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

MOST - Machine Learning
Projet Kaggle - Rapport

Élèves : **Mehdi SAIDI**
Théo LE MOAL
Alexandre WILK
Antoine TIREAU

Sommaire

1 Introduction

Dans un futur lointain, en l'an 2912, les voyages interstellaires sont devenus monnaie courante et les paquebots spatiaux transportent des milliers de passagers d'un système solaire à un autre. Le Spaceship Titanic, un paquebot interstellaire, avait pour mission de transporter près de 13 000 passagers. Malheureusement, lors de son voyage, le Spaceship Titanic a heurté une anomalie spatio-temporelle , provoquant un événement catastrophique : des passagers ont été transportés dans une dimension alternative.

L'objectif de ce projet est de créer un modèle prédictif capable d'identifier quels passagers ont été transportés vers cette dimension alternative en se basant sur les dossiers personnels récupérés du système informatique endommagé du vaisseau.

Les données à notre disposition comprennent des informations sur les passagers, telles que leur identifiant, la planète d'origine, l'âge ou la destination.

Pour mener à bien ce projet, nous suivrons une méthodologie rigoureuse, incluant l'analyse exploratoire des données, le prétraitement des données, une sélection des caractéristiques, l'entraînement et l'évaluation de différents modèles de Machine Learning, l'optimisation des hyperparamètres et, enfin, la réalisation de prédictions sur les données de test.

En combinant analyse de données et Machine Learning , notre objectif est d'élaborer un modèle prédictif fiable et utile pour répondre à cette problématique. Ce rapport détaillera l'ensemble des étapes suivies, les résultats obtenus et les conclusions tirées, en vue de fournir un aperçu complet de notre travail sur ce projet.

Ce schéma représente la trame que l'on va suivre tout du long du projet : on commencera par expliquer le formatage effectué sur le jeu de données, ainsi que les nouvelles variables que nous avons créés. Ensuite, on parlera des différentes méthodes abordées pour remplir les valeurs manquantes. On présentera ensuite les modèles utilisés, pour terminer par l'optimisation des hyperparamètres.

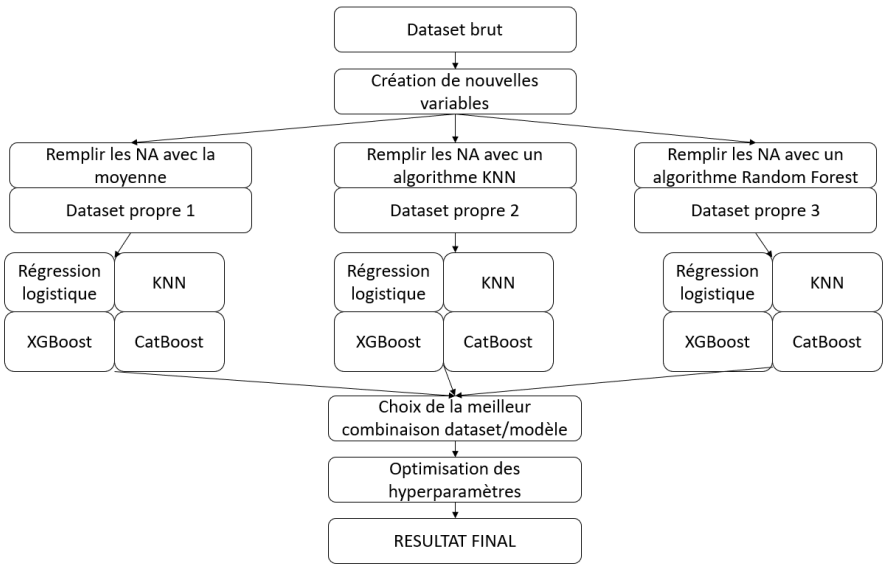

FIGURE 1 – Schéma de la trame du projet

2 Développement

2.1 Présentation et travail sur le dataset

2.1.1 Présentation du dataset

Le dataset est composé d'une variable cible ' Transported ' ainsi que de 13 variables explicatives. Ces variables explicatives sont les suivantes :

- PassengerId - Un identifiant unique pour chaque passager. Chaque identifiant prend la forme ggg-pp où gggg indique le groupe auquel appartient le passager et pp est leur numéro au sein du groupe.
- PlanèteDomicile - La planète de départ du passager, généralement leur planète de résidence permanente.
- CryoSommeil - Indique si le passager a choisi d'être placé en animation suspendue pendant la durée du voyage. Les passagers en cryosommeil sont confinés dans leurs cabines.
- Cabine - Le numéro de la cabine où séjourne le passager. Prend la forme pont/num/côté, où côté peut-être soit P pour Tribord (Port) ou S pour Bâbord (Starboard).
- Destination - La planète où le passager débarquera
- Âge - L'âge du passager.
- VIP - Indique si le passager a payé pour un service VIP spécial pendant le voyage.
- ServiceChambre, RestaurationRapide, CentreCommercial, Spa, PontRV - Montant facturé par le passager pour chacune des nombreuses installations de luxe du Spaceship Titanic.
- Nom - Les prénoms et noms de famille du passager.
- Transporté - Indique si le passager a été transporté dans une autre dimension. Il s'agit de la variable cible, la colonne que l'on essaye de prédire.

Nous avons étudié ce dataset pour en déduire les axes de travaux principaux, tels que les variables importantes à considérer en premier pour prédire au mieux la variable cible et améliorer l'efficacité de nos modèles. Pour cela, nous avons principalement étudié la correlation entre la variable cible et chacune des variables explicatives, notamment à travers la répartition de leurs valeurs selon si la cible est transportée ou non.

2.1.2 Traitement/formatage des variables

Dans le cadre de notre projet, nous avons effectué un travail approfondi sur le dataset afin de préparer les données pour les modèles de machine learning et d'optimiser leur performance. Ce travail a consisté en plusieurs étapes toutes essentielles :

1. Suppression des variables inutiles

```
count    2217.0
mean        4.0
std         2.0
min         1.0
25%         2.0
50%         3.0
75%         5.0
max        18.0
Name: Surname, dtype: float64
```

FIGURE 2 – Distribution des occurences de Surname

Nous analysons que vingt passagers dans les données d'entraînement ont des noms en double. L'analyse des caractéristiques supplémentaires montre que les passagers avec des noms en double ont d'autres différences significatives indiquant qu'ils semblent être des personnes différentes qui partagent un nom. De plus, elle nous a initialement parue exploitable en différenciant hommes et femmes puis en essayant de trouver une corrélation entre les personnes transportées et leur sexe. Cependant, tous les prénoms étant des prénoms "futuristes", ils ne sont pas associables à un sexe donné et donc inexploitables.C'est pourquoi nous avons décidé de supprimer cette variable 'Nom'
La deuxième variable enlevée du dataset est celle du PassengerID. La seule information utile dans l'ID d'une personne est de connaître les personnes présentes dans sa cabine, puis pouvoir faire l'étude du sort des différentes personnes au sein d'une même cabine pour trouver une possible corrélation. Cette étude a pu être fait avec la variable cabine. Le résultat est ci-dessous :

```
0.000000    48.277439
1.000000    42.439024
0.500000     4.893293
0.666667     1.448171
0.333333     1.173780
0.750000     0.548780
0.600000     0.228659
0.250000     0.213415
0.800000     0.167683
0.833333     0.152439
0.400000     0.121951
```

FIGURE 3 – Proportions dans Cabin

On voit donc que 42% des cabines ont eu tous leurs membres transportés, et 48% ont vu tous leurs membres non transportés. Il y a donc 9 chances sur 10, en connaissant l'état d'une personne, qu'on puisse en déduire l'état des autres membres de la cabine, ce qui est très intéressant.

2. Numérisation du dataset

Afin que notre dataset soit exploitable par les algorithmes de machine learning, nous avons numérisé toutes les variables booléennes, c'est-à-dire substituer par 0 les False et 1 les True.

CryoSleep	CryoSleep
False	0.0
False	0.0
False	0.0
False	0.0
False	0.0
False	0.0
False	0.0
True	0.0
False	1.0
True	0.0

FIGURE 4 – Transformation de CryoSleep

Après avoir traité les données de Cryosleep, nous avons réalisé un graphique donnant le poucentage de passagers transporté en fonction de son etat de cryosleep.

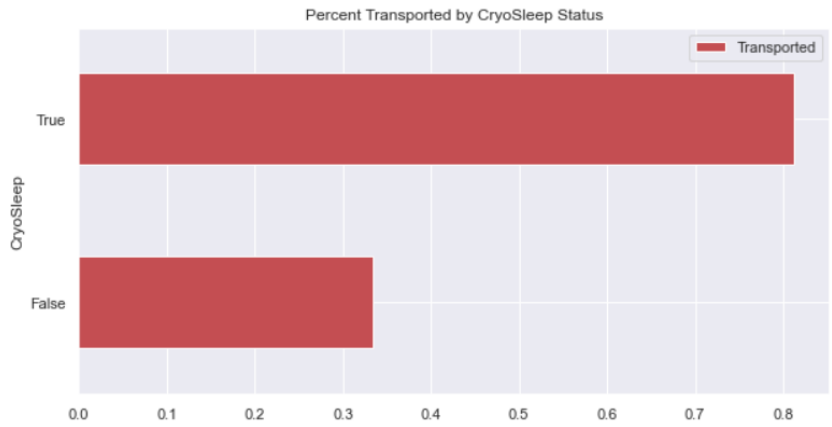

FIGURE 5 – Pourcentage de Transported par CryoSLeep

Nous analysons qu'environ un tiers des passagers ont choisi d'être placés dans CryoSleep pendant le voyage et que ceux-ci semblent avoir plus de 80 pourcents de chances d'être transportés dans une autre dimension, ce qui en fait probablement l'une de nos variables prédictives les plus importantes.

3. One-Hot encoding

Le One-Hot Encoding est une technique de prétraitement de données utilisée pour convertir des variables catégorielles en vecteurs numériques binaires afin de faciliter l'utilisation de ces données dans les algorithmes de machine learning. L'intérêt du One-Hot Encoding est qu'il permet de convertir des données catégorielles en une forme numérique qui peut être facilement traitée par les algorithmes de machine learning. Les algorithmes de machine learning ne peuvent traiter que des données

numériques, il est donc important de convertir les variables catégorielles en une forme numérique appropriée pour permettre leur utilisation dans les algorithmes de machine learning. De plus, K catégories dans une colonne au départ impliquent la création de K-1 colonnes afin d'éviter la redondance de l'information (s'il y a un 0 sur les K-1 colonnes créées, cela signifie donc que l'observation est de la K-ième catégorie, pas besoin de rajouter une autre colonne). Ici, nous avons donc One-Hot encodé Home Planet et Destination.

Nous obtenons donc les graphiques suivants :

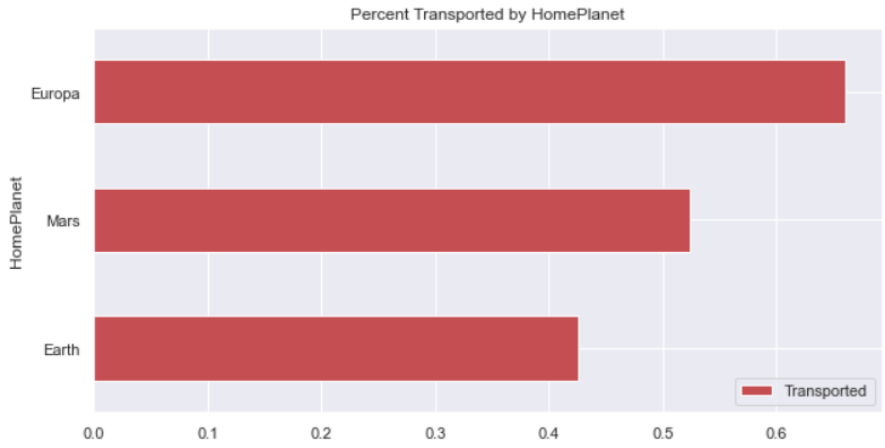

FIGURE 6 – Pourcentage de Transported par HomePlanet

Nous analysons que la Terre est la HomePlanet la plus fréquente pour les passagers. Les passagers d'Europe ont près de 15 pourcents de chances en plus d'être transportés par rapport aux passagers de la Terre qui ont environ 8 pourcents de chances en moins d'être transportés.

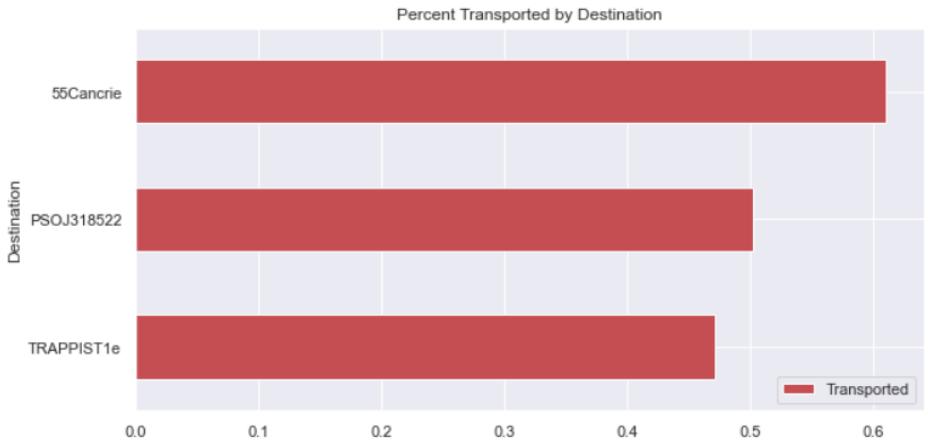

FIGURE 7 – Pourcentage de Transported par Destination

Trappist-1e est de loin la destination de passagers la plus courante. Les passagers voyageant vers le 55 Cancri e ont plus de 60 pourcents de chances d'être transportés, tandis que les passagers voyageant vers Trappist-1e ont un peu moins de 50 pourcents de chances d'être transportés.

2.1.3 Analyse et Création de données

1. Création de données

Nous avons créé la donnée PassengerGroup à partir de la fonctionnalité PassengerId, qui peut fournir plus d'informations sur des groupes entiers qui ont été transportés. PassengerGroup peut également servir de nœud autonome ou de propriété de passager lorsque nous avons créé notre modèle de graphique.
En utilisant Passenger-Group, nous avons compté le nombre de passagers dans chaque groupe et l'avons ajouté en tant que fonctionnalité. Dans notre modèle graphique, il peut s'agir d'une propriété pour les passagers ou les groupes

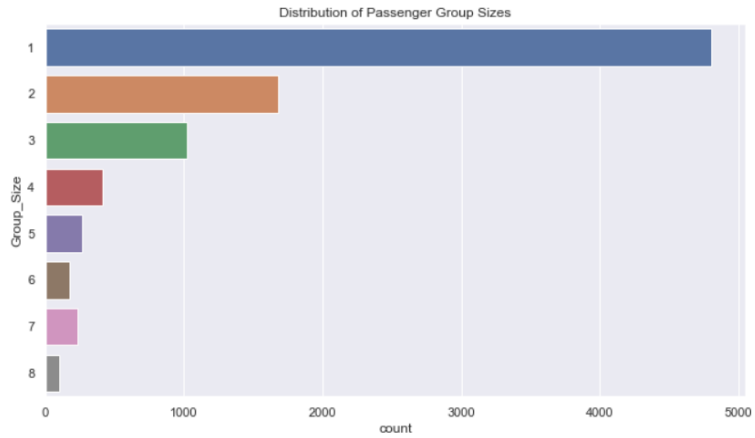

FIGURE 8 – Distribution des Passenger Group Sizes

Il y a environ 6200 groupes de passagers et plus de la moitié des passagers voyagent seuls. De plus, eu de groupes ont plus de cinq passagers, le plus grand comprenant huit passagers.
Nous avons jugé utile de créer la donnée Total_Spend représentant la somme total dépensé par les passagers et voila ce que nous en avons obtenus :

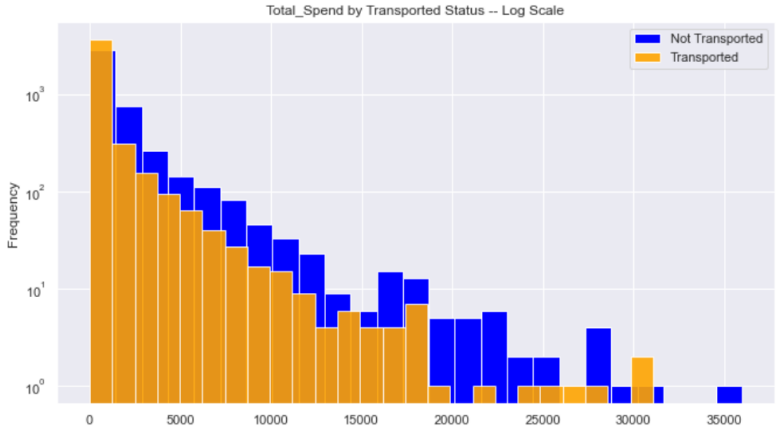

FIGURE 9 – Total-Spend par Transported – Log Scale

Les passagers avec des dépenses nulles (ou minimes) semblent être transportés à un taux plus élevé que les passagers qui ont dépensé de l'argent à bord. Cependant, cette caractéristique peut être corrélée avec les passagers en statut CryoSleep (et

donc qui n'ont pas dépensé), qui avaient également un taux de transport beaucoup plus élevé que le reste des passagers.

2. Analyse de certaines données

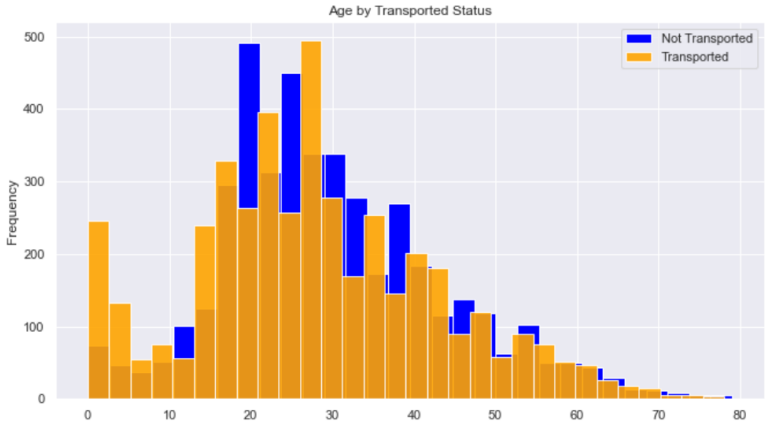

FIGURE 10 – Age par Transported

Les passagers de moins de 20 ans semblent avoir un taux de transport plus élevé. Les passagers âgés d'environ 20 à 40 ans semblent avoir une probabilité plus faible d'être transportés. À partir de 40 ans les pourcentages apparaissent à peu près homogènes entre Transporté et Non.

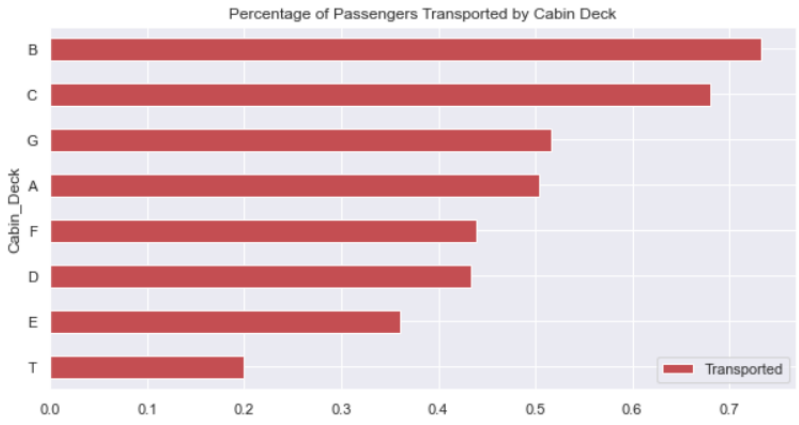

FIGURE 11 – Pourcentage de Transported par CabinDeck

Les passagers des ponts B et C semblent avoir une chance considérablement plus élevée d'être transportés, tandis que les passagers des ponts E et du T peu peuplé ont des changements beaucoup plus faibles d'être transportés.

2.2 Gestion des valeurs manquantes

La gestion des données manquantes est une étape cruciale pour l'entraînement d'un algorithme de machine learning efficace. En effet, les données manquantes peuvent perturber le modèle et affecter sa capacité à effectuer des prévisions précises. Dans cette partie, nous présenterons les différentes manières dont nous avons traités ces valeurs, et

l'impact de celles-ci sur la performance de nos modèles. La manière la plus naive de traiter ces valeurs (c'est-à-dire simplement les supprimer) est impossible ici, puisque Kaggle requiert une prédiction pour chacun des 4277 passagers du set de test. C'est pourquoi nous avons concentré nos efforts sur deux autres méthodes d'imputation : par moyenne et par regression.

2.2.1 Imputation par moyenne

Cette méthode consiste à remplacer les valeurs manquantes d'une feature numérique par par la moyenne des observations. Dans le cas de notre dataset, seules les features Age, et les sommes dépensées sont des valeurs numériques, les autres features étant qualitatives (VIP, Destination, HomePlanet...). Pour gérer les valeurs manquantes des features qualitatives, nous avons donc fait le choix dans un premier temps de les remplacer par leur mode (c'est-à-dire leur valeur la plus fréquante). Pour les features numériques, nous avons aussi testé nos modèles en remplaçant par les valeurs médiannes. Les performances au final (présentées à la fin de ce paragraphe) ne diffèrent que de très peu, ce qui semble assez suprenant lorsque l'on s'intéresse à la distribution des sommes dépensées. En effet, une majorité de gens ne dépensent pas ou peu, mais une minorité dépensent beaucoup ce qui tire la moyenne vers le haut mais n'est pas représentatif de ce que les gens dépensent réellement "en moyenne" (cela se traduit par une médiane à 0, mais une moyenne à plus de 400 pour la feature FoodCourt par exemple).

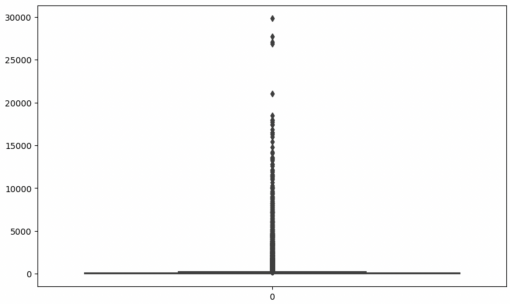

FIGURE 12 – BoxPlot de la feature FoodCourt

En revanche, cela est moins surprenant lorsque l'on s'intéresse à la distribution des âges :

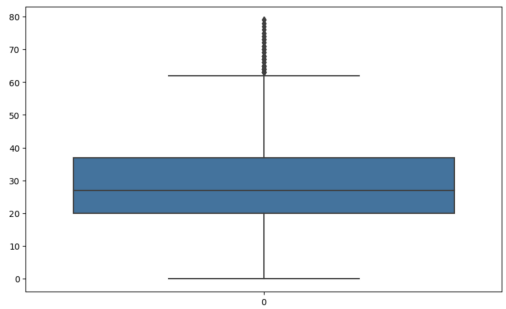

FIGURE 13 – BoxPlot de la feature Age

2.2.2 Imputation par régression

1. Imputation avec la méthode des *k plus proches voisins*.

L'algorithme des k plus proches voisins (ou KNN, pour "K-Nearest Neighbors" en anglais) est une méthode d'apprentissage supervisé utilisée pour la classification et la régression. Le principe de base de l'algorithme est de déterminer la classe ou la valeur d'une nouvelle observation en examinant les k échantillons les plus proches de cette observation dans l'ensemble de données d'entraînement. Pour ce faire, l'algorithme calcule la distance (ici, euclidienne) entre l'observation considérée et tous les échantillons de l'ensemble de données d'entraînement, puis sélectionne les k échantillons les plus proches en termes de distance. Une fois les k échantillons les plus proches sélectionnés, la classe ou la valeur de la nouvelle observation est déterminée en fonction de la classe ou de la valeur majoritaire des k échantillons les plus proches. Dans le cas d'une classification, la classe majoritaire est attribuée à la nouvelle observation, tandis que dans le cas d'une régression, la valeur moyenne des k échantillons les plus proches est utilisée pour prédire la valeur de la nouvelle observation.
Dans notre cas, les colonnes qui contiennent des données manquantes sont traitées comme la variable cible tandis que les autres colonnes sont utilisées comme variables d'entrée. Les valeurs manquantes dans la colonne cible sont prédites en utilisant les valeurs des autres colonnes.

Puisque nos features sont à la fois catégorielles et numériques, nous utilisons l'algorithme KNN en mode régression sur l'ensemble du dataset, puis nous "recatégorisons" les variables catégorielles transformées en variables numériques par l'algorithme. En effet, puisque nos variables catégorielles sont représentées par des 0 ou des 1, il suffit de calculer si la valeur manquante calculée par l'algorithme est plus proche de 0 ou de 1 pour remplacer la valeur "par regression" par celle "par classification" de l'algorithme (en supposant que l'on a un nombre de voisin impair).
Dans notre cas, après avoir split le dataset de train, nous remarquons que les performances de nos modèles sont meilleures si nous utilisons un grand nombre (>20) de voisins pour l'imputation. La distance choisie étant euclidienne, nous avons d'abord dû centrer et réduire le dataset, afin que chaque composante ait la même importance dans le calcul de la distance. Cette méthode, tout comme la suivante peu introduire un biais, en particulier si les données manquantes sont fréquentes. Dans nos dataset, pour chaque colonne, il y a environ 100 données manquantes, ce qui reste relativement peu fréquent et justifie l'utilisation de ces méthodes d'imputation.

2. Imputation avec la méthode *Random Forest*

   L'algorithme Random Forest est une méthode d'apprentissage supervisé utilisée pour la classification, la régression et d'autres tâches de prédiction. Il est basé sur l'ensemble d'arbres de décision aléatoires. Voici les étapes principales de l'algorithme Random Forest :

   (a) Collecte des données d'entrée : Les données d'entrée doivent être collectées sous forme de tableau avec des colonnes représentant les caractéristiques et des lignes représentant les exemples.

   (b) Sélection aléatoire d'un échantillon de données :L'algorithme sélectionne aléatoirement un échantillon de données à partir de l'ensemble de données d'entrée.

   (c) Construction d'un arbre de décision : Un arbre de décision est construit à

partir de l'échantillon de données sélectionné. L'arbre de décision est construit en choisissant la meilleure caractéristique de séparation à chaque nœud.

(d) Répéter les étapes 2 et 3 : Les étapes 2 et 3 sont répétées plusieurs fois pour créer un ensemble d'arbres de décision.

(e) Prédiction : Pour prédire la classe ou la valeur d'un exemple inconnu, l'algorithme passe l'exemple à travers chaque arbre de décision de l'ensemble et prend la moyenne des prédictions pour obtenir la prédiction finale.L'algorithme Random Forest est une méthode d'apprentissage supervisé utilisée pour la classification, la régression et d'autres tâches de prédiction. Il est basé sur l'ensemble d'arbres de décision aléatoires.

Tout comme pour l'imputation avec KNN, les colonnes qui contiennent des données manquantes sont traitées comme la variable cible tandis que les autres colonnes sont utilisées comme variables d'entrée. Les valeurs manquantes dans la colonne cible sont prédites en utilisant les valeurs des autres colonnes. Nous utilisons cet algorithme en mode régression, puis retransformons les variables qui doivent l'être en catégorielles suivant le même processus que celui décrit lors de l'imputation avec la méthode des k plus proches voisins. Cette méthode, bien que plus coûteuse en ressource que l'algorithme KNN, permet souvent d'obtenir de meilleurs résultats que ce dernier.

2.3 Présentation et choix du modèle

2.3.1 Présentation des différents modèles travaillés

Après que le dataset fut étudié puis travaillé pour le rendre exploitable, nous avons testé différents modèles de maching learning, du plus simple à de plus complexes, pour obtenir le meilleur score de prédiction. Voici les différents modèles que nous avons étudié durant notre étude :

1. Régression logistique

   (a) Principe du modèle

   La régression logistique est un modèle statistique utilisé pour prédire la probabilité d'une occurrence d'un événement binaire (par exemple, 0 ou 1) en fonction de variables explicatives. La régression logistique utilise la fonction logistique (ou sigmoïde) pour transformer les prédictions linéaires en probabilités comprises entre 0 et 1.

   (b) Hyperparamètres principaux

   Nous avons joué sur la norme de la pénalité appliquée, le solver utilisé, c'est-à-dire l'algorithme d'optimisation utilisé, ainsi que multi_class, qui correspond à la façon dont le modèle doit être ajusté pour traiter les problèmes de classification multiclasse. Nous obtenons le résultat suivant

2. k-NN (k plus proches voisins)

   (a) Principe du modèle

   Le modèle k-NN est un algorithme d'apprentissage supervisé basé sur l'instance qui fonctionne en mémorisant les exemples d'apprentissage. Pour effectuer une prédiction pour une nouvelle instance, l'algorithme recherche les k exemples les plus proches (voisins) dans l'espace des caractéristiques et attribue la classe majoritaire parmi ces voisins à la nouvelle instance.

   (b) Hyperparamètres principaux

   Le paramètre principal sur lequel nous avons travaillé est le nombre de plus proches voisins k. C'est l'hyperparamètre le plus influent. Nous avons donc choisi le k le plus optimal. Voici le résultat :

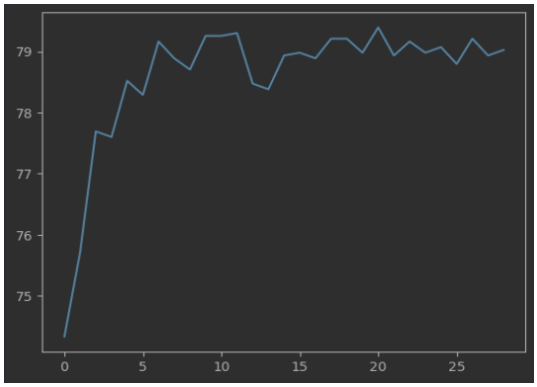

FIGURE 14 – Evolution du score de prédiction en fonction de k

3. Gradient Boosting XGBoost (eXtreme Gradient Boosting)

   (a) Principe du modèle

   XGBoost est une implémentation efficace de l'algorithme de gradient boosting pour l'optimisation d'arbres de décision. Le gradient boosting construit des modèles en ajoutant de manière itérative de nouveaux arbres de décision qui corrigent les erreurs des arbres précédents. XGBoost utilise une combinaison d'optimisation du gradient et de techniques de régularisation pour améliorer la performance et réduire le surajustement.

(b) Hyperparamètres principaux

Les hyperparamètres principaux sont la profondeur maximale (profondeur maximale des arbres de décision) et le nombre d'arbres (nombre d'arbres de décision à ajouter au modèle) . Nous obtenons le résultat suivant

4. Gradient Boosting avec CatBoost

(a) Principe du modèle

CatBoost est une bibliothèque open-source de gradient boosting pour les données catégorielles et numériques. Elle a été développée par la société Yandex et est disponible en open source depuis 2017. Le modèle CatBoost utilise un algorithme de régularisation appelé régularisation de bord. Cette technique consiste à réduire la complexité du modèle en limitant le nombre de feuilles dans chaque arbre de décision, ce qui permet de réduire le surapprentissage et d'améliorer les performances de prédiction.

(b) Hyperparamètres principaux

Les principaux hyperparamètres du modèle CatBoost sont les suivants :
— Le taux d'apprentissage (learning_rate) : il s'agit de la vitesse à laquelle le modèle apprend pendant l'entraînement.
— La profondeur maximale de l'arbre (max_depth) : elle détermine la profondeur maximale de chaque arbre de décision dans l'ensemble de modèles.
— Le taux de régularisation (reg_lambda) : il s'agit du taux de régularisation L2 qui contrôle l'amplitude des coefficients de pondération des variables.

2.3.2 Choix du modèle

Pour choisir le modèle qui nous permet d'atteindre la meilleure précision, on va implémenter plusieurs modèles et comparer les résultats obtenus avec ces différents modèles. On a choisi d'implémenter 4 modèles :
— Un modèle de régression logistique
— Un modèle KNN
— un modèle de gradient boosting XGBoost
— un modèle de gradient boosting CatBoost

Comme on l'a précisé au début du rapport, on va tester chaque modèle, pour chacun de nos 3 datasets. On sélectionnera ensuite le modèle ayant le meilleur score de prédiction.

Après avoir testé l'ensemble des modèles, on obtient les résultats suivants :

Tout d'abord, on remarque que l'influence du choix du modèle a un impact modéré sur le score de prédiction : chaque modèle nous permet d'obtenir un score de prédiction compris entre 0.76104 et 0.80421.

On va donc sélectionner le modèle CatBoost, avec le dataset où les valeurs manquantes ont été remplis grâce à l'algorithme de machine learning Random Forest.

	Dataset 1 - NA rempli avec la moyenne	Dataset 2 - NA rempli avec algorithme KNN	Dataset 3 - NA rempli avec Random Forest
Régression logistique	0,7884	0,78536	0,78957
KNN	0,77601	0,76104	0,77811
Gradient Boosting XGBoost	0,79588	0,78653	0,80032
Gradient Boosting CatBoost	0,8038	0,79869	0,80421

FIGURE 15 – Résultats soumis sur Kaggle, en fonction du modèle et du dataset

2.4 Optimisation des hyperparamètres

La première étape de l'optimisation des hyperparamètres consiste à sélectionner les hyperparamètres que vous souhaitez optimiser : vous ne pouvez pas tous les optimiser, par exemple le modèle de régresseur XGBoost comporte plus de 20 hyperparamètres. Nous nous concentrerons donc sur les hyperparamètres qui ont le plus d'influence sur la qualité de prédiction du modèle. De plus, nous devons sélectionner une fourchette de valeurs pour chaque paramètre, afin de limiter l'espace de recherche.

Voici une liste des trois algorithmes les plus répandus pour effectuer l'optimisation des hyperparamètres :

— Grid search : Elle effectue une recherche exhaustive en évaluant toutes les combinaisons de candidats. Cette solution n'est pas idéale car elle est très coûteuse sur le plan informatique.

— Random Search : Elle effectue une recherche aléatoire dans l'espace de recherche : l'utilisateur spécifie ici le nombre de recherches. Bien que cette méthode permette à l'utilisateur de préciser la zone de recherche en éliminant les zones où les résultats sont médiocres, elle est coûteuse et ne permet pas toujours de trouver la meilleure combinaison d'hyperparamètres.

— Bayesian Search : Contrairement à la méthode Grid Search et à la méthode Random Search, la méthode Bayesian Search utilise les itérations précédentes pour guider les itérations suivantes. Elle consiste à construire une distribution de fonctions (processus gaussien) qui décrit au mieux la fonction à optimiser. Dans le cas présent, l'optimisation des hyperparamètres, la fonction à optimiser est celle qui, étant donné les hyperparamètres, retourne la performance du modèle entraîné auquel ils conduiraient. Après chaque étape, cette distribution de fonctions est mise à jour et l'algorithme détecte les régions de l'espace des hyperparamètres qui sont les plus intéressantes à explorer et celles qui ne le sont pas. Après un nombre défini d'itérations, l'algorithme s'arrête et renvoie le tuple optimal. L'optimisation bayésienne est une méthode plus efficace pour explorer les possibilités.

La méthode Bayesian Search ressort comme étant la méthode la plus efficace : on a donc décidé d'implémenter cette méthode. Pour cela, on a utilisé la librairie hyperopt, une librairie très efficace pour traiter des problèmes d'optimisation.

On commence par définir l'espace de recherche des paramètres : pour chaque paramètre que l'on souhaite fixer, on définit un intervalle de recherche. Ensuite, on définit une fonction objective : dans notre cas, on va séparer les données en 4 parties : X_train, Y_train, X_test, Y_test, puis on va prédire la variable Transported. Finalement, la fonction renvoie le score de prédiction, qui est donc la variable finale à optimiser.

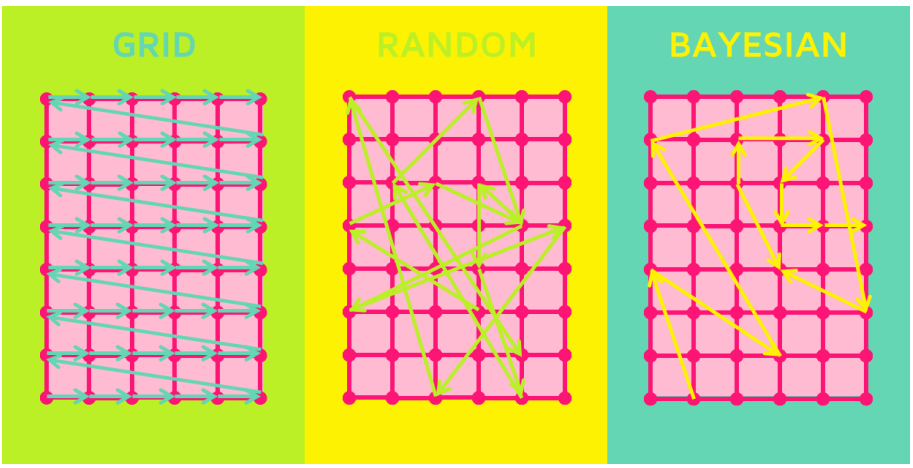

FIGURE 16 – Représentation des 3 méthodes d'optimisation des hyperparamètres

Après plusieurs essais, nous avons obtenu des résultats peu convaincants. La fonction nous a renvoyé la liste des paramètres optimaux, que l'on va utiliser pour faire nos prédictions. Une fois les résultats soumis sur Kaggle, le score de prédiction diffère fortement du résultat obtenu lors de l'optimisation. Cela vient du fait que l'on optimise les hyperparamètres sur notre échantillon train : ce n'est pas idéal. De plus, la méthode Bayesian Search est une méthode assez complexe, on pense que l'on ne maîtrise pas correctement les fondamentaux de la statistique bayésienne pour utiliser cette méthode.

On a donc décidé d'implémenter la méthode Grid Search pour optimiser les hyperparamètres. De la même manière que pour la méthode Bayesian Search, on commence par définir l'espace des paramètres. Ensuite, la méthode Grid Search va tester l'ensemble des combinaisons possibles dans l'espace des paramètres. L'application de cette méthode nous a permis de trouver une combinaison d'hyperparamètres qui nous permet d'augmenter légèrement le score de prédiction final.

Finalement, grâce à l'optimisation des hyperparamètres, notre score de prédiction est passé de 0.80421 à 0.80523, soit une hausse de 0.00102, ce qui représente environ 3444 passagers prédit correctement sur 4277, contre 3440 avant l'optimisation.

FIGURE 17 – Dernier résultat obtenu sur Kaggle

3 Conclusion

Le projet Kaggle Spaceship Titanic nous a permis de nous exercer pour la première fois à l'apprentissage automatique. Ce projet nous a permis de découvrir la chaîne complète des différentes étapes à réaliser lors d'un projet de machine learning. Il nous a d'abord permis d'apprendre à traiter un jeu de données, pour le mettre au propre et l'utiliser à bon escient. Nous avons également appris à créer des variables pour apporter de l'information. On a pu tester plusieurs méthodes pour remplir les valeurs manquantes et comparer leur efficacité. Ensuite, on a pu tester plusieurs modèles de machine learning et comparer leur performance. Pour finir, on a pu découvrir plusieurs méthodes d'optimisation des hyperparamètres.

Ce projet nous a permis de tirer certaines conclusions relatives au déroulement d'un projet de machine learning. Tout d'abord, on a remarqué que le traitement des données est une étape cruciale : en effet, c'est la partie du projet qui a le plus influencé notre score de prédiction. Le choix du modèle a évidemment un impact, mais du pire modèle au meilleur, la variation de notre score de prédiction n'est que d'environ 0.05. En revanche, les choix de créations de variables peuvent avoir un impact beaucoup plus important : on a remarqué des différences de l'ordre de 0.15 sur notre score de prédiction en fonction du nombre de variables créées dans le dataset.

De plus, nous avons remarqué que l'optimisation des hyperparamètres ne nous permet pas d'augmenter considérablement le score de prédiction. On imagine que cela est valable pour ce projet et que dans d'autres cas, l'optimisation des hyperparamètres peut apporter un gain de précision plus important.

Liste des figures