POLYTECH SORBONNE - MAIN5

Projet Apprentissage Statistique

Projet : Higgs Boson Machine Learning Challenge

Étudiant : Vincent Liu

Encadrants : Olivier Schwander
Patrick Gallinari

Année 2019-2020

Table des matières

1 Introduction

Le dataset contient 818 238 évènements, il est partitionné en 4 parties correspondant au jeu de données de train, deux jeux de données de test (leaderboard public et privé Kaggle) et à une partie inutilisée lors du challenge.

Pour le projet, je vais utiliser le jeu de donnée de train pour l'apprentissage (250 000 évènements) et le jeu de donnée du leaderboard public Kaggle comme test.

Chaque évènement est décrit par 30 features correspondant à des grandeurs physiques. Il s'agit d'un problème de classification binaire où il faut prédire deux catégories d'évènement :

— background (b)

— signal (s) considéré comme la classe positive

De plus, un poids est attribué à chaque évènement, qui reflète son importance. Le poids est utilisé pour calculer une pondération lors du calcul des vrais positifs et des faux positifs. La mesure du modèle sera effectuée avec l'AMS.

2 Observation et analyse des données

Tout d'abord, on observe que le jeu de donnée est **déséquilibré**, il y a deux fois plus d'évènements background que signal :

— 164 333 évènements background (b)

— 85 667 évènements signal (s)

2.1 Données manquantes

Les données manquantes sont encodées par un code d'erreur -999. Sur le jeu de donnée d'entraînement, 1580 052 valeurs sont manquantes, ce qui représente 21% des valeurs. Il y a 12 features qui présentent des valeurs manquantes. 181 886 évènements (73% des évènements) sont décrites par au moins une valeur manquante. Cela constitue un trop grand nombre pour pouvoir supprimer des lignes ou des colonnes entières.

Les valeurs manquantes sont liées par la variable PRI_jet_num qui est un nombre entre 0 et 3. On observe sur la Figure 1 que les valeurs manquantes sont plus nombreuses lorsque on a un faible nombre de jets (jet num égale à 0 ou 1).

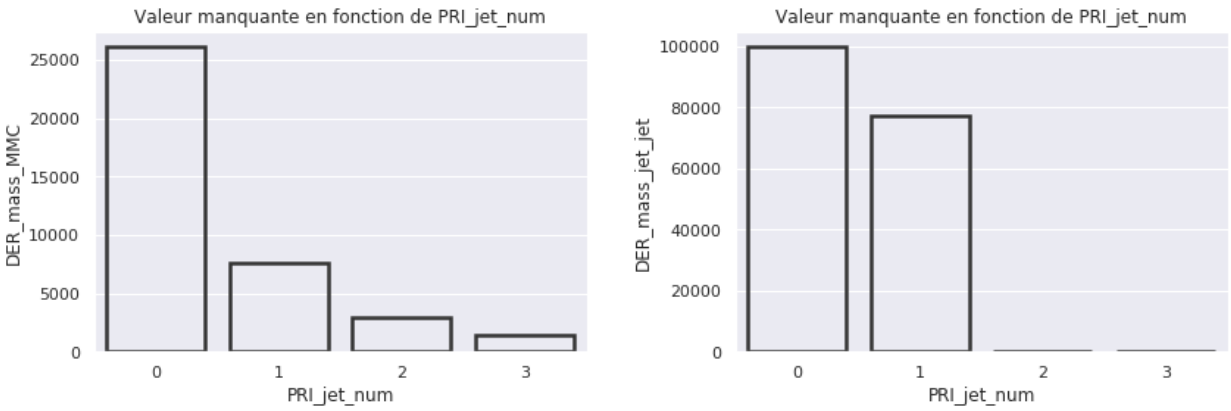

FIGURE 1 – Distribution de valeurs manquantes pour DER_mass_MMC et DER_mass_jet_jet

2.2 Distribution des données

Ensuite, j'ai effectué une analyse descriptive des données. J'ai tracé la distribution de chaque feature en fonction des catégories s et b. On observe que s et b peuvent être séparées légèrement par leur répartition des valeurs pour certaines features comme DER_mass_transverse_met_lep et DER_mass_MMC. Ce n'est pas le cas pour la majorité des features, par exemple, pour PRI_lep_pt et PRI_met_phi, la distribution de s et b semble identique (Figure 2).

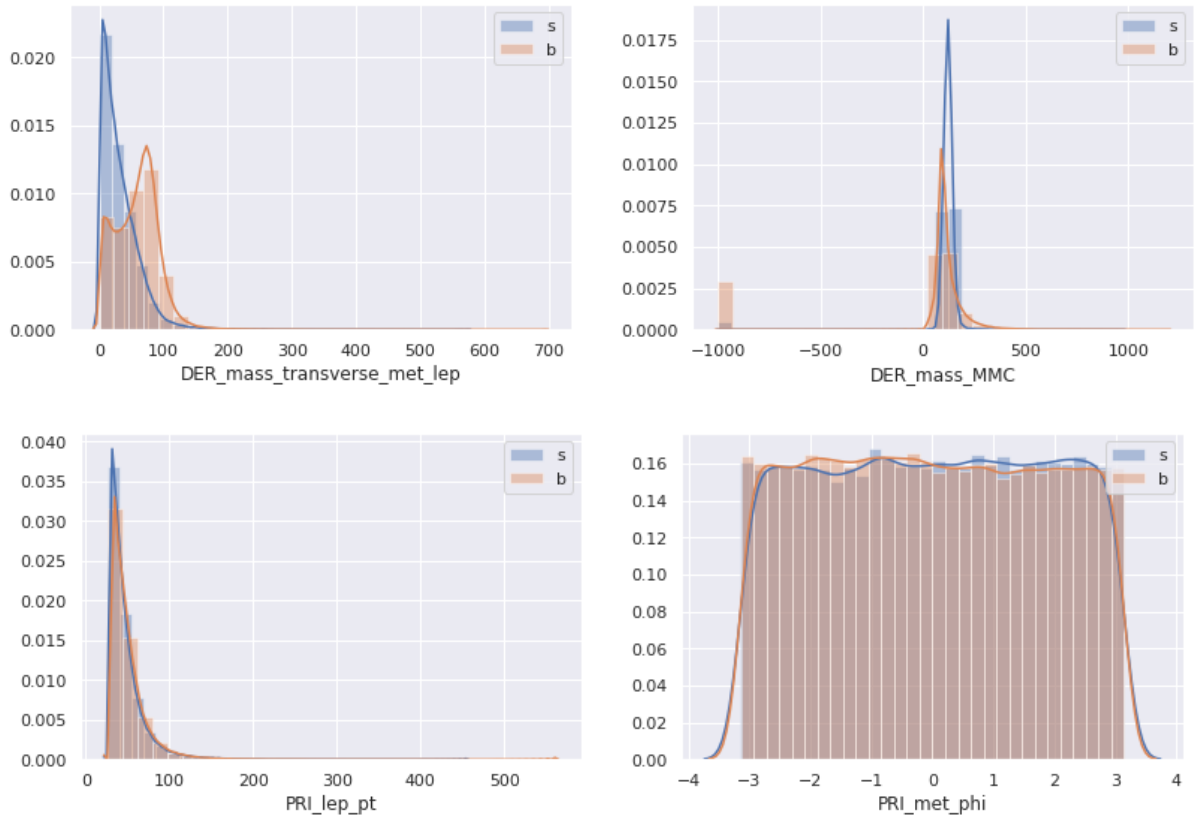

FIGURE 2 – Distribution des valeurs des features pour s et b

2.3 Outliers

L'étape suivante a été de repérer des valeurs aberrantes ou extrêmes. Des boxplots ont été générés sur différentes features pour visualiser l'étendue des valeurs.

On observe plusieurs allures de boxplot. Par exemple,

- Les boxplots des variables DER_mass_MMC, DER_mass_vis, DER_pt_ht, DER_pt_tot, DER_sum_pt, DER_pt_ratio_lep_tau (et quelques autres variables) ont la même allure. Ces variables présentent des médianes proches de 100 avec un faible écart inter-quartile. Les valeurs extrêmes sont entre 0 et 200 approximativement. Cependant, on observe un nombre important de outliers. Sur la Figure 3 à gauche, des valeurs atypiques de 1200 sont présentes.
- Les boxplots de ces variables PRI_tau_eta, PRI_tau_phi, PRI_lep_eta, PRI_lep_phi ont la même allure. Les valeurs de ces variables s'étendent de -3 à 3 avec une médiane de 0 approximativement. Il ne semble pas y avoir de outliers pour ces variables, comme suggère la Figure 4 à droite.

Un exemple de chacunes de ces deux allures est illustré sur la Figure 3 avec un boxplot de la variable DER_mass_MMC et un boxplot de la variable PRI_tau_eta.

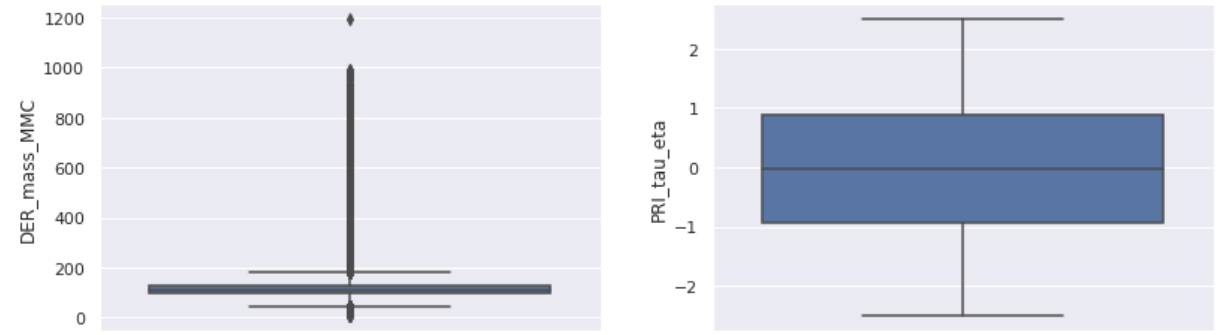

FIGURE 3 – Exemple de boxplot pour DER_mass_MMC et PRI_tau_eta

2.4 Analyse en composantes principales

J'ai utilisé l'ACP centré afin d'obtenir un résumé le plus pertinent possible des données. Après avoir calculé les 2 premiers axes principaux, j'ai projeté 9000 observations dans l'espace de dimension 2 que forment ces axes.

Pour la Figure 4 à gauche, j'ai indiqué en bleu (resp. en orange) les observations correspondant à un signal (resp. un background). On observe que 3 clusters se dégagent de cette représentation. Les classes s et b sont présentes au sein de chaque cluster. Au sein d'un même cluster (à gauche de la Figure 4), on observe que les observations background et signal sont confondues.

Sur la Figure 4 à droite, j'ai indiqué les observations en fonction de la variable jet_num. On observe que les clusters qui se dégagent de l'ACP correspondent aux clusters formés par les nombres des jets. C'est une variable assez importante pour la caractérisation d'une observation, car elle explique la variance de nos données, mais elle n'est pas suffisante pour expliquer les variables cibles.

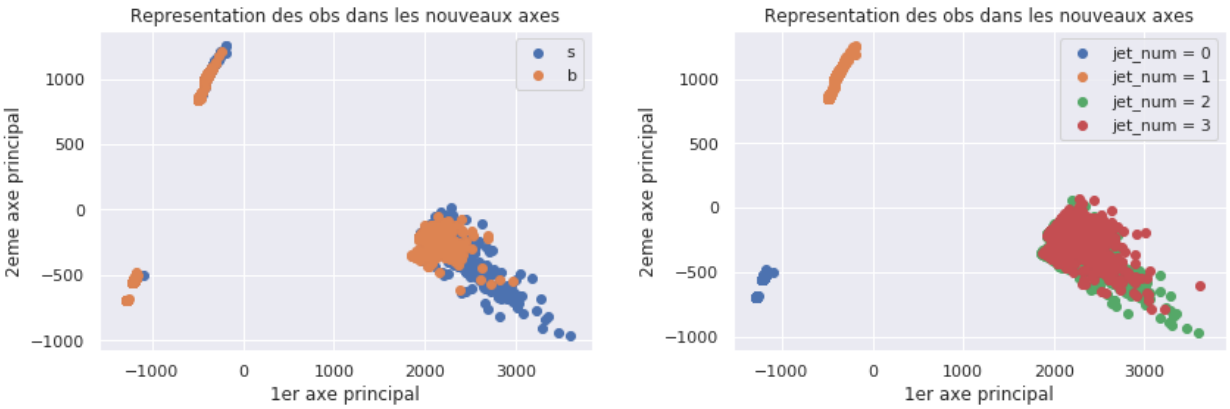

FIGURE 4 – Projection des observations sur le plan principal

Sur le cercle de corrélation, seulement 11 variables (1/3 des variables) sont bien représentées (proche de 1 sur le cercle). Les variables les mieux représentées sont toutes des variables décrivant la nature du jet. Par exemple, il s'agit de : DER_deltaeta_jet_jet, DER_mass_jet_jet, PRI_jet_num, PRI_jet_leading_pt, PRI_jet_leading_eta ...

On observe que à partir de 3 axes principaux, 99% de la variance des données est expliquée. L'interprétation qui peut être faite est que cela coïncide parfaitement avec le nombre de clusters visualisés sur la Figure 4, et donc correspond au nombre de jets.

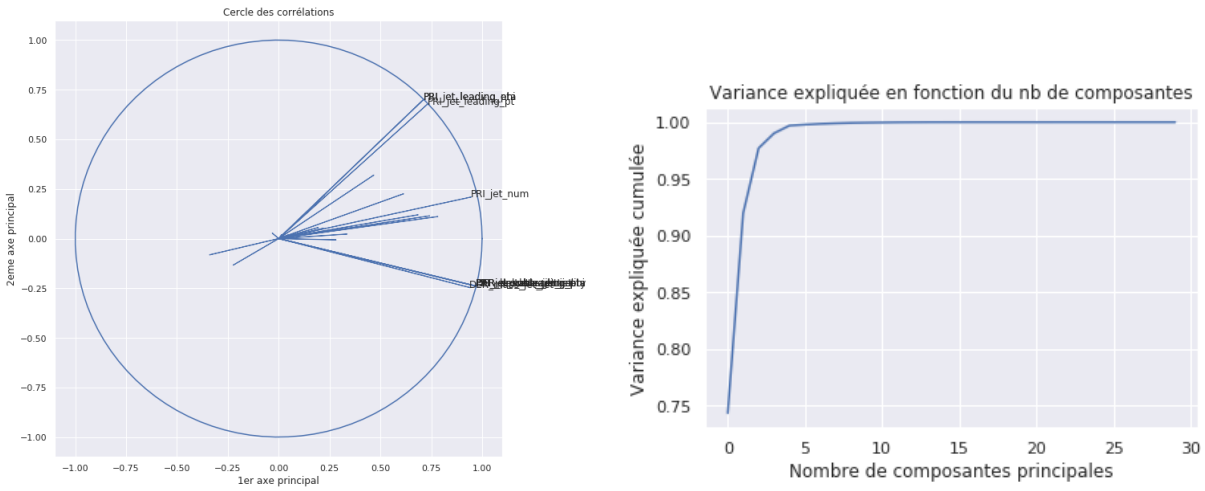

FIGURE 5 – Projection des observations sur le plan principal et variance expliquée par les composantes

2.5 Matrice de corrélation

Sur la matrice de corrélation, on observe des zones de couleurs vives où les variables sont fortement corrélées (positivement ou négativement). Certaines n'ont aucune corrélation linéaire comme indiquées par les zones claires.

Du à la forte corrélation de certaines variables, il peut être intéressant de retirer une dizaine de features pour se retrouver à 20 features, pour ainsi simplifier nos premiers modèles.

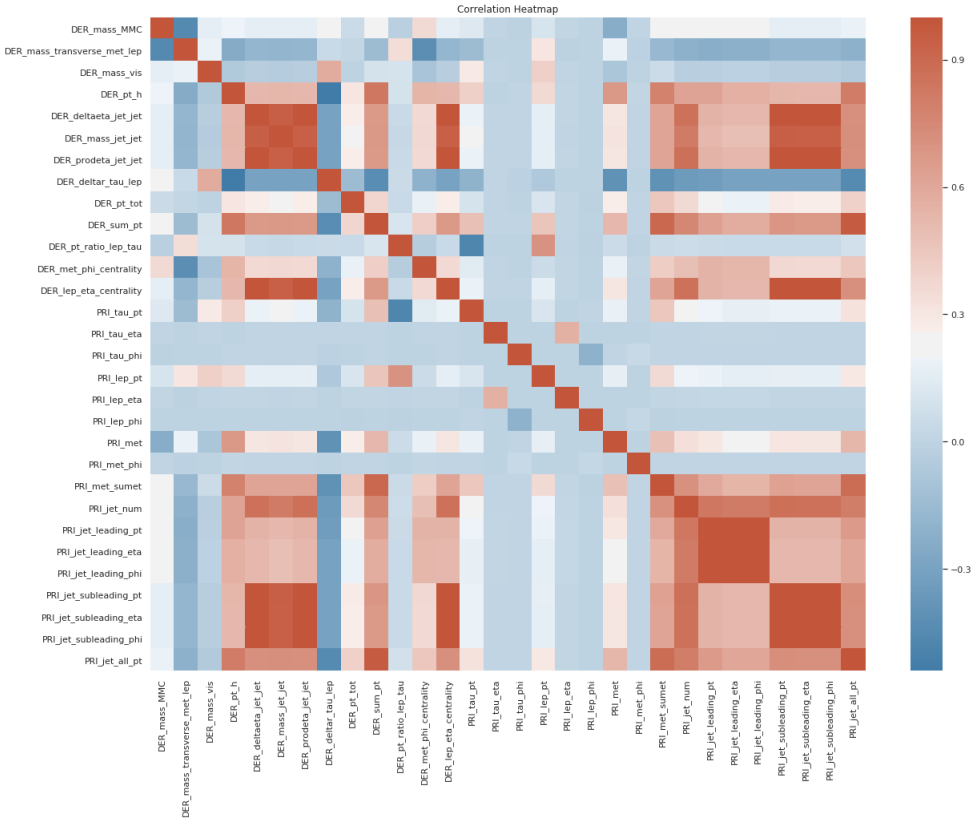

FIGURE 6 – Matrice de corrélation des variables

3 Prétraitements des données

Le jeu de donnée étant déséquilibré, je me suis posé la question sur la nécessité de fabriquer un dataset équilibré. Par exemple, il est possible sous échantillonner les évènements background ou sur échantillonner les évènement signal, afin d'obtenir un jeu de donnée équilibré.

Pour le projet, je me suis limité au jeu de donnée de base. Certaines méthodes de la bibliothèque scikit learn permettent d'attribuer un poids lors de l'apprentissage à chaque observation. J'ai utilisé les poids du jeu de donnée pour compenser le déséquilibre.

3.1 Complétion des valeurs manquantes

De nombreuses valeurs sont manquantes, on ne peut pas se permettre de supprimer les observations ou les variables incriminées. J'ai complété par la médiane car la moyenne est sensible aux valeurs extrêmes des outliers. La gestion des valeurs manquantes a permis d'améliorer le gradient boosting tree mais pas les autres modèles.

3.2 Gestion des outliers

Nous n'avons pas les connaissances physiques nécessaires pour savoir si les valeurs extrêmes sont plausibles ou aberrantes. J'ai tout de même traité ces outliers en supprimant 2500 observations lorsqu'elles présentaient une valeur correspondant aux 0.01 % valeurs les plus élevées.

3.3 Feature Engineering

La distribution de certaines variable ne semble pas symétrique, j'ai essayé des transformations non linéaires comme le logarithme pour les variables non négatives. Le but étant d'améliorer la symétrie de notre distribution de données.

3.4 Feature Scaling

Comme les valeurs des features ne sont pas sur la même échelle, j'ai réduit et centré les valeurs. Pour la régression logistique, cela a permis au modèle de converger plus rapidement et d'être plus robuste. Pour les arbres de décisions et random forest, cela n'a pas eu beaucoup d'impact.

3.5 Suppression de certaines colonnes

Les features ne semblent pas toutes utiles pour expliquer la variable cible, comme suggère la variable PRI_met_phi sur la Figure 2. Certaines variables sont fortement corrélées, j'ai envisagé des modèles simples au début en retirant 9 variables. Pour les algorithmes utilisant des arbres de classification, j'ai décidé de les laisser car ils apportaient un meilleur résultat.

3.6 Séparation des données

Le jeu d'apprentissage de Kaggle a été séparé en deux parties : un pour apprendre et un pour évaluer les hyperparamètres (Split 90-10). J'ai donc utilisé les poids "Kaggle weigths" qui sont normalisées pour chacun des dataset Kaggle pour calculer le AMS. Lors du split en train et validation, j'ai divisé les poids du train par 0.9 et du validation par 0.1, afin d'obtenir une meilleur approximation du AMS sur le test set.

J'ai effectué la cross validation seulement pour certains modèles. Lorsque c'est le cas, j'utilise le 5-fold validation pour valider les hyperparamètres.

Le jeu de test correspond au jeu du Leaderboard public de Kaggle.

4 Méthodes et résultats

4.1 Modèles simples

Régression logistique utilisant une seule colonne

La première chose à faire est de tester des modèles très simples. Pour cela, j'ai entraîné une régression logistique sur une seule variable, pour chacune des variables, sans prétraitement des données. J'ai observé des résultats d'AMS très faible (inférieur à 1) pour la majorité des variables. Seulement 5 variables ont obtenu un AMS supérieur à 1 et sont présentées sur la Figure 7. On observe que la régression logistique en utilisant seulement la variable DER_mass_transverse_met_lep obtient le meilleur AMS sur le validation set à 1,74. La performance du modèle évalué sur le test set est : **1,77 AMS**. On observe que ce modèle simple se généralise très bien aux nouvelles données.

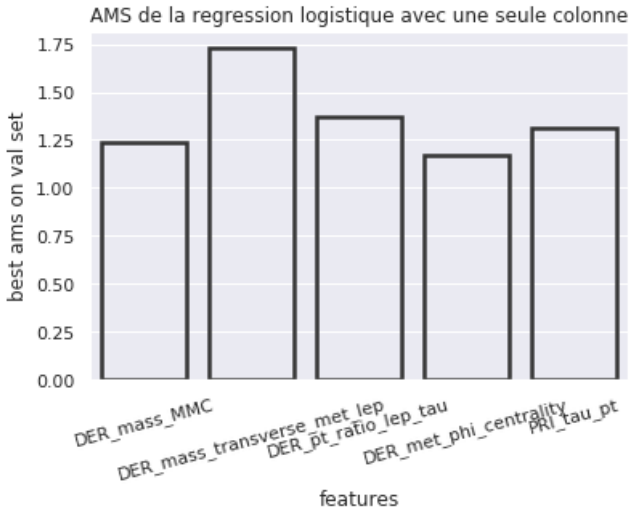

FIGURE 7 – Performance de la régression logistique utilisant une seule colonne en utilisant un threshold = 0.8

Arbre de décision

Un autre modèle très simple est l'arbre de décision. Sur la Figure 8, j'ai affiché les règles des premiers noeuds. On observe que les variables qui partitionnent le mieux les classes s et b sont les mêmes que les variables qui obtiennent un bon AMS pour l'expérience précédente : DER_mass_transverse_met_lep, DER_mass_MMC et DER_met_phi_centrality.

On peut conclure que ce sont les variables les plus importantes pour notre modèle pour expliquer la variable cible.

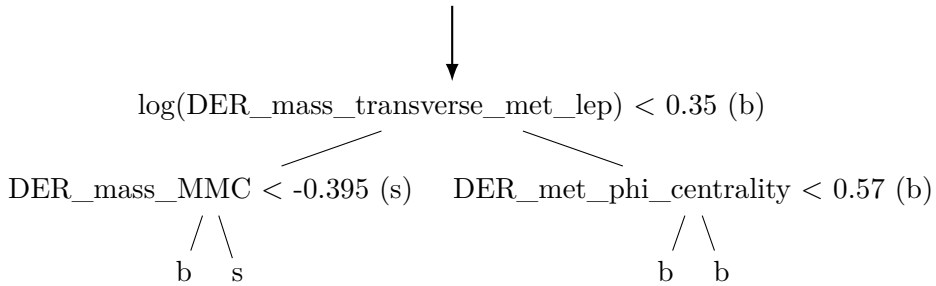

FIGURE 8 – Arbre de décision (premières règles)

C'est un algorithme qui peut facilement souffrir de sur-apprentissage. En effectuant une validation sur le hyperparamètre de la profondeur maximale de l'arbre et le threshold, on observe qu'on arrive à obtenir un AMS de 2,8 sur le validation set.

Sur la Figure 9 à gauche, on observe les scores AMS avec un threshold de 80 lorsqu'on fait varié la profondeur. Pour le validation de la profondeur, on observe sur la Figure 9 deux zones.

— une partie où le biais est fort et la variance est faible, suggéré par le faible gap entre la courbe du train et de la validation. C'est le phénomène de sous-apprentissage.

— une partie où la variance est forte, mais le biais est fort. Le modèle a simplement mémorisé les données et ne se généralise pas bien. C'est le phénomène de sur-apprentissage.

Nous avons pris une profondeur maximale = 8 qui est un bon compromis biais-variance.

Une fois ce paramètre fixé, nous avons optimisé le threshold pour la probabilité d'appartenance. Sur scikit learn, pour les arbres de décision, cette probabilité correspond à la fréquence d'apparition de la classe dans la feuille prédite. Le meilleur score est obtenu pour threshold = 85, qui ne semble pas très robuste car le score chute sévèrement après cette valeur. J'ai donc préféré fixé la valeur threshold = 80 qui devrait mieux se généraliser à de nouvelles données.

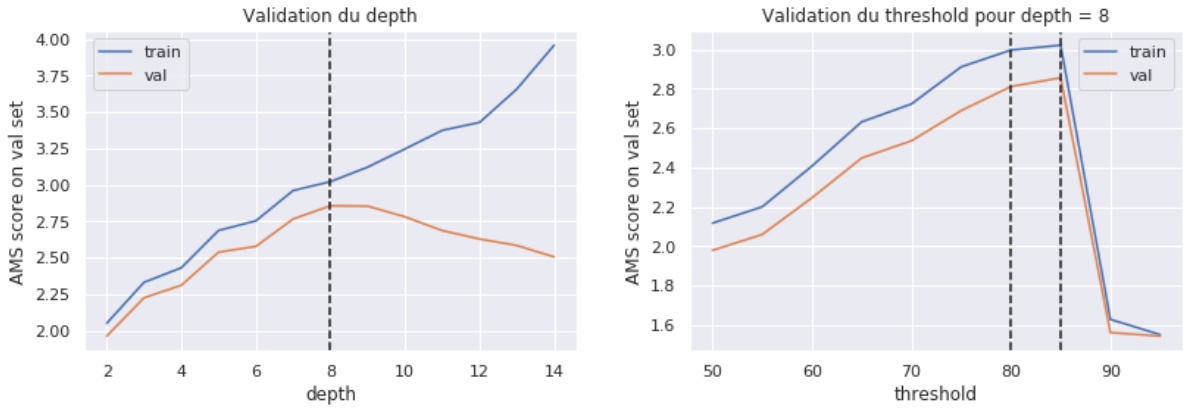

FIGURE 9 – Validation des paramètres sur le validation set pour l'arbre de décision

Une fois qu'on fixe les meilleurs hyper-paramètres trouvés sur le validation set, la performance du modèle évaluée sur le test set est : **2.26 AMS**, qui est meilleur que la simple régression logistique.

4.2 Régression logistique

Pour la régression logistique sur les données nettoyées, j'ai essayé de faire varier le terme de régularisation L2 sur une échelle logarithmique entre 10^{-5} et 10^{3}. Le terme de pénalité intéressant était 0.1, car il maximise le score sur le jeu de validation. J'ai fixé cette valeur, puis validé la valeur du threshold à 75. On observe que le modèle n'est pas assez complexe, mais il se généralise très bien sur le jeu de données de validation, on obtient : 2.13 AMS. Sur le test set, on obtient : **1.97 AMS**.

Les scores obtenus sur le test set sont nettement inférieurs aux scores de validation pour l'arbre de décision et la régression logistique. L'étude de ces méthodes simples a donc suggéré la nécessité de techniques plus robustes afin de valider nos modèles et nos hyper paramètres comme **la cross-validation**.

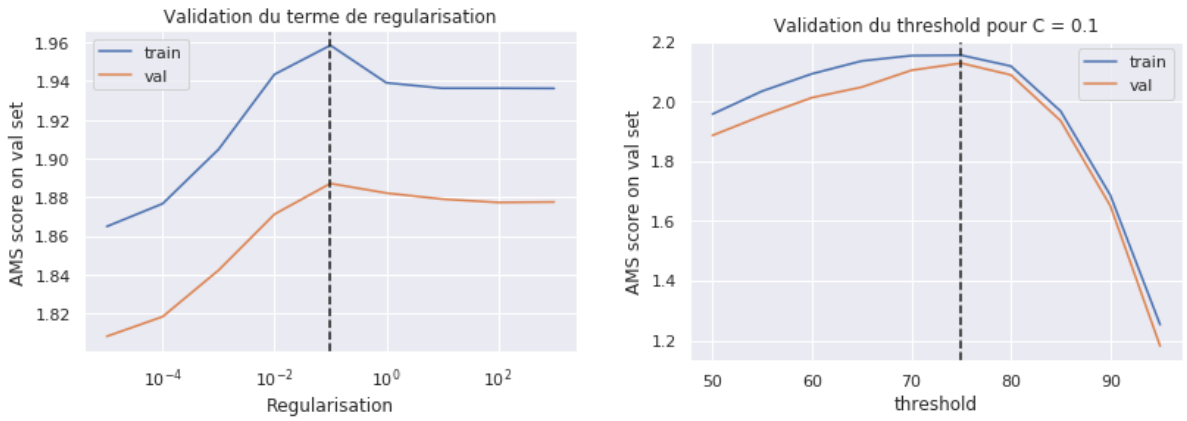

FIGURE 10 – Validation des paramètres sur le validation set pour la régression logistique

4.3 Random Forest

Par cross validation sur 5 folds, j'ai fait varié le nombre d'arbres n pour $n \in \{10, 100, 500, 1000\}$ et affiché mes scores obtenus sur les différents folds. Sur la Figure 11, on observe que plus on a d'arbres, plus le modèle sur-apprend les données. Donc la valeur de $n = 10$ arbres a été retenu pour pouvoir obtenir un modèle qui se généralise bien.

Ensuite, j'ai cherché une valeur de profondeur maximal d dans $d \in \{5, 10, 15, 20\}$. On observe que clairement le modèle avec une profondeur de 5 pour les arbres est trop simple. En revanche pour 15 et 20, le modèle est trop complexe : on observe que le AMS de validation chute considérablement. Finalement, la valeur de $d = 10$ profondeur maximal a été retenu qui constitue un bon trade off biais-variance.

FIGURE 11 – Cross-Validation des paramètres pour la random forest avec threshold = 80

Le score obtenu avec les meilleurs hyper paramètres sur le test set est : **2.54** AMS qui est une légère amélioration comparée aux arbres de classification.

Sur scikit learn, on peut afficher les variables les plus importantes pour la random forest, il s'agit des variables suivantes : 'DER_mass_MMC', 'DER_mass_transverse_met_lep', 'DER_mass_vis'. On retrouve donc les mêmes variables importantes de masse que les expériences précédentes.

4.4 Gradient Boosting Tree

Pour le gradient boosting tree, j'ai également effectué une cross validation sur le nombre d'arbres à utiliser. Sur la Figure 12, on observe que le modèle est robuste, contrairement au random forest qui souffre de sur-apprentissage. En effet, lorsque le nombre d'arbres augmente, le gap entre le score de train et validation reste proche et la performance augmente. J'ai donc opté pour un modèle à 200 arbres, mais il est possible d'en prendre plus sans trop souffrir de sur-apprentissage à mon avis.

On observe que le modèle est très sensible à la profondeur des arbres, à partir de 7 le modèle sur-apprend, comme suggère le grand écart entre le score sur le train et le validation. En fixant les meilleurs hyper-paramètres, (200 estimateurs et 5 de profondeurs) on observe que le gradient boosting est plus séduisant que les précédents modèles puisqu'on obtient un score beaucoup plus élevé : **3.35 AMS** sur le test set.

FIGURE 12 – Cross-Validation des paramètres pour le gradient boosting avec threshold = 80

4.5 Réseau de neurones

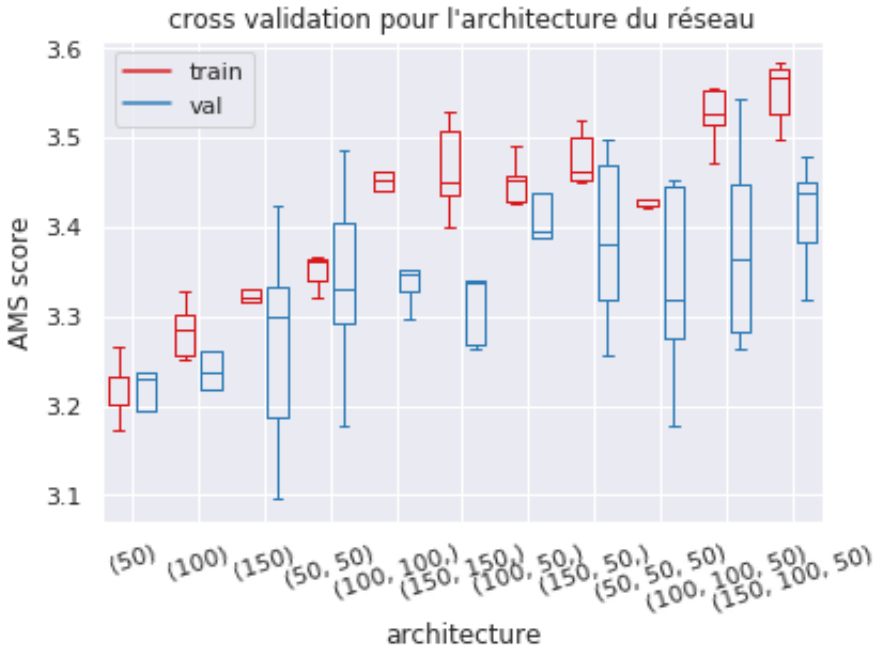

FIGURE 13 – Expérimentation de plusieurs architectures de réseaux avec threshold = 80

J'ai essayé plusieurs architectures sur 5-folds. Les scores obtenus sur le train et le validation sont affichés sur la Figure 13. Nous observons que plus la taille du réseau de neurones est grand, plus on arrive à apprendre les patterns sur le train. Ce n'est pas le cas pour la généralisation sur le validation set. On observe que le score sur le validation set varie considérablement d'un fold à un autre, puisque les boxplots bleus sont très étendus.

Le réseau de neurones le plus robuste semble être celui à 2 couches avec 100 et 50 unités cachées, car le score de validation ne varie pas énormément.

J'ai fixé cette architecture et j'ai ensuite testé une validation des hyper paramètres. Avec le nombre d'epochs fixé à 7, on trouve : **3.35 AMS** sur le test set, un score robuste comme le Gradient Boosting Tree.

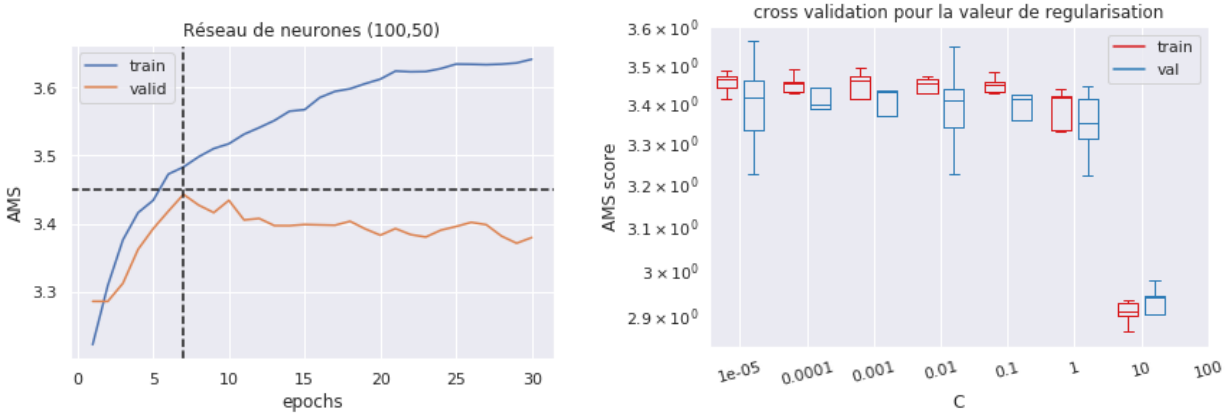

FIGURE 14 – Validation des paramètres pour les réseaux de neurones avec threshold = 80

5 Conclusion

Après avoir étudier les données, nous avons compris l'importance de traiter les données. J'ai essayé une transformation log afin d'étaler la distribution de certaines distributions. D'autres transformations non-linéaires peuvent être envisagées comme la transformation racine carrée. Il a été difficile de trouver des transformations pertinentes sur les données, étant donné notre manque de connaissance en physique. En regardant la documentation, il existe des features crées par des physiciens comme les features Cake. Comme nous avons vu que le feature engineering avait beaucoup d'impact sur la performance, il serait intéressant de pouvoir tester l'impact de ces features.

Nous avons essayé des modèles simples sur nos algorithmes, cela nous a permis de comprendre les features les plus importantes qui sont DER_mass_MMC et DER_mass_transverse_met_lep. Ensuite, nous avons testé des modèles plus complexes. Les modèles intéressants qui se dégagent de nos expériences est le gradient boosting et les réseaux de neurones.

Utiliser un threshold au dessus de 50 était important pour optimiser le AMS. Renormaliser les weights en fonction de la taille du jeu de donnée nous a permis d'obtenir des AMS de confiance.

Nous avons observé l'importance de la technique de cross validation sur nos données. Il était très simple de overfit notre jeu de donnée, par exemple utilisant un threshold trop optimisé sur nos données de validation par exemple sur la Figure 9. La cross validation a été un bon moyen pour être confiant sur nos hyper-paramètres.

Les axes de développement pour obtenir un meilleur modèle seraient de trouver et de tester des meilleurs transformations sur nos données, tester d'autre méthodes de complétion de valeurs manquantes. Ensuite, j'aurai essayé d'optimiser le réseau de neurones avec les paramètres de régularisation, dropout ou d'autres algorithmes comme le SVM et des ensembles de réseaux de neurones.

Références

[1] Données : http://opendata.cern.ch/record/328

[2] Baseline : https://higgsml.lal.in2p3.fr/software/starting-kit/

[3] Discussion des meilleurs stratégies : https://www.kaggle.com/c/higgs-boson/discussion/10344

[4] Article sur le challenge : http://proceedings.mlr.press/v42/cowa14.pdf