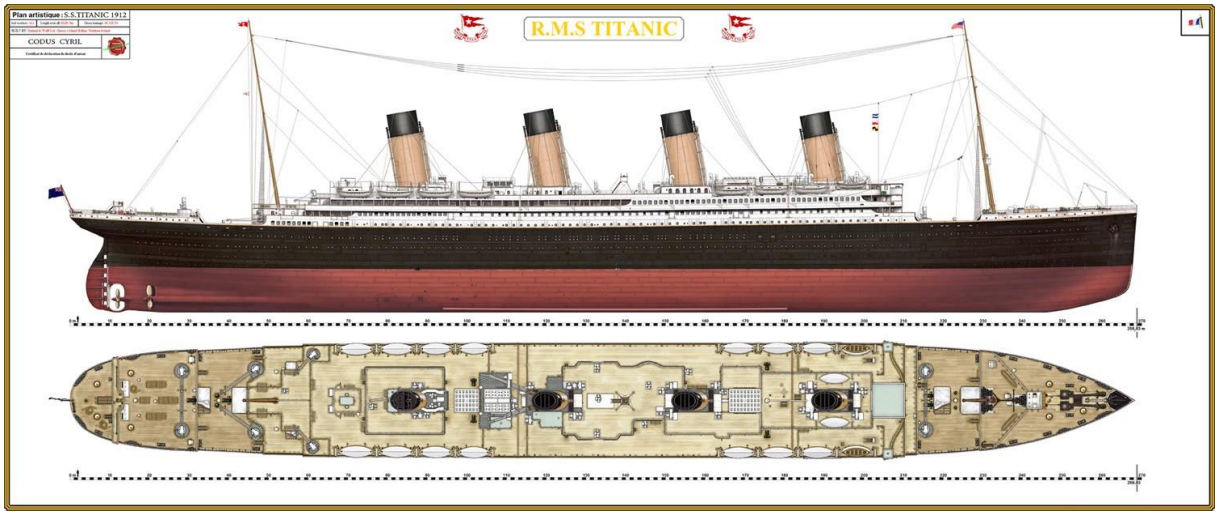
Techniques d’apprentissage

# Projet

IFT 712




Arnaud Godart 19 156 869
Julien Brosseau 19 124 617

Session d’automne 2019

# Table des matières

# Introduction

Dans le cadre du cours IFT712 de la session d'automne 2019, nous sommes amenés à utiliser les modèles vus en Machine Learning sur un jeu de données de notre choix. Le jeu de données du Titanic recense les caractéristiques des passagers présent lors du naufrage du paquebot, celles de leur billet ainsi que leur survie. Ce jeu de données est très connu dans le domaine de l’analyse de données. Il nous permet ici de mettre en place plusieurs stratégies de machine learning dans le prolongement des enseignements de techniques d’apprentissage de Machine Learning.

Peut-on prédire la survie d’un passager du Titanic en fonction de son profil ?



Julien Brosseau
Arnaud Godart

# 1. Démarche scientifique

## 1.1. Traitement des données

Pour que les classifications utilisées dans ce projet donnent des résultats optimums, il est nécessaire d'effectuer un traitement des données récupérées sur *Kaggle*[1] avant de les utiliser au sein des modèles de techniques d'apprentissage.

Les données d'entrainements pour la base données « Titanic », cette dernière est constituée d'un ensemble de 891 passagers différents pour lesquels on a 12 types d'informations associés.

Les données de tests pour la base données « Titanic », cette dernière est constituée d'un ensemble de 418 passagers différents pour lesquels on a 11 types d'informations associés.

Ainsi, en fonction des prédictions émises par le modèle et les valeurs réelles issues de ce fichier, nous sommes à même de déterminer la performance du modèle de classification par simple comparaison.

Une fois les différentes variables et leurs valeurs observées, nous nous sommes aperçus que certaines étaient redondantes, trop incomplètes ou pas intéressantes à conserver pour la suite de ce projet. En ce sens, nous avons supprimé ou recodé plusieurs variables présentes dans le jeu de données.

## 1.2. Sélection des variables

La base de données[2] récupérée sur *Kaggle* contient trois fichiers .csv :

- GENDER_SUBMISSION.CSV (2 variables)
- TEST.CSV (11 variables)
- TRAIN.CSV (12 variables)

Le fichier « train.csv » contient toutes les caractéristiques disponibles pour une partie des passagers (418 passagers) y compris l'information sur leur survie pour chacun d'entre eux. Pour l'autre partie des passagers (891 passagers), l'information sur la survie est disponible dans le fichier « gender_submission.csv » tandis que les autres informations pour ce groupe de passagers se trouvent dans le fichier « test.csv ». Ces 2 derniers fichiers ont été utilisé pour vérifier les performances de chacune des méthodes implémentées au cours de ce projet après avoir été entrainés chacune sur le premier fichier contenant moins de passagers.

Le dataset du fichier d'entrainement est séparé en 2. Le nouveau dataset *x_train* intègre toutes les caractéristiques à l'exception de l'information de survie insérée dans le dataset *t_train*. Ce sont les variables d'entrée des modèles. Le but étant de prévoir si un passager, pris au hasard, a une chance de survivre ou non, c'est pourquoi nous avons procédé ainsi.

Les datasets X_TEST et T_TEST reprennent respectivement les variables des fichiers « test.csv » et l'information de survie du fichier TRAIN.CSV.

Les datasets X_TEST et X_TRAIN ont donc la même architecture d'information bien qu'ayant un ensemble de passagers différents. Certaines variables contenues dans ces datasets ne sont pas nécessaires pour

déterminer la survie d'un passager. Nous avons donc supprimé les variables suivantes avant de faire effectuer le travail d'apprentissage.

- PASSENGERID indiquant l'identifiant d'un passager du Titanic.
- NAME renseignant sur le nom de chaque passager du Titanic.
- TICKET indiquant le numéro de ticket de chaque passager du Titanic.
- CABIN renseignant sur la cabine de chaque passager du Titanic
  (la classe du passager est suffisante pour obtenir des zones au sein du Titanic).

Ces variables contenaient soit trop d'enregistrements uniques soit beaucoup de valeurs étaient manquantes en plus d'être recoupées par d'autres variables pour pouvoir être exploitées.

## 1.3. Cas des valeurs manquantes

Ce problème de valeurs manquantes a aussi été rencontré pour d'autres variables du dataset. C'est par exemple le cas pour la variable EMBARKED ou AGE indiquant respectivement le lieu d'embarquement des passagers ou l'âge du passager. Pour la première variable, nous avons décidé de remplacer les valeurs « null » par le mode de la variable, c'est-à-dire la donnée la plus fréquemment trouvée. Pour la deuxième, nous avons remplacé les valeurs manquantes par la valeur médiane du groupe de passagers auquel il appartient (homme, 1ère classe par exemple).

## 1.4. Transformation des variables en variables quantitatives discrètes en valeurs numériques

Les variables qualitatives ont été transformées en variables quantitatives. Le genre des passagers indiqué par la variable SEX par « male » ou « female » a été remplacé par respectivement « 1 » ou « 0 ». De la même manière, le lieu d'embarquement défini par « S », « C » ou « Q » dans la variable *Embarked* a été remplacé par respectivement « 0 », « 1 » ou « 2 ».

D'autres variables quantitatives continues ont été transformés en variables quantitatives discrètes par la création d'intervalles comme c'est le cas pour la variable AGE ou FARE. Plusieurs possibilités d'intervalles ont été envisagées. Nous avons finalement arrêté 5 classes pour l'âge (enfant, adolescent, jeune, adulte et sénior) et 4 classes pour le prix du billet (pauvre, modeste, aisé et riche).

L'ensemble de ces étapes nous ont permis d'effectuer un traitement des données, nous sommes maintenant prêts à mettre en place les méthodes de techniques d'apprentissage.

## 1.5. Méthodes de techniques d'apprentissage

Le projet consistait à implémenter au moins six méthodes de classification différentes à l'aide de la bibliothèque « *sklearn* ». Nous avons finalement implémenté 7 méthodes, en décidant d'intégrer *Adaboost* au cours de ce projet. Pour chacune d'entre-elle, nous avons travaillé sur certains hyperparamètres pour

améliorer les résultats. La validation croisée a généralement été faite à l'aide de la bibliothèque « GridSearchCV[3] » sauf quand la librairie du modèle intégrait déjà la fonctionnalité.

L'analyse des résultats des méthodes est réalisée dans une partie séparée.

### 1.5.1. Régression logistique

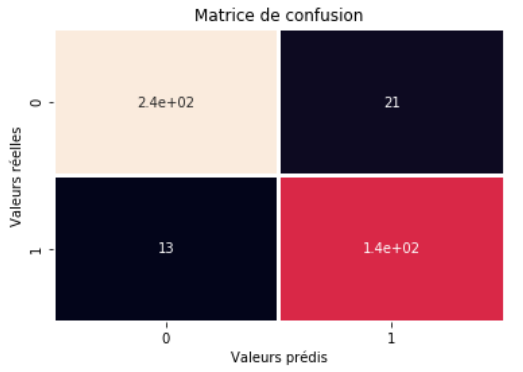
La classification par la régression logistique[4] est un modèle de régression binomiale. En d'autres termes, il s'agit de générer un modèle mathématique simple. C'est sans doute l'une des méthodes les plus utilisées dans un but pédagogique par sa simplicité de compréhension et sa rapidité d'implémentation. Elle est considérée comme l'une des méthodes de base en techniques d'apprentissage.

De plus, nous avons pu implémenter cette méthode au cours de l'un des TP de techniques d'apprentissage dont dépend ce projet.

Concernant les choix d'implémentation vis à vis des hyperparamètres, nous avons choisi le terme « Cs ». Il s'agit de l'inverse du terme de régularisation. La régularisation permet d'éviter le surapprentissage de notre modèle. Plus « Cs » est petit, plus la régularisation sera forte. Après plusieurs tentatives, nous avons choisi une fourchette de valeurs entre 3 et 4 avec un pas de 0.001 pour cet hyperparamètre. Cet intervalle générait les plus faibles taux d'erreurs lors de l'exécution du modèle.

```
Vous avez entré la méthode : logistic

Erreur d'entrainement : 17.84511784511784 %
Erreur de test : 8.133971291866027 %
Meilleur hyperparametre : [3.64]
```




```
--- 25.801653623580933 secondes ---
```

**Figure 1 : Sortie de la méthode régression logistique**

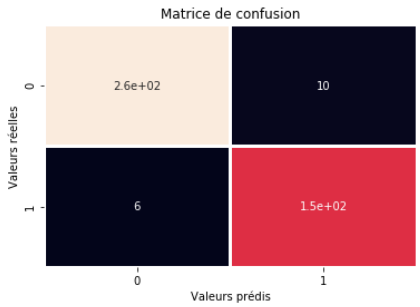
### 1.5.2. Ridge

La classification par Ridge est aussi appelée la régularisation de Tikhonov[5]. Cette méthode de régularisation est sans doute la plus utilisée pour la résolution de problèmes qui ne sont pas bien posés et pour les problèmes inverses. Ridge est connexe à l'algorithme de Levenberg-Marquardt pour la résolution de problèmes non-linéaires de moindres carrés.

Nous avons choisi de travailler sur l'hyperparamètre « ALPHA ». Il s'agit du terme de régularisation. Comme pour la classification par régression logistique, il s'agit d'éviter le surapprentissage de notre modèle. La meilleure fourchette constatée au cours des exécutions se traduit par des valeurs entre 0 et $10^{-3}$ avec un pas de $0.5*10^{-5}$.

```
Vous avez entré la méthode : ridge

Erreur d'entrainement : 19.191919191919194 %
Erreur de test : 3.8277511961722466 %
Meilleur hyperparametre : 0.000995
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\model_selection
\_search.py:813: DeprecationWarning: The default of the `iid` parameter
will change from True to False in version 0.22 and will be removed in 0.24.
This will change numeric results when test-set sizes are unequal.
  DeprecationWarning)
```




```
--- 17.713644981384277 secondes ---
```

FIGURE 2 : SORTIE DE LA MÉTHODE RIDGE



Julien Brosseau
Arnaud Godart

### 1.5.3. Adaboost

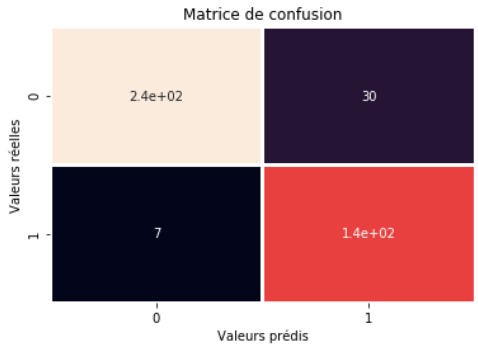
La classification par *Adaboost*[6] est basée sur un algorithme de *boosting*. *Adaboost* est souvent utilisée en combinaison avec des modèles à faible capacité (qui ont une tendance au sous-apprentissage) afin d'améliorer les performances de ces derniers. Il est notamment sensible aux données bruitées ou peu corrélées. La méthode consiste à combiner des modèles linéaires simples (x = 1 ou y = 3).

Cette méthode n'a pas été spécifiquement étudié en TP mais a été mentionné durant les cours de techniques d'apprentissage. Nous nous renseignés de manière plus approfondie sur cette dernière grâce à la libraire « *sklearn* ».

Les hyperparamètres travaillés pour cette méthode sont les termes « N_ESTIMATORS » et « LEARNING_RATE ». Le premier correspond au nombre maximum d'estimation où le « *boosting* » se termine. Quant au second, il définit le taux d'apprentissage. Plus ce dernier est grand, moins la contribution des classifieurs est élevée. C'est-à-dire qu'*Adaboost* prend moins en compte les modèles qui ont été générés au préalable. Nous avons retenu des fourchettes de valeur entre 5 et 15 avec un pas de 1 pour « N_ESTIMATORS » et entre 0.8 et 1.8 avec un pas de 0.1 pour « LEARNING_RATE » comme les meilleures valeurs pour ces hyperparamètre.

```
Vous avez entré la méthode : adaboost

Erreur d'entrainement : 17.39618406285073 %
Erreur de test : 8.851674641148321 %
Meilleur hyperparametre : {'learning_rate': 1.5999999999999999,
'n_estimators': 9}
```




```
--- 29.50943350791931 secondes ---
```

**FIGURE 3 : SORTIE DE LA MÉTHODE ADABOOST**

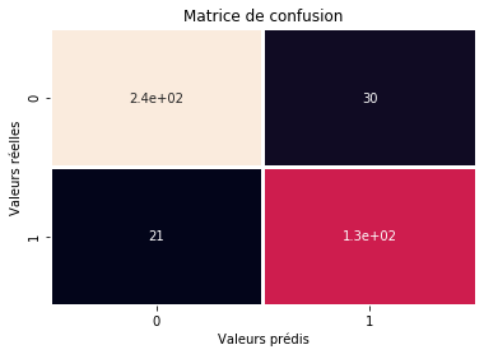
### 1.5.4. K-Nearest Neighbors

La classification par *K-Nearest Neighbors (K-NN)*[7] consiste à étiqueter une variable en fonction de son environnement. En d'autres termes, on prend les voisins les plus proches de la variable considérée et par un principe de vote majoritaire, on lui attribue la classe la plus représentée dans son voisinage. La méthode *K-NN* est basée sur l'apprentissage préalable, ou l'apprentissage faible, où la fonction est évaluée localement. Le calcul définitif étant effectué à l'issue de la classification.

Les hyperparamètres sur lesquels nous avons plus particulièrement porté notre attention sur cette méthode sont « N_NEIGHBORS », « ALGORITHM » et « LEAF_SIZE ». Le premier définit le nombre de voisins utilisés pour la classification. Le deuxième indique l'algorithme suivi par K-Nearest Neighbors afin d'effectuer la classification. Quant au troisième, il permet d'améliorer la rapidité d'apprentissage du modèle (pour les algorithmes "ball_tree" et "kd_tree").

```
Vous avez entré la méthode : nearestneighbors

Erreur d'entrainement : 16.498316498316502 %
Erreur de test : 12.200956937799045 %
Meilleur hyperparametre : {'algorithm': 'kd_tree', 'leaf_size': 27,
'n_neighbors': 7}
```




```
--- 65.95109724998474 secondes ---
```

FIGURE 4 : SORTIE DE LA MÉTHODE K-NEAREST NEIGHBORS



Julien Brosseau
Arnaud Godart

### 1.5.5. Perceptron multi couches

Le perceptron multicouche (*MLPClassifier* en anglais)[8] est couramment utilisé lorsque les données ne peuvent être séparés linéairement. C'est le cas lors de la représentation linéaire de l'opérateur logique XOR aussi appelé ou « exclusif ». Il est la version évoluée du perceptron classique qui effectue son apprentissage à l'aide de valeurs binaires (0/1). Il faut veiller à ne pas mettre trop de neurones au sein de la couche cachée pour éviter un sur-apprentissage.

Nous avons choisi de travailler sur l'hyperparamètre ALPHA correspondant au terme de régularisation L2. De meilleurs résultats sont obtenus avec un intervalle de 0 à 0.05 avec un pas de 0.01.


**FIGURE 5 : SORTIE DE LA MÉTHODE PERCEPTRON MULTI COUCHES**



Julien Brosseau
Arnaud Godart

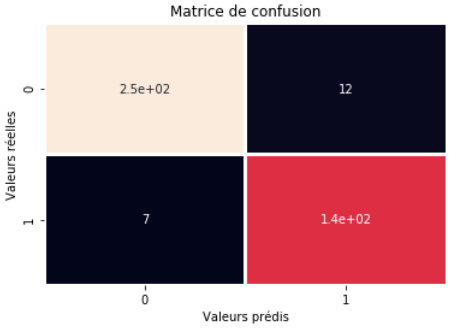
### 1.5.6. SVM

Les machines à vecteurs de support (SVM en anglais)[9] font apparaître la notion de marge. La séparation des classes est matérialisée par une frontière, la méthode fait en sorte que les points soient le plus éloignée de celle-ci. L'espace ainsi formé est appelé la marge. Les points les plus proches de cette marge constituent les vecteurs de support. Ces points sont particulièrement importants, c'est sur eux que la méthode se concentre pour effectuer une « bonne » classification.

Sur ce modèle nous travaillons sur le terme de régularisation C et sur le noyau KERNEL. Nous avons choisi un intervalle de 0.5 à 1 avec un pas de 0.01 pour ce premier hyperparamètre et la possibilité d'utiliser un noyau de type LINÉAIRE ou RBF pour ce second hyperparamètre même si l'algorithme a choisi exclusivement ce deuxième type noyau plus robuste lors de ses exécutions.

```
Vous avez entré la méthode : svm

Erreur d'entrainement : 16.947250280583614 %
Erreur de test : 4.545454545454541 %
Meilleur hyperparametre : {'C': 0.8900000000000003, 'kernel': 'rbf'}
```




```
--- 56.259591579437256 secondes ---
```

FIGURE 6 : SORTIE DE LA MÉTHODE SVM



Julien Brosseau
Arnaud Godart

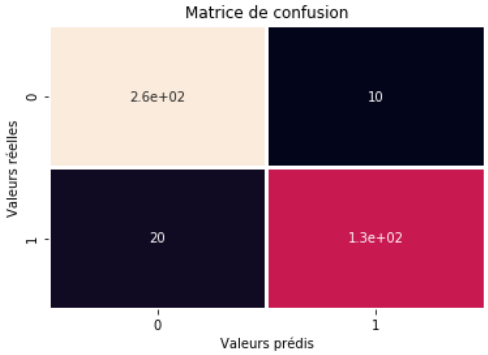
### 1.5.7. Random Forest

La méthode *Random Forest*[10] s'appuie sur la génération d'arbres combinés pour mener à bien la classification recherchée. C'et le principe de forêt d'arbres décisionnels.

Pour cette méthode, nous nous sommes concentrés sur deux hyperparamètre : N_ESTIMATORS qui désigne le nombre d'arbre dans la forêt puis MAX_DEPTH pour la profondeur maximale de l'arbre. Ces hyperparamètres sont nécessairement des entiers. Les meilleurs résultats étaient obtenus lorsque le nombre d'arbres variait de 15 à 20 et leur profondeur de 3 à 7.

```
Vous avez entré la méthode : randomforest

Erreur d'entrainement : 17.059483726150393 %
Erreur de test : 7.177033492822971 %
Meilleur hyperparametre : {'max_depth': 4, 'n_estimators': 17}
```




```
--- 10.782688856124878 secondes ---
```

FIGURE 7 : SORTIE DE LA MÉTHODE RANDOM FOREST



Julien Brosseau
Arnaud Godart

# 2. Validation croisée & Recherche d'hyperparamètres

## 2.1. Choix pour la validation croisée

Pour tous ces modèles de classification implémentés au cours du projet, le paramètre CV (pour *Cross Validation*) est à déterminer. Afin d'établir au mieux ce paramètre, nous nous sommes basés sur les explications données en cours vis-à-vis de la validation croisée.

Le but de cette technique est de diviser notre ensemble en plusieurs sous-ensembles. K définit généralement le nombre de sous-ensemble, c'est le paramètre recherché. Par la suite, chaque sous-ensemble est entrainé indépendamment les uns des autres. Finalement, nous obtenons k erreurs quadratiques moyennes (une par sous-ensemble après entraînement). En calculant la moyenne de toutes ces erreurs nous obtenons l'erreur de prédiction de notre ensemble de départ.

Par définition, plus nous avons de sous-ensembles, plus nous entraînons le modèle sur les données d'entraînements. Plus nous entraînons le modèle sur les données, moins il y a d'erreurs constatées sur la classification des données. Et moins d'erreurs, cela se traduit par un modèle mieux entrainé contribuant à fournir de meilleurs résultats.

Dans la réalité, il est difficile, voire impossible dans la majorité des cas, d'obtenir une erreur de prédiction de zéro ou une valeur tendant vers cette dernière. En effet, il y a toujours des données aberrantes, dont il ne faut tenir compte dans les différents jeux de données ou des données reflétant un comportement totalement différent des autres.

Dans ce contexte, nous constatons que le fait d'augmenter la valeur du paramètre « CV » augmente la précision du modèle mais seulement jusqu'à un certain point. Dans notre cas, avec la base de données « Titanic », nous avons pu observer que ce palier se situait aux alentours de la valeur CV = 15. Nous avons donc décidé de fixer cette valeur pour ce paramètre. En effet, continuer l'itération ne fait qu'augmenter le temps de calcul pour entraîner nos modèles de classification, sans aucun intérêt au-delà.

## 2.2. Choix pour la recherche d'hyperparamètres

Pour chaque modèle de classification que nous avons implémenté dans ce projet, au moins un hyperparamètre (ALPHA, CL, N_NEIGHBORS, …) a été travaillé. Cela afin de ne pas laisser les valeurs par défaut déjà disponible et de mobiliser nos connaissances issues des cours et TP de technique d'apprentissage en la matière.

Par définition, ces paramètres sont définis avant le processus d'apprentissage. En fonction de la complexité du modèle de classification, nous pouvons avoir aucun hyperparamètre, comme un seul ou plusieurs. Ces derniers n'influent pas sur la performance d'un modèle mais sur sa rapidité et son efficacité d'apprentissage par rapport à un jeu de données.

Concrètement, chaque ensemble d'entraînement est différent et implique une variation des hyperparamètres idéaux. C'est à nous de les trouver en les faisant évoluer et en les testant.

Il n'existe pas réellement de méthodes en pratique pour trouver les valeurs idéales des hyperparamètres au vu de l'infinité des valeurs qu'ils peuvent prendre. À ce propos, la méthode recommandée est de laisser une fourchette autour de la valeur de l'hyperparamètre qui nous semble la meilleure. C'est ainsi que nous avons procédé dans ce projet.

## 2.3. Rafraîchissement des données

Depuis le début, nous avons toujours utilisé le même ensemble pour entraîner nos méthodes, à savoir les données ici du fichier « TRAIN.CSV ». De la même manière, nous avons toujours testé l'apprentissage de nos méthodes avec les données issues du fichier « TEST.CSV ». Par définition, l'apprentissage du modèle et le test de ce dernier doivent être fait avec deux jeux de données différents.

Mise à part ce cas basique, nous avions émis la possibilité de « créer » de nouvelles données ou de « mélanger » les données déjà à notre disposition afin de potentiellement améliorer l'entraînement de nos différents modèles de classification. En effet, il nous est venu à l'idée d'implémenter une méthode de *bootstrapping* dans notre projet. L'idée du *bootstrapping* est de créer des sous-ensembles, sur le principe des échantillons, composés d'un certain nombre d'éléments de l'ensemble principal. Le but ici, comme d'habitude, est de permettre à notre modèle d'effectuer un meilleur apprentissage.

Nous n'avons malheureusement pas pu implémenter cette méthode en raison de contrainte de temps dû à la réalisation de plusieurs projets en parallèle. Cela aurait permis, en la testant sur nos modèles, d'obtenir potentiellement un meilleur apprentissage. Néanmoins, nous savons que le *bootstrapping* n'améliore pas de façon drastique les résultats finaux. Aux vues de ces derniers, nous n'avons pas considéré la méthode comme étant vitale au projet puisque Nos résultats nous apparaissant déjà comme satisfaisant, il ne nous est pas apparu prioritaire de mettre en place cette technique.

# 3. Analyse des résultats

À partir des informations connus sur le passager, le modèle est capable de prédire s'il va survivre ou non.

Les modèles utilisés ont chacun des paramètres et des façons de fonctionner différentes mais permettent en les confrontant de donner plus de robustesse à la prédiction finale.

Les différents modèles utilisés ont des taux d'erreurs d'apprentissage et de test assez faibles. Ils sont toujours inférieurs à 20%. La matrice de confusion nous renseigne systématiquement sur ces informations et permet de se rendre compte des données bien classés et de celles qui ne le sont pas. Les faux positifs et les faux négatifs sont semblables avec de très faibles proportions dans la majorité des méthodes.

De la même manière nous pouvons affirmer grâce au graphique suivant qu'une passagère a moins de 20 % de risque de mourir et donc que sa survie est quasiment assurée, nos modèles vont délivrer dans ce cas une prédiction de survie, toutefois cela peut être nuancé par d'autres informations connues.

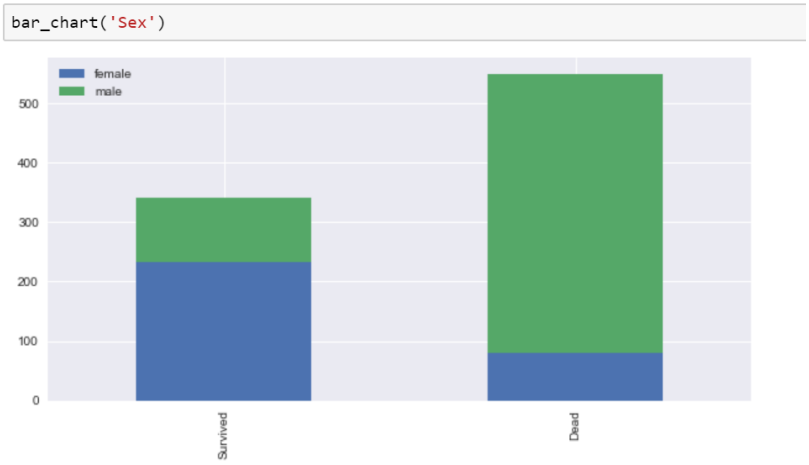



FIGURE 8 : GRAPHIQUE REPRÉSENTANT LA SURVIE EN FONCTION DU SEXE DU PASSAGER

En effet, bien qu'une proportion relativement similaire de passager ait survécu dans les 3 classes, la 3ème classe du paquebot accueillait beaucoup plus de monde et est de fait celle ayant la probabilité la plus faible de survie. Ainsi, le modèle pourrait nuancer cet avis si cette passagère était issue de la 3ème classe.

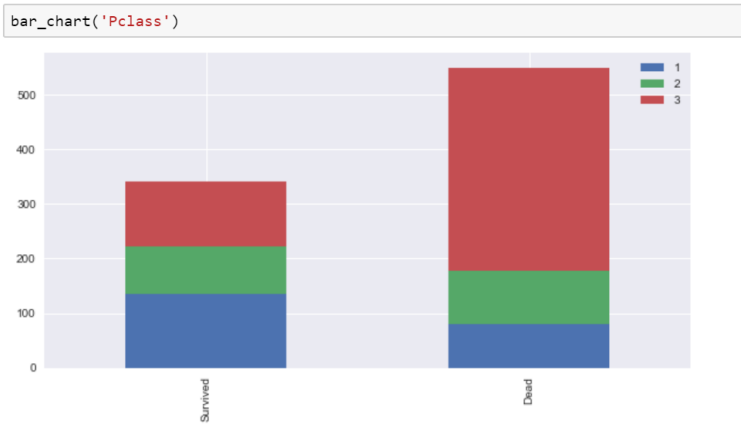



FIGURE 9 : GRAPHIQUE REPRÉSENTANT LA SURVIE EN FONCTION DE LA CLASSE DU PASSAGER

C'est là qu'intervient la puissance de nos modèles pour nous indiquer une prédiction à un taux d'erreur proche de celui de test.

Si l'erreur d'entraînement de nos modèles est systématiquement autour de 16 à 17 %, celle de test est toujours nettement inférieure. Cela est synonyme d'un bon apprentissage. Nos modèles ne sont donc pas confrontés au surapprentissage ou au sous apprentissage. Ils ont donc un bon équilibre biais-variance. Si le perceptron a le taux d'erreur d'entraînement le plus faible avec 16,5 %, la méthode ridge, à elle, le taux d'erreur de test le plus faible avec 3,83 %. Ce dernier résultat est d'autant plus important qu'il permet de valider qu'il s'agit de la méthode qui s'en sort le mieux dans l'exercice de prédiction.

Les matrices de confusion confirment les bons résultats des taux d'erreurs analysées ci-dessus.

Toutefois, le taux d'erreur d'apprentissage très élevé de la méthode ridge est à prendre en compte ; avec plus de 19 %, il est le plus élevé de tous les modèles. Ce contraste entre les 2 taux de cette méthode est surprenant.

Les taux d'entraînement plus élevés pourraient s'expliquer par la présence de données aberrantes.

Un autre critère que l'on peut noter est le temps d'exécution de ces méthodes. Celui-ci varie d'une dizaine de secondes à un peu plus d'une minute à l'exception du perceptron qui prends 129 secondes, soit plus de 2 minutes pour être exécuté. Random forest est la méthode la plus rapide avec près de 11 seconds nécessaires à son exécution. Elle possède également un bon taux d'erreur de test à 7 %, inférieur à son taux d'erreur d'apprentissage, qui est de 17 %. Ces temps restent faibles dans notre projet mais peuvent avoir un impact conséquent pour des datasets plus volumineux.

La méthode Random Forest semble être la meilleure ici avec un bon compromis des différentes mesures.

# Conclusion

Les différentes techniques d'apprentissage issues du Machine Learning permettent de classifier les passagers présents à bord du Titanic et de prédire à posteriori leur survie éventuel en fonction de leur profil. Ce projet à été l'occasion de mettre en application les connaissances acquises au cours de l'enseignement de techniques d'apprentissage et de se familiariser à la notion de validation croisée. Les modèles implémentés ici sont réutilisables à condition de les adapter et pourraient ainsi permettre d'obtenir de nouvelles informations dans de nombreux domaines.

# Bibliographie

[1] « Kaggle: Your Home for Data Science », consulté le 4 décembre 2019, https://www.kaggle.com/.
[2] « Titanic: Machine Learning from Disaster », consulté le 3 décembre 2019, https://kaggle.com/c/titanic.
[3] « sklearn.model_selection.GridSearchCV — scikit-learn 0.22 documentation », consulté le 4 décembre 2019, https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html?highlight=gridsearchcv#sklearn.model_selection.GridSearchCV.
[4] « Régression logistique », in *Wikipédia*, 21 novembre 2019, https://fr.wikipedia.org/w/index.php?title=R%C3%A9gression_logistique&oldid=164728021.
[5] « Régularisation de Tikhonov », in *Wikipédia*, 8 janvier 2019, https://fr.wikipedia.org/w/index.php?title=R%C3%A9gularisation_de_Tikhonov&oldid=155610151.
[6] « AdaBoost », in Wikipédia, 16 octobre 2019, https://fr.wikipedia.org/w/index.php?title=AdaBoost&oldid=163601524.
[7] « Méthode des k plus proches voisins », in Wikipédia, 22 novembre 2019, https://fr.wikipedia.org/w/index.php?title=M%C3%A9thode_des_k_plus_proches_voisins&oldid=164741704.
[8] Amal Nair, « A Beginner's Guide To Scikit-Learn's MLPClassifier », Analytics India Magazine (blog), 20 juin 2019, https://analyticsindiamag.com/a-beginners-guide-to-scikit-learns-mlpclassifier/.
[9] « Support Vector Machine — Introduction to Machine Learning Algorithms », consulté le 4 décembre 2019, https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47.
[10] « Understanding Random Forest - Towards Data Science », consulté le 3 décembre 2019, https://towardsdatascience.com/understanding-random-forest-58381e0602d2.