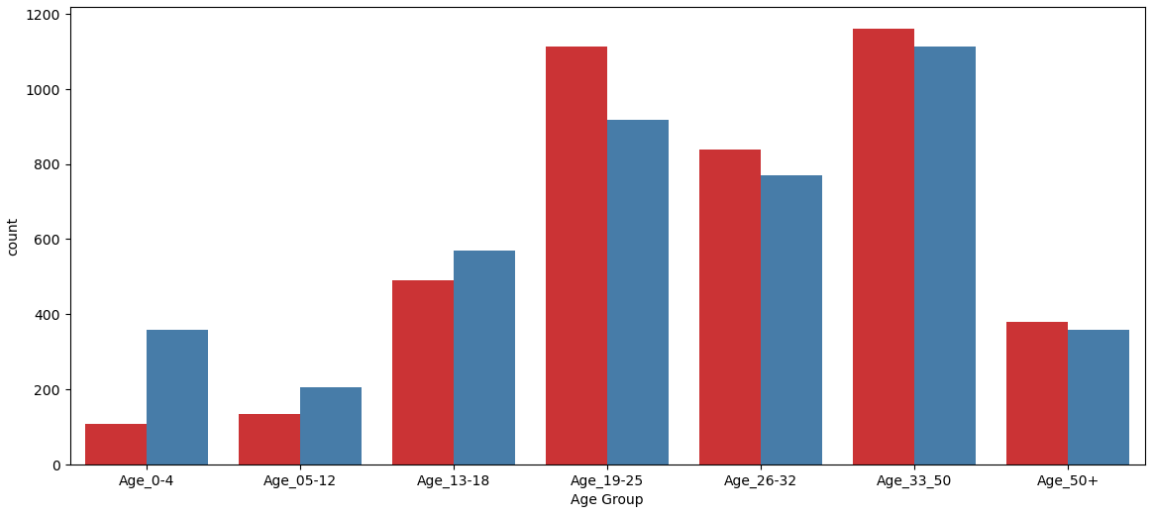
FIGURE 5 – Distribution en fonction des dépenses au spa

À noter que la distinction de la variable `Transported` est la plus forte sur les dépenses relatives au spa.

2.3 Transformations des données

Pour préparer notre jeu de données à l'analyse, nous avons appliqué une série de transformations visant à enrichir et à clarifier les informations disponibles.

- **Regroupement par âge :** Les passagers ont été divisés en catégories d'âge prédéfinies, facilitant ainsi l'analyse démographique.



n

FIGURE 6 – Distribution en fonction des différentes catégories d'âge créés

- **Traitement de l'identifiant du passager :** L'identifiant unique du passager a été décomposé pour identifier les groupes de voyage et le nombre de membres dans chaque groupe, permettant de distinguer les voyageurs solos des voyageurs en groupe.
- **Détail de la cabine :** Les informations de la cabine ont été séparées en pont, numéro et côté, tout en gérant les valeurs manquantes et en convertissant les numéros de cabine en données numériques.

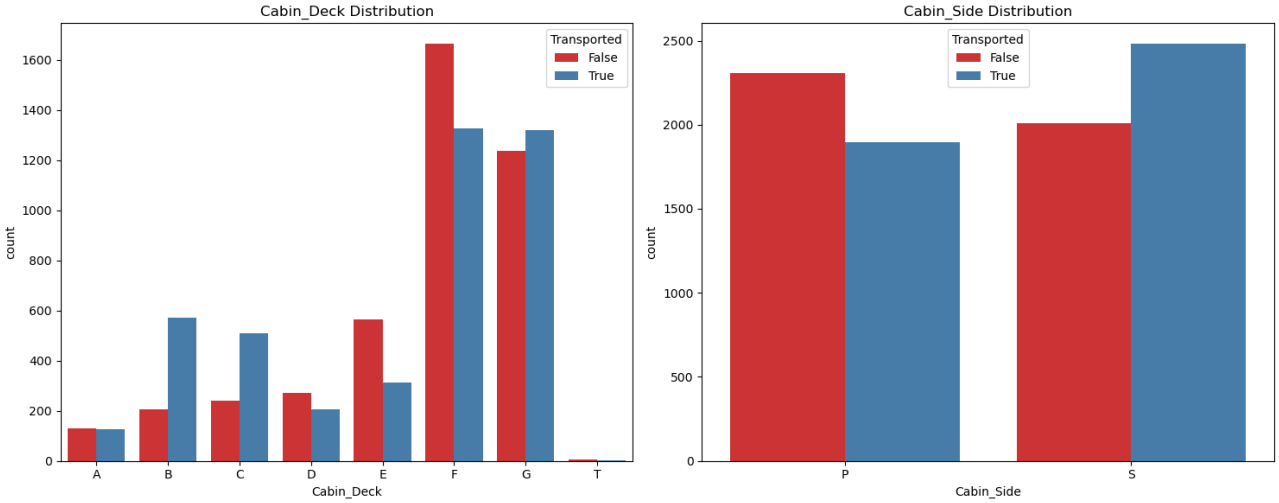


FIGURE 7 – Distribution en fonction du pont et du côté de la cabine

- **Régions de cabine :** Les numéros de cabine ont été utilisés pour créer des régions de cabine, offrant une autre dimension d'analyse spatiale sur le navire.

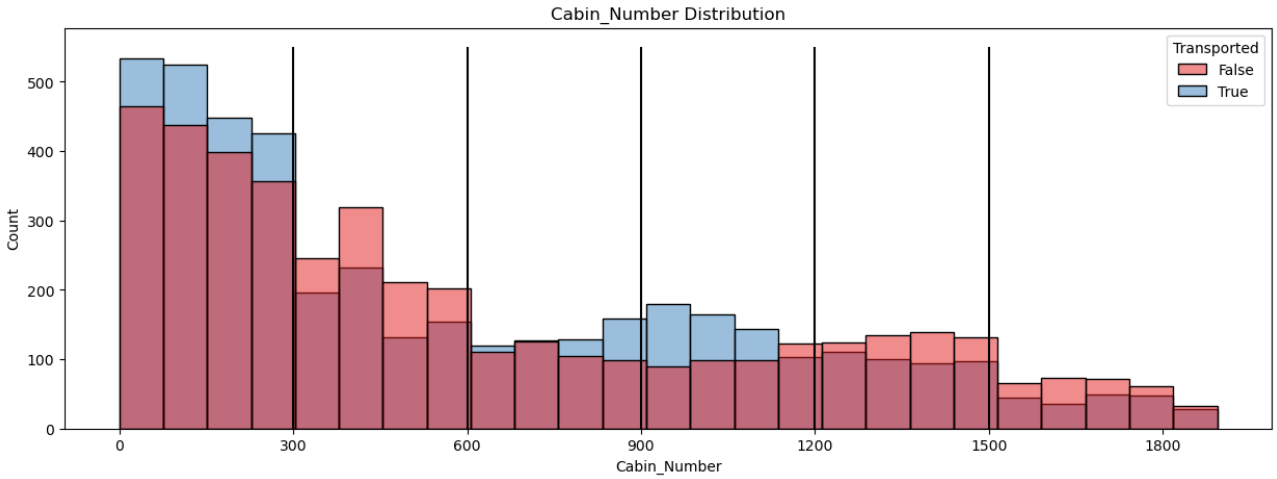


FIGURE 8 – Distribution en fonction de la région de la cabine

- **Dépenses des passagers :** Les montants dépensés dans divers services à bord ont été agrégés en une dépense totale, et une catégorisation des dépenses a été introduite pour distinguer les différents niveaux de consommation.

De plus, nous avons supprimé la variable `VIP` qui ne sera pas incluse dans notre modélisation, puisqu'au vu de l'analyse précédente, elle pourrait conduire à un surajustement dans notre modèle.

Ces étapes de transformation sont essentielles pour structurer notre analyse et **construire des modèles prédictifs solides**.

2.4 Traitement final : Encodage "One-Hot" et "Label Encoding"

Les variables catégorielles ont été traitées différemment selon leur nature pour mieux s'adapter aux exigences des modèles de machine learning :

- **Encodage One-Hot :** Les variables nominales `HomePlanet` et `Destination` ont été converties en utilisant l'encodage One-Hot, créant une nouvelle colonne binaire pour chaque catégorie unique.
- **Encodage des labels :** Un encodage de label a été appliqué à une série de variables catégorielles ordinales et binaires. Cela inclut les états de `CryoSleep`, si un passager voyage seul, divers attributs de la cabine, les groupes d'âge, les catégories de dépenses et d'autres colonnes binaires nouvellement identifiées.

L'encodage "One-Hot" a été préféré lorsque il n'y avait pas forcément d'ordre dans les valeurs des données. Il n'y a pas exemple aucun argument qui permettrait d'associer 1 à la Terre, et 2 à Mars ou inversement. Cet encodage est donc parfait pour ce cas là puisqu'il crée alors différentes variables binaires.

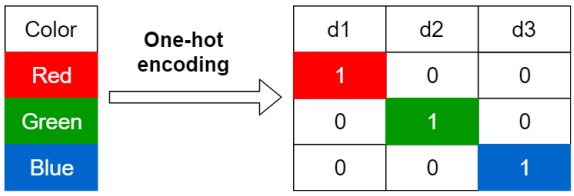


FIGURE 9 – Fonction de l'encodage "One-Hot"

Ces variables binaires nouvelles et non labellisées ont été ajoutées à la liste des variables à encoder, tandis que la variable cible `Transported` a été exclue de cette transformation.

En outre, une transformation logarithmique a été appliquée aux variables de dépenses à bord pour stabiliser la variance et normaliser la distribution des dépenses. La formule utilisée est $\log(1+x)$, assurant la gestion des valeurs zéro.

- **Transformation logarithmique :** Appliquée aux montants dépensés pour les services tels que le `RoomService`, le `FoodCourt`, le `ShoppingMall`, le `Spa`, et le `VRDeck`, ainsi qu'à la variable `Total Expenditure`.

Ces étapes de prétraitement sont essentielles pour aligner les données sur les besoins des algorithmes de machine learning, améliorant ainsi (*on l'espère*) l'efficacité et la précision des prédictions.

2.5 Remplissage de données manquantes

Petit résumé des **données manquantes** dans notre jeu de données initial :

Variable	**Nombre de valeurs manquantes**	**% de valeurs manquantes**
HomePlanet	201	2.31%
CryoSleep	217	2.50%
Cabin	199	2.29%
Destination	182	2.09%
Age	179	2.06%
VIP	203	2.34%
RoomService	181	2.08%
FoodCourt	183	2.11%
ShoppingMall	208	2.39%
Spa	183	2.11%
VRDeck	188	2.16%
Name	200	2.30%

TABLE 2 – Résumé des valeurs manquantes dans les données du *Spaceship Titanic* (sur un total de 8693 valeurs)

Il peut alors nous venir plusieurs idées pour **remplir les données manquantes**. Nous allons présenter les méthodes que nous avons utilisés.

2.5.1 Méthodes naïves

La méthode naïve appliquée pour le traitement des valeurs manquantes repose sur des heuristiques simples.

1. Pour les **variables catégorielles**, la valeur la plus fréquente est utilisée pour remplir les lacunes.

2. Pour les **variables numériques**, la médiane sert de remplaçant.

Cette approche, bien que rudimentaire, permet une imputation rapide et reste tout de même intéressante.

Elle s'écrit alors très simplement en Python :

```
cat_cols = train_df.select_dtypes(include=["object","bool"]).columns.tolist()
cat_cols.remove("Transported")
num_cols = train_df.select_dtypes(include=["int","float"]).columns.tolist()

def methode_naive(df):
    df[cat_cols] = SimpleImputer(strategy="most_frequent").fit_transform(df[cat_cols])
    df[num_cols] = SimpleImputer(strategy="median").fit_transform(df[num_cols])

methode_naive(train_df)
methode_naive(test_df)
```

Pourquoi ne pas supprimer les données corrompues ?

Une autre idée repose sur le fait de vouloir tout simplement **supprimer les lignes où il y a des données manquantes**. Seulement, on fait face à un réel problème puisqu'il est nécessaire de remplir les données manquantes pour notre ensemble de données `test_df`, dans l'objectif de pouvoir envoyer des résultats valides sur Kaggle.

La complétion des variables manquantes se fait alors naïvement, et induit donc un biais par rapport aux modèles entraînés sur `train_df` où les données manquantes avaient été ignorées.

Dans l'entièreté des modèles, le résultat est donc similaire, voire un peu moins performant.

2.5.2 Utilisation d'une autre heuristique

Après avoir essayer quelques modèles en gardant la méthode naïve d'imputation des données manquantes, il semblait important de réfléchir à une nouvelle heuristique pour compléter les données corrompues.

Après quelques discussions, nous sommes finalement partis sur cette méthode d'imputation :

- Pour les passagers voyageant **en groupe** ayant des données manquantes :
  - ▷ Pour les attributs numériques : remplir les valeurs manquantes avec la moyenne des attributs correspondants des autres membres du groupe.
  - ▷ Pour les attributs catégoriels : utiliser la valeur la plus fréquemment observée (mode) parmi les membres du groupe.
- Pour les passagers voyageant **seuls** avec des données manquantes :
  - ▷ Pour les attributs numériques : utiliser la moyenne des passagers qui se trouvent dans la même zone de cabine (`Cabin_Region`), sur le même pont (`Cabin_Deck`), et du même côté (`Cabin_Side`) du vaisseau.
  - ▷ Pour les attributs catégoriels : adopter la valeur la plus commune (mode) parmi les passagers partageant les mêmes caractéristiques de cabine.

3 Récapitulatif de nos données de travail

Nous avons réalisé une multitude de traitements dans nos données pour enfin pouvoir tester différents modèles d'apprentissage automatique.

En effet, nous venons de présenter les différentes méthodes que nous avons appliquées sur notre jeu de données : transformation de variables, encodage "One-Hot", suppression de variables non significatives, remplissage des données manquantes, etc...

En bref, voici donc le code final en Python que nous utilisons pour initialiser nos données de travail `X_train` et `X_test` (et ses variantes normalisées) :

```
Y = train_df["Transported"]
X = train_df.drop(columns=["Transported"])

X_scaled = StandardScaler().fit_transform(X)
test_df_scaled = StandardScaler().fit_transform(test_df)

X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.2,random_state=0)

X_train_scaled, X_test_scaled, Y_train_scaled, Y_test_scaled =
↪  train_test_split(X_scaled,Y,test_size=0.2,random_state=0)
```

On prend donc aléatoirement 20% des données d'entraînement pour en faire des données de test. Ces données seront extrêmement utiles pour entraîner nos modèles (notamment dans **l'optimisation des hyperparamètres**).

Voici donc, ci-contre, un rapide résumé de notre jeu de données transformé `X_train`. Nous avons donc finalement **28 variables** dans notre ensemble de données. Elles sont classées en catégories pour faciliter l'analyse et la compréhension :

- **État du passager :**
  - ▷ `CryoSleep` : Indique si le passager était en sommeil cryogénique.
  - ▷ `Travelling_Solo` : Si le passager voyage seul.
  - ▷ `No Spending` : Si le passager n'a effectué aucune dépense.
- **Dépenses à bord :**
  - ▷ `RoomService`, `FoodCourt`, `ShoppingMall`, `Spa`, `VRDeck` : Montants dépensés par le passager pour les services à bord.
  - ▷ `Total Expenditure` : La somme totale des dépenses du passager.
  - ▷ `Expenditure Category` : Catégorisation des dépenses du passager.
- **Informations sur le groupe :**

- ▷ `Group`, `Member` : Identifiant du groupe et numéro du membre.
- ▷ `Group_size` : Nombre de membres dans le groupe du passager.

- **Informations sur la cabine :**
  - ▷ `Cabin_Deck`, `Cabin_Side` : Emplacement de la cabine du passager.
  - ▷ `Cabin_Region1` à `Cabin_Region6` : Indicateurs des régions de cabine basées sur le numéro de la cabine.

- **Catégorisation :**
  - ▷ `Age Group` : Catégorie d'âge du passager.

- **Origine et Destination :** Encodage One-Hot des variables `HomePlanet` et `Destination`, subdivisées en :
  - ▷ `HomePlanet_Earth`, `HomePlanet_Europa`, `HomePlanet_Mars`
  - ▷ `Destination_55 Cancri e`, `Destination_PSO J318.5-22`, `Destination_TRAPPIST-1e`

Ces variables fournissent une vue d'ensemble complète des caractéristiques des passagers, allant de leur choix de sommeil cryogénique à leurs dépenses à bord, et seront cruciales pour prédire le comportement ou l'état futur des passagers à l'aide de modèles prédictifs.

4 Modèles d'apprentissage automatique

4.1 Modèles de régression linéaire

4.1.1 La régression logistique

Dans notre recherche de modèle prédictif, le premier modèle dont nous avons eu logiquement l'idée de tester est **la régression logistique**.

En effet, la régression logistique est souvent le point de départ dans l'analyse des relations entre une variable cible catégorielle et un ensemble de variables indépendantes. Dans notre étude, la variable cible Y est binaire, ce qui en fait un candidat idéal pour le modèle de régression logistique. Ce modèle est préféré car il est bien adapté à **la recherche de variables cibles binaires**.

La régression logistique repose sur l'hypothèse que les observations y_i sont des réalisations indépendantes de variables aléatoires Y_i, chacune suivant une loi de Bernoulli. Le lien entre la probabilité $p_\beta(x_i)$ de la variable aléatoire et les variables explicatives x_i est fait via la fonction logistique, aussi appelée sigmoïde, définie comme suit :

$$\operatorname{logit} p_\beta(x_i) = \ln\left(\frac{p_\beta(x_i)}{1 - p_\beta(x_i)}\right) = \beta^T x_i,$$

où β est **le vecteur des coefficients à estimer**. La fonction sigmoïde, qui est la fonction de transfert de ce modèle, est donnée par :

$$p_\beta(x_i) = \frac{e^{\beta^T x_i}}{1 + e^{\beta^T x_i}}.$$

Cette fonction traduit les combinaisons linéaires des variables explicatives en probabilités comprises entre 0 et 1.

Ce modèle reste simple à interpréter et à implémenter.

Voici son implémentation en Python :

```
regression_model = LogisticRegression(random_state=0)
regression_model.fit(X_train_scaled, Y_train)

Y_pred = regression_model.predict(X_test_scaled)

print("Accuracy: ", accuracy_score(Y_test, Y_pred))
```

On obtient des résultats de précision de l'ordre de 0.773 à 0.775 selon les essais.

Ce premier modèle fonctionne bien, mais il est certain que nous pouvons faire mieux !

4.1.2 Méthode de régression pénalisée

Après l'application de la régression logistique, nous avons exploré la **régression régularisée Elastic Net**, un modèle qui intègre à la fois **la pénalité L1 de Lasso**, qui peut réduire certains coefficients à zéro (offrant ainsi une sélection de variables), et **la pénalité L2 de Ridge**, qui réduit la variance des coefficients mais ne les annule jamais complètement.

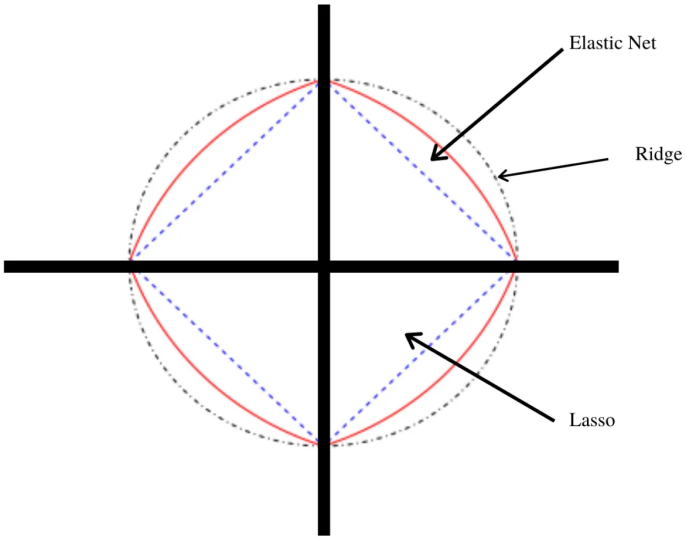


FIGURE 10 – Explication visuelle derrière "Elastic Net"

Cela permet de bénéficier d'un équilibre entre sélection de variables et régularisation, qui est contrôlé par deux hyperparamètres principaux :

- `C` : Inverse de la force de régularisation. Plus C est petit, plus la régularisation est forte.
- `l1_ratio` : Le ratio de mélange entre la régularisation Lasso et Ridge. `l1_ratio` $= 1$ est le Lasso, tandis que `l1_ratio` $= 0$ est le Ridge.

Les résultats de l'optimisation des hyperparamètres pour le modèle Elastic Net, effectuée par recherche sur grille avec validation croisée, sont résumés dans le tableau suivant :

Hyperparamètre	**Valeur Optimale**
Penalty	elasticnet
L1 Ratio	0.743
C	0.0336

TABLE 3 – Résultats de l'optimisation des hyperparamètres pour la régression Elastic Net

Et voici l'implémentation de notre modèle Elastic Net avec les meilleurs hyperparamètres :

```python
# Initialiser le modèle ElasticNet avec le l1_ratio et C trouvés
model_elasticnet = LogisticRegression(random_state=0, solver='saga',
↪  penalty='elasticnet', l1_ratio=l1_ratio_meilleur, C=C_meilleur, max_iter=5000)

# Entraînement du modèle ElasticNet avec les données d'entraînement normalisées
model_elasticnet.fit(X_train_scaled, Y_train)

# Prédire les valeurs sur l'ensemble de test prétraité
Y_test_pred = model_elasticnet.predict(test_df_scaled)
```

Avec une validation croisée de 5 plis pour un total de 2500 ajustements, les meilleurs hyperparamètres trouvés sont ceux indiqués dans le tableau, fournissant un meilleur score de validation croisée de 0.7778 et une précision (Accuracy) sur le jeu de test de 0.7764.

Le résultat est donc meilleur que celui de la simple régression logistique, mais nous sommes convaincu que des modèles plus complexes comme les arbres de décision ou les SVM peuvent arriver à de meilleurs résultats.

4.2 Les SVM

4.2.1 Description du modèle

Les machines à vecteurs de support (SVM) sont une méthode d'apprentissage supervisé pour la classification et la régression. Voici un résumé de leur fonctionnement et des hyperparamètres principaux :

- Les SVM cherchent l'hyperplan qui maximise la marge entre les classes dans l'espace des caractéristiques. Pour les données non linéairement séparables, elles utilisent le *kernel trick* pour projeter les données dans un espace de plus haute dimension où elles peuvent être séparées linéairement.
- **Hyperparamètres clés :**
  - ▷ **C** : Contrôle le compromis entre la classification correcte des points d'entraînement et la maximisation de la marge de l'hyperplan séparateur. Une valeur élevée vise à classer correctement tous les points d'entraînement, mais peut réduire la marge.
  - ▷ **Kernel** : Spécifie le type de transformation pour traiter les séparations non linéaires. Les options courantes incluent *linéaire*, *polynomial*, *gaussien*, et *sigmoïde*.
- Les SVM sont particulièrement efficaces dans des espaces de haute dimension et quand le nombre de dimensions dépasse le nombre d'échantillons. Cependant, leur performance peut diminuer pour de très grands ensembles de données et dépend fortement du choix du kernel et de la configuration des hyperparamètres.

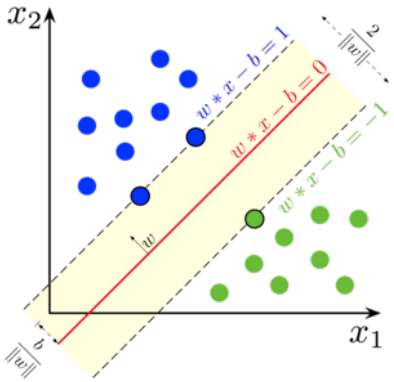


FIGURE 11 – Principe de fonctionnement des SVM

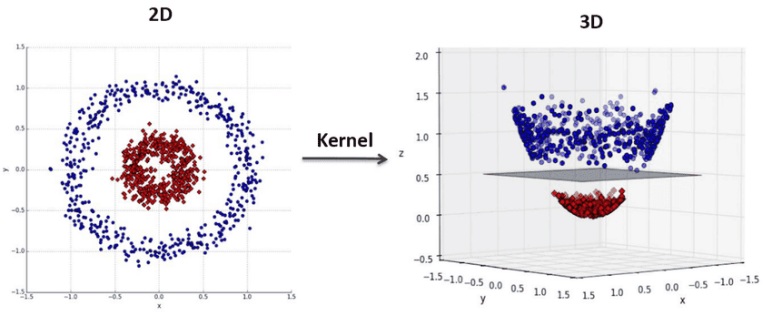


FIGURE 12 – Exemple de l'utilisation d'un noyau gaussien

4.2.2 Modélisation mathématique

La théorie mathématique des machines à vecteurs de support (SVM) repose sur la formulation d'un problème d'optimisation et l'utilisation d'une fonction de décision. Les concepts clés sont :

- **Problème d'optimisation :** Le modèle SVM résout un problème d'optimisation quadratique avec des contraintes. L'objectif est de minimiser une fonction qui dépend des poids θ, du biais θ_0, et des variables de relâchement ξ :

$$\min_{\theta,\theta_0,\xi} \quad \frac{1}{2}\theta^T\theta + C\sum_{i=1}^{n}\xi_i$$
$$\text{sous contraintes} \quad \begin{cases} y_i(\theta^T x_i + \theta_0) \geq 1 - \xi_i, \\ \xi_i \geq 0. \end{cases}$$

  Ici, C est un hyperparamètre qui contrôle **la pénalité des erreurs de classification**, et ξ_i sont **les variables de relâchement** qui mesurent le degré de marge de violation par les données.

- **Fonction de décision :** La décision est prise en fonction du signe du résultat de la fonction d'hypothèse :

$$\begin{cases} h(x) = \theta^T\phi(x) + \theta_0, \\ G(x) = \text{sign}(h(x)). \end{cases}$$

  La fonction $\phi(x)$ représente le mappage des données d'entrée dans un espace de caractéristiques, souvent réalisé par l'usage d'**une fonction noyau**. Le noyau permet de travailler dans un espace de haute dimension sans avoir à calculer explicitement les coordonnées dans cet espace, ce qui est particulièrement utile pour des données non linéairement séparables.

À noter que nous n'avons pas développé entièrement la théorie mathématique qui nous a été présenté en cours par M. Hanczar. Nous nous sommes contenté de la formulation naïve du problème.

Dans une analyse plus poussée, nous aurions abordé la transformation de notre problème d'optimisation primaire vers sa **formulation duale**. Cette dernière révèle un nouvel ensemble de variables, les coefficients de Lagrange α_i, qui prennent la place des variables de relâchement ξ_i dans la formulation originale.

$$\begin{cases} h_\theta(x_j) = \displaystyle\sum_{i=1}^{n} \alpha_i y_i K(x_j, x_i) + \theta_0, \\ G(x) = \text{sign}(h_\theta(x)) = \text{sign}\left(\displaystyle\sum_{i=1}^{n} \alpha_i y_i K(x_j, x_i) + \theta_0\right). \end{cases}$$

La formulation duale fait apparaître précisément le choix de la fonction noyau dans la fonction de décision, éclairant l'importance du noyau dans l'architecture globale du modèle SVM.

4.2.3 Résultats obtenus

Nous avons donc tenté d'exploiter les SVM dans notre projet d'apprentissage automatique Kaggle. Pour avoir les meilleurs résultats possibles, il était nécessaire d'essayer d'optimiser les hyperparamètres du SVM.

En premier lieu nous avons essayé d'utiliser **un noyau gaussien**.

Choix du noyau gaussien :

Nous cherchons à optimiser les valeurs des hyperparamètres C et $\gamma\,(= \frac{1}{2\sigma^2})$. L'hyperparamètre C contrôle le degré de régularisation (évitant le surajustement), γ définit l'ampleur de l'influence d'un seul point de données, et kernel spécifie le type de noyau utilisé, ici gaussien (RBF). Nous utilisons `RandomizedSearchCV` pour explorer efficacement l'espace des hyperparamètres, testant 100 combinaisons d'hyperparamètres différentes à travers 5 folds de validation croisée, pour un total de 500 ajustements de modèle.

Paramètre	Valeur Optimale
C	15.840958
gamma	0.0086557
kernel	rbf

TABLE 4 – Résultats de l'optimisation des hyperparamètres pour SVM avec Noyau Gaussien

Et voici l'implémentation de notre SVM avec les meilleurs hyperparamètres :

```
# Initialisation du modèle SVM
SVM_model = SVC(C= C_meilleur, gamma=gamma_meilleur, kernel='rbf')

# Entraînement du modèle SVM avec les données d'entraînement normalisées
SVM_model.fit(X_train_scaled, Y_train)

# Prédiction sur l'ensemble de test normalisé
Y_test_pred = SVM_model.predict(test_df_scaled)
```

Ces paramètres ont conduit à un meilleure score de validation croisée de 0.7947, et certains essais montaient **à plus de 0.80**, marquant ainsi **le meilleur résultat obtenu jusqu'à présent** pour ce modèle sur le jeu de données en question.

Choix des autres noyaux :

Nous avons également essayé cette optimisation pour d'autres noyaux connus, tels que **le noyau linéaire**, **le noyau polynomial** (avec différents degrés) et **le noyau sigmoïde**, mais cela n'a rien donné de mieux concluant.

Les précisions obtenues avec ces noyaux variaient entre 0.77 à 0.79, ce qui n'est pas significativement différent des résultats obtenus avec le noyau gaussien. Ces expériences renforcent notre décision d'utiliser le noyau gaussien pour notre modèle final, étant donné sa capacité à mieux capturer les complexités de nos données.

4.3 Les Arbres de Décision

4.3.1 Description du modèle

Les arbres de Décision forment un modèle de Machine Learning utilisé pour les tâches de classification. Il fonctionne **en divisant l'ensemble de données en sous-ensembles** plus petits basés sur des caractéristiques significatives, de manière récursive, jusqu'à ce que ces sous-ensembles soient les plus homogènes possible en termes de la variable cible. Ce processus de division est similaire à la prise de décision. Les divisions sont réalisées de manière à maximiser la pureté des sous-ensembles résultants.

Les avantages du `DecisionTreeClassifier` incluent sa facilité d'interprétation et de visualisation, sa capacité à gérer des données numériques et catégorielles, et le fait qu'il ne nécessite pas beaucoup de préparation des données. Cependant, ses inconvénients comprennent **sa tendance au surajustement**, particulièrement avec des arbres très profonds ou des ensembles de données complexes.

Pour évaluer la qualité des divisions pour catégoriser, deux mesures principales sont utilisées : l'indice de Gini et l'entropie. L'indice de Gini mesure l'impureté d'un sous-ensemble ; un indice de Gini de 0 indique une pureté parfaite, où tous les éléments du sous-ensemble appartiennent à la même classe. L'entropie, d'autre part, mesure le degré de désordre ou d'incertitude dans un ensemble. Une entropie de 0 signifie qu'il n'y a pas d'incertitude, c'est-à-dire que tous les éléments sont de la même classe. Lors de la construction d'un arbre de décision, ces mesures sont utilisées pour choisir les meilleures divisions en maximisant la réduction de l'impureté ou du désordre après chaque division.

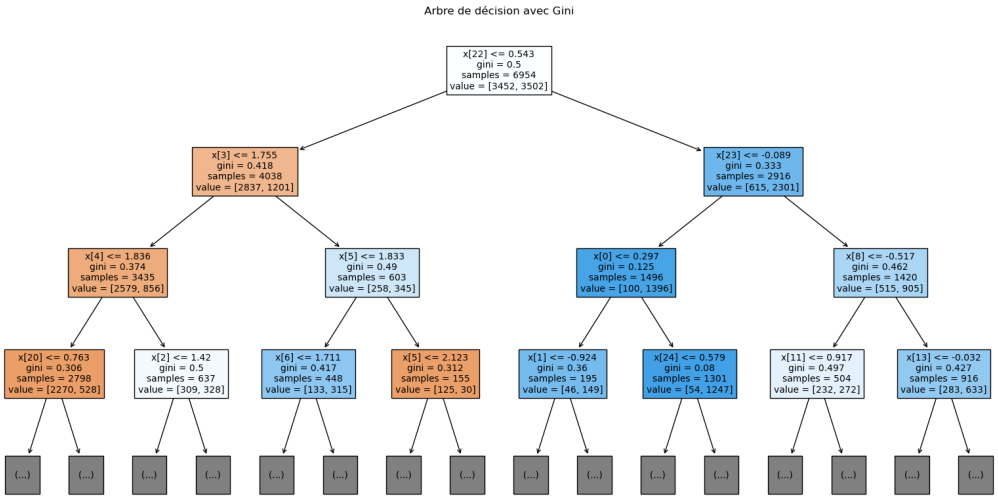


FIGURE 13 – Arbre de décision

4.3.2 Processus de classification

Premièrement, nous calculons la distribution des classes $p_N = (p_{N,0}, p_{N,1})$ pour la catégorie N :

$$p_{N,k} = \frac{\#\{i : x_i \in N, y_i = 0/1\}}{|N|}.$$

Ensuite, nous pouvons considérer une mesure d'impureté I telle que :

$$G(N) = G(p_N) = \sum_{k=0}^{1} p_{N,k}(1 - p_{N,k}) \qquad \text{(Indice de Gini)}$$

$$H(N) = H(p_N) = -\sum_{k=0}^{1} p_{N,k} \log_2(p_{N,k}) \qquad \text{(Indice d'Entropie)}$$

et pour $I = G$ ou $I = H$, nous considérons le gain d'information

$$IG(j,t) = I(N) - \frac{|N_L(j,t)|}{|N|} I(N_L(j,t)) - \frac{|N_R(j,t)|}{|N|} I(N_R(j,t)).$$

L'algorithme CART (Classification and Regression Trees) est une méthode de construction d'arbres de décision binaire qui peuvent servir à la fois pour la classification et la régression. Il crée un modèle qui prédit la valeur d'une variable cible en apprenant des règles de décision simples déduites des caractéristiques des données.

Dans le contexte de la classification, CART procède comme suit :

1. Pour chaque feuille N de l'arbre actuel, trouver la meilleure paire (caractéristique, seuil) (j,t) qui maximise $IG(j,t)$.
2. Créer les deux nouveaux enfants de la feuille.
3. Arrêter si un critère d'arrêt est atteint.
4. Sinon continuer.

Afin d'arrêter ce processus, on décide d'une profondeur maximale de l'arbre, ou un niveau suffisamment bas d'impureté.

4.3.3 Résultats obtenus

Dans le cadre de notre jeu de donnée, nous avons effectué 2 `RandomizedSearchCV`, un pour chaque indice, afin de déterminer les hyperparamètres qui donnent les meilleurs résultats pour l'indice Ginni ou d'Entropie.

Critère	Max Depth	Min Samples leaf	Min samples split	Meilleur Accurency
Gini	7	9	19	0.78861106
Entropy	9	16	11	0.78817847

TABLE 5 – Résultats de `RandomizedSearchCV` pour les critères "Gini" et "Entropy"

Et voici, par exemple, l'implémentation de notre arbre de décision avec les meilleurs hyperparamètres :

```
# Initialisation du modèle Arbres de décision
Arbre_model = DecisionTreeClassifier(criterion= 'gini', max_depth=7,
↪  min_samples_leaf=min_leaf_best, min_samples_split=min_split_best)

# Entraînement du modèle des arbres avec les données d'entraînement
Arbre_model.fit(X_train_scaled, Y_train)

# Prédiction sur l'ensemble de test
Y_test_pred = Arbre_model.predict(test_df_scaled)
```

Ainsi, le critère Gini pour ces meilleurs hyperparamètres, on obtient la meilleure précision pour ce modèle avec une accuracy de 0.78861106.

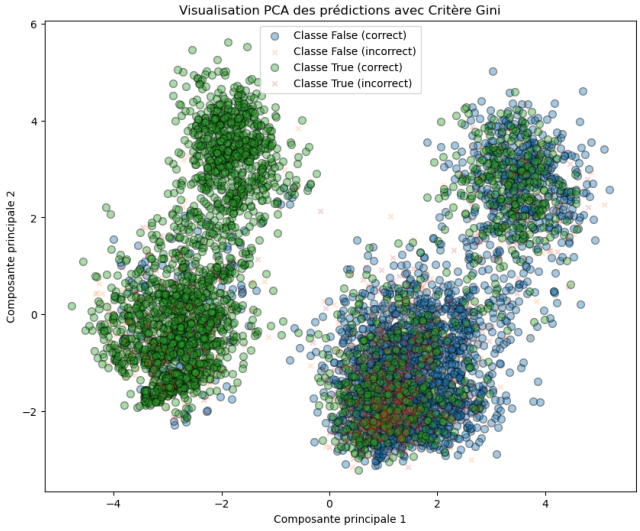


FIGURE 14 – Modélisation 2D par la méthode PCA de la catégorisation effectuée

4.4 Le Boosting

4.4.1 Description de la méthode

Le *boosting* est une technique d'ensemble qui vise à créer un modèle prédictif fort **à partir d'une série de modèles faibles**. L'idée principale est d'ajouter séquentiellement des modèles au modèle d'ensemble de manière à ce que les modèles suivants corrigent les erreurs des

modèles précédents. Parmi les algorithmes de boosting les plus connus, on trouve AdaBoost (Adaptive Boosting), Gradient Boosting, XGBoost (eXtreme Gradient Boosting), LightGBM (Light Gradient Boosting Machine), et CatBoost.

Dans le cadre de ce cours, nous avons étudié **AdaBoost**, une méthode de boosting très populaire qui combine plusieurs classificateurs faibles pour former un classificateur robuste. AdaBoost **ajuste les poids des observations** en fonction des prédictions de l'itération précédente afin de donner plus d'importance aux cas mal classés.

Nous avons choisi d'appliquer AdaBoost en commençant par un modèle prédictif initial basé sur **des arbres de décision**. Et pour optimiser notre modèle, les hyperparamètres suivants doivent être soigneusement ajustés :

- **n_estimators** : Le nombre d'arbres à inclure dans le modèle. Un nombre plus élevé peut améliorer la performance mais risque aussi de conduire à du surajustement.
- **learning_rate** : Contrôle la contribution de chaque arbre au modèle final. Des valeurs plus faibles peuvent nécessiter plus d'arbres mais peuvent améliorer la généralisation.
- **base_estimator** : Bien que AdaBoost puisse techniquement utiliser n'importe quel modèle comme estimateur de base, nous utilisons des arbres de décision. Les paramètres de ces arbres (comme la profondeur maximale) sont cruciaux pour la performance globale de l'ensemble.

En ajustant ces hyperparamètres, nous visons à construire un modèle AdaBoost qui améliore significativement la performance par rapport à notre modèle d'arbre de décision initial.

4.4.2 Fonctionnement de l'algorithme

Durant le cours avec M. Janodet, nous avons découvert cet algorithme de boosting *AdaBoost* ainsi que son pseudo-code. Voici donc son pseudo-code :

Algorithm 1 AdaBoost

Entrée : Ensemble de données d'entraînement (x_i, y_i), où $i = 1, \ldots, N$, $y_i \in \{-1, +1\}$
Initialiser les poids des observations $w_i = 1/N$ pour $i = 1, \ldots, N$
for $t = 1$ **à** nombre d'itérations **do**
    Entraîner un apprenant faible $h_t(x)$ en utilisant les poids w_i
    Calculer l'erreur de h_t : $\varepsilon_t = \sum_{i=1}^{N} w_i \cdot \mathbf{I}(y_i \neq h_t(x_i))$
    Calculer le poids de l'apprenant h_t : $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\varepsilon_t}\right)$
    Mettre à jour les poids des observations :
    $w_i \leftarrow w_i \cdot \exp(-\alpha_t \cdot y_i \cdot h_t(x_i))$ pour tout i
    Normaliser les poids w_i pour qu'ils somment à 1
end for
Modèle final : $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t \cdot h_t(x)\right)$

4.4.3 Résultats obtenus

Voici les résultats que nous avons obtenus en essayant d'optimiser les hyperparamètres dans le cadre de la méthode de *boosting* AdaBoost.

Paramètre	**Valeur Optimale**
base_estimator__max_depth	2
learning_rate	0.02915
n_estimators	871

TABLE 6 – Résultats de l'optimisation des hyperparamètres pour AdaBoost

Et voici l'implémentation de notre méthode AdaBoost avec les meilleurs hyperparamètres :

```
# Configuration de l'arbre de décision avec le meilleur max_depth trouvé
best_dt = DecisionTreeClassifier(max_depth=2, random_state=0)

# Initialisation du modèle AdaBoost avec les meilleurs hyperparamètres trouvés
ada_boost_model = AdaBoostClassifier(estimator=best_dt, n_estimators=871,
↪  learning_rate=0.02915053062825177, random_state=0)

# Entraînement du modèle AdaBoost avec les données d'entraînement normalisées
ada_boost_model.fit(X_train_scaled, Y_train)

# Prédiction sur l'ensemble de test normalisé
Y_test_pred = ada_boost_model.predict(test_df_scaled)
```

Avec ces paramètres, on arrive à des résultats extrêmement bons (**une accuracy proche de 0.805**). Les résultats sont aussi bons que ceux donnés par nos meilleurs SVM, alors qu'on partait ici du modèle des arbres de décision.

5 Conclusion

En conclusion, ce travail a soigneusement exploré l'amélioration des prédictions en transformant et affinant les données fournies. Nous avons décomposé des variables complexes telles que `Passenger_Id` et `Cabin`, appliqué des techniques d'encodage telles que le One Hot Encoding et le Label Encoding, et éliminé des variables jugées non significatives. Ces étapes étaient cruciales pour affiner notre modèle prédictif et *espérer* améliorer la précision globale de nos algorithmes d'apprentissage automatique.

Avec des données nettoyées et un total de **28 variables prédictrices**, nous avons procédé à l'évaluation de plusieurs modèles de machine learning. La liste des modèles testés comprend la régression logistique, Elastic Net, les machines à vecteurs de support (SVM), les arbres de décision et le boosting avec AdaBoost. Chaque modèle a été soigneusement sélectionné pour sa pertinence et son potentiel dans le contexte de notre problématique.

Et les résultats Kaggle ?

Les soumissions à la compétition Kaggle ont montré une amélioration progressive des résultats. Nous avons commencé avec des scores avoisinant les 0.78, qui se sont accrus à 0.79 grâce à Elastic Net.

L'application de SVM a permis une percée, atteignant **un score de 0.80360**, tandis que les arbres de décision plafonnaient à 0.78. Après divers ajustements, notre score maximal avec AdaBoost a été de 0.80056. Bien que nous n'ayons pas surpassé toutes les équipes de l'ENSIIE et du M1 informatique, nous sommes fiers de **notre classement final de 589ème sur 2667 participants** (en sachant que le *vrai* premier a une précision de 0.82207).

Ce projet a été une démonstration rigoureuse de l'application de l'analyse de données et de l'apprentissage automatique dans un contexte à la fois ludique et compétitif.

Meilleur Score Kaggle	**Classement Kaggle**
0.80360	$589^{\text{ème}}$ sur 2667

TABLE 7 – Meilleur score et classement dans la compétition Kaggle