FDMS - Analyse de données sur le boson de Higgs

Alexia Bourmaud et Louise Marchal

Table des matières

Table des illustrations

TP1 - Analyse préliminaire des données

La Data science est de plus en plus utilisée dans des domaines très variés tels que la physique. Grâce à des algorithmes de classification, il est possible de détecter si une particule observée est un boson de Higgs. Une base de données a été mise à disposition dans le cadre d'un concours Kaggle. Elle est accessible sur le site *Open Data*.

1. Observations visuelles

La base de données est composée de 818238 exemples caractérisés par 35 attributs. Parmi ces attributs, il y a le label qui indique l'étiquette de l'exemple. On observe également que les attributs « KaggleSet » et « KaggleWeight » n'ont pas de rapport avec les valeurs observées par ATLAS mais plutôt avec le concours en lui-même, nous les avons donc retirés. De plus l'attribut « EventId » représente le numéro donné à chaque exemple, il n'est pas utile pour l'étude.

2. Etude des données

La base de données est déséquilibrée, elle possède 34% d'exemples étiquetés bosons de Higgs et 66% étiquetés arrière-plan. Il faudra donc, lors de la création de la base d'apprentissage, équilibrer les classes afin de ne pas biaiser l'apprentissage.

Afin d'étudier pour chaque attribut la répartition des données, nous avons réalisé un histogramme par attribut.

Figure 1 : Histogramme des différents attributs de la base

On observe que certains attributs ont des valeurs discrètes comme par exemple l'attribut « PRI_jet_num », d'autres sont continues comme « PRI_tau_eta ». D'autres encore semble avoir des valeurs discrètes mais lorsque l'on l'observe de plus près on remarque qu'elles sont continues. De plus certains attributs ont un pic de valeurs extrêmes à -999. Ce sont des valeurs manquantes. En regardant de plus près on observe que 40% des exemples ont trois attributs qui ont des valeurs non renseignées et 30% qui en ont dix. Il faudra modifier ces valeurs lors de l'apprentissage extrêmes pour ne pas qu'elles influent sur les résultats.

Afin de déterminer les attributs les plus pertinents pour savoir si la particule est un boson de Higgs ou non, nous avons entrainé un arbre de décision sur nos données. Pour ce faire, nous avons séparés les données de leurs labels. Les attributs les plus hauts dans l'arbre peuvent être considérés comme les plus influents.

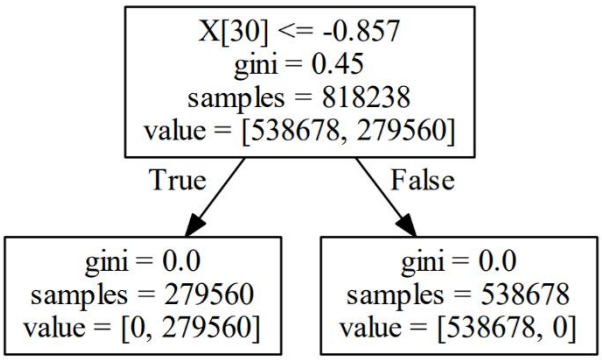


Figure 2 : Arbre de décision

On remarque que l'attribut « Weight » permet à lui seul de déterminer si la particule est un boson de Higgs ou non. En observant plus précisément la documentation sur le projet, on remarque que la variable « Weight » correspond au poids de l'évènement. Cette variable est utilisée pour rééquilibrer la base lors du calcul de l'AMS car l'évènement « boson de Higgs » est beaucoup moins représenté que l'évènement « background ». L'AMS (Approximate Median Significance) est un score permettant de pénaliser les faux positifs.

On a donc décidé de supprimer cet attribut de la base de données et réalisé de nouveaux arbres de décisions.

Nous avons réalisé plusieurs tests en modifiant la profondeur maximale de l'arbre. En effet, en faisant diminuer la profondeur maximale on diminue le nombre d'attributs pertinents afin de réduire le temps de calcul mais en réduisant le nombre d'attributs on supprime de l'information et ainsi on diminue les possibilités d'avoir un bon score. C'est-à-dire qu'en fonction de la précision exigée par le client on pourra plus ou moins réduire le nombre d'attributs utilisés pour le modèle.

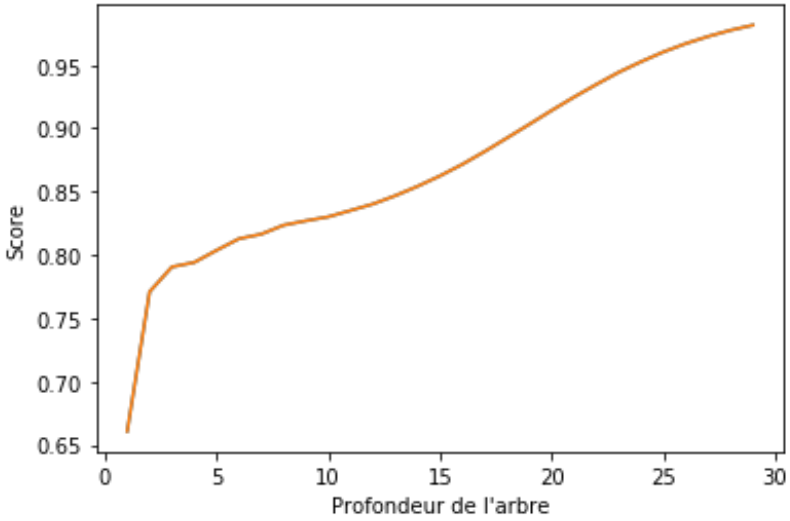


Figure 3 : Score de bonne classification en apprentissage en fonction de la profondeur de l'arbre de décision

On voit que pour une profondeur de 1, le score est de 66% de bonnes classifications, cela correspond à la proportion d'exemples étiquetés arrière-plan. On en déduit que ce premier modèle prédit la classe majoritaire.

Par soucis de lisibilité nous nous sommes penchées sur l'arbre de profondeur maximum quatre.

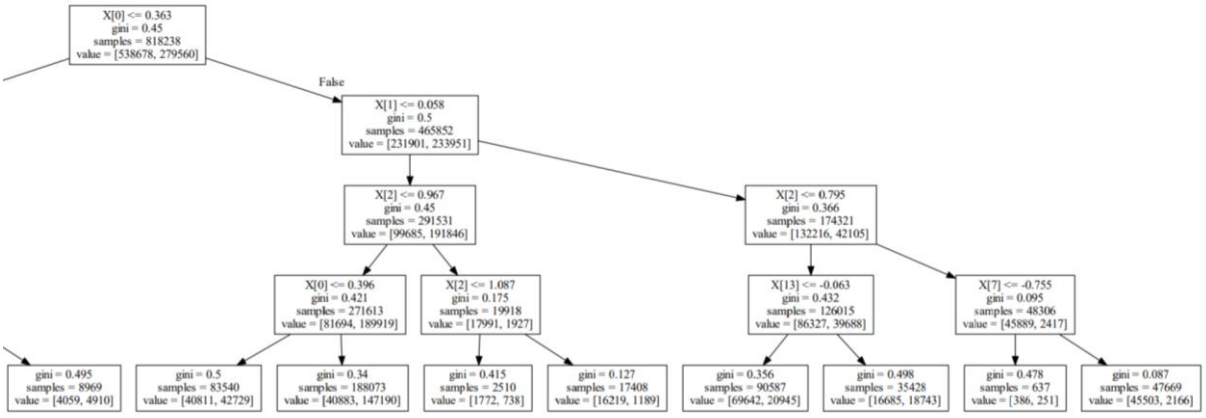


Figure 4 : Moitié de l'arbre de décision de profondeur maximum quatre

Cet arbre nous permet de mettre en évidence les six attributs les plus utiles (« DER_mass_MMC », « DER_mass_transverse_met_lep », « DER_mass_vis », « DER_delta_tau_lep », « DER_met_phi_centraly », « PRI_tau_pt »). Pour cette profondeur d'arbre le score de bonnes classifications est de 79%.

Afin de bien déterminer les attributs qui sont redondants, nous avons fini par faire une matrice de corrélation.

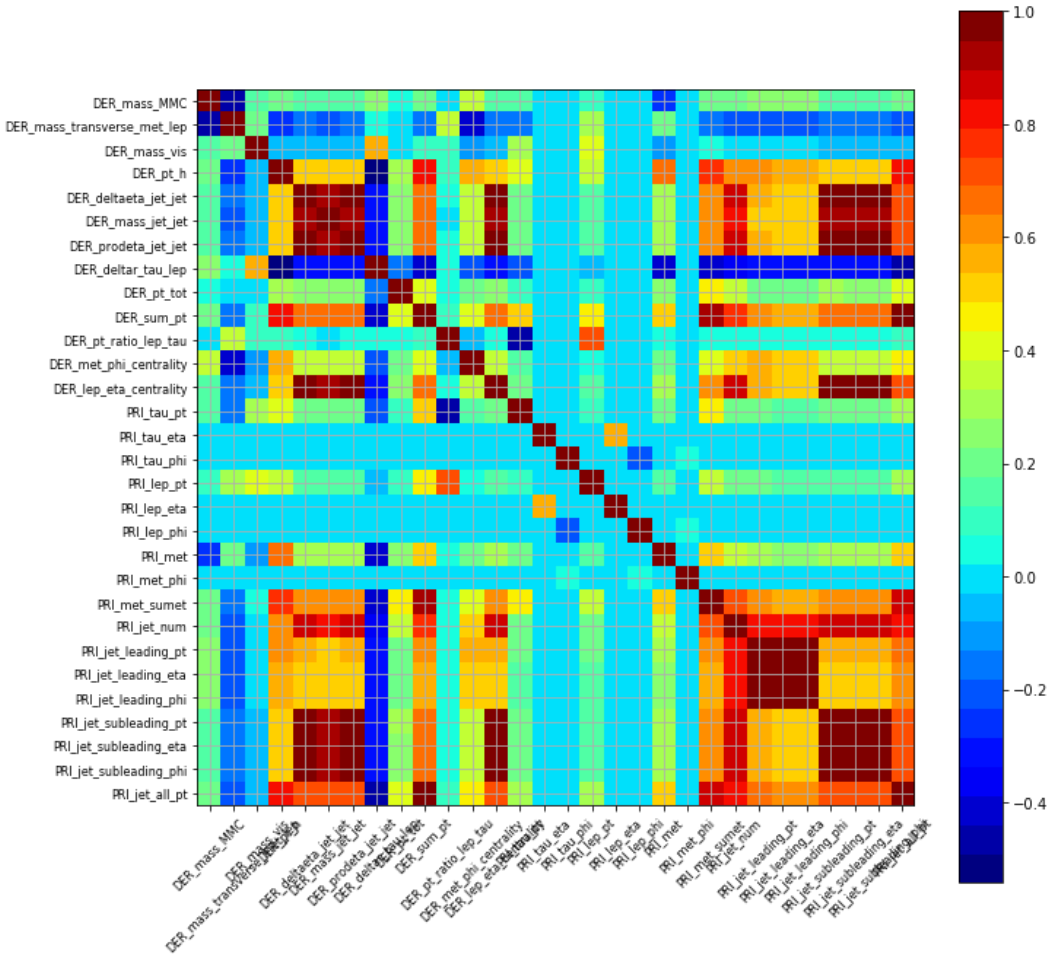


Figure 5 : Matrice de corrélation

On voit que certains attributs sont fortement corrélés comme « PRI_jet_subleading_eta » et « PRI_jet_subleading_pt ». Il faudra décider d'un seuil de corrélation au-dessus duquel les attributs sont redondants, il faudra ensuite supprimer l'un des attributs corrélés. Plus ce seuil est faible plus il y aura d'informations perdues mais les calculs seront plus rapides.

3. Conclusion – Ouverture

Cette analyse préliminaire nous a permis de mettre en évidence que la proportion de particule de bosons de Higgs est nettement inférieure à celle des particules d'arrière-plan, il faudra donc, lors de l'apprentissage, équilibrer notre jeu de données afin de ne pas biaiser le modèle. D'autre part, nous avons remarqué la présence de valeurs non renseignées qu'il faudra traiter avec précaution lors de l'apprentissage de nos modèles. Enfin grâce aux arbres de décisions et à la matrice de corrélation nous pourrons sélectionner les attributs les plus pertinents en fonction du score souhaité par le client.

TP2 – Analyses secondaires – Scores de références

A l'aide de la matrice de corrélation réalisée dans la partie précédente nous avons pu mettre en évidence des attributs fortement corrélés. Pour chaque couple d'attributs fortement corrélés, c'est-à-dire avec une corrélation supérieure à 99%, nous avons supprimé l'un des deux attributs afin de simplifier notre base de données. Cela nous permet de conserver seulement 23 attributs sur 30.

1. Baselines

Dans cette partie, nous avons appliqué quelques classifieurs naïfs sur nos données afin d'obtenir des scores de bonnes classifications de références afin de pouvoir comparer les prochains modèles plus sophistiqués.

Pour faire apparaître ces scores, nous avons entraîné nos différents classifieurs sur une base d'apprentissage comportant 90% des exemples de la base de données initiale tirés aléatoirement. Les 10% d'exemples restant composent la base de test. Cette base permet de tester l'efficacité de nos modèles en utilisant des exemples que nos modèles n'ont jamais rencontrés.

Les classifieurs naïfs sont dit naïfs car ils n'apprennent pas réellement, par exemple le classifieurs aléatoire prédit chacune des classes de façon arbitraire. Le classifieur majoritaire prédit toujours la classe majoritaire dans l'ensemble d'apprentissage. Le classifieur stratégie génère des prédictions en respectant la distribution des labels de l'ensemble d'apprentissage. Dans notre cas cela signifie que dans 65% des cas il prédit la classe « non boson » et à 35% la classe « boson ».

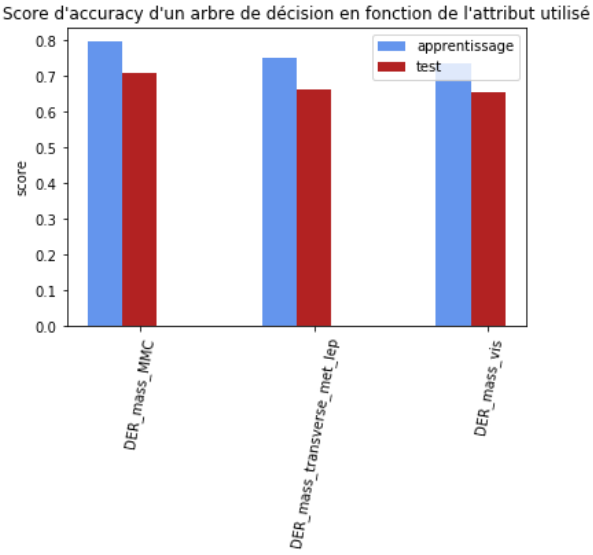
	Score app	Score test
aléatoire	0.499785	0.497311
classe majoritaire	0.658585	0.656128
stratégie	0.550435	0.548225

Figure 6 : Score en apprentissage et en test avec des classifieurs naïfs

Le classifieur classe majoritaire obtient un score de 65% de bonne classifieur, ce score correspond au pourcentage de label « non boson de Higgs ». Donc notre modèle final devra au moins faire 65% de bonnes classifications pour le considérer efficace.

2. Scores de références en utilisant un seul attribut

Dans le but d'obtenir des scores de références plus portés sur les attributs, nous avons mis en place plusieurs classifieurs simples, c'est-à-dire que nous avons gardé les paramètres par défaut présent dans la bibliothèque scikit-learn. Dans la partie 2 du tp 1, nous avions mis en évidence les attributs qui avaient le plus d'influence dans l'apprentissage. Nous avons donc appris un arbre de décision sur un seul de ces attributs à la fois pour pouvoir comparer leur importance dans l'apprentissage.



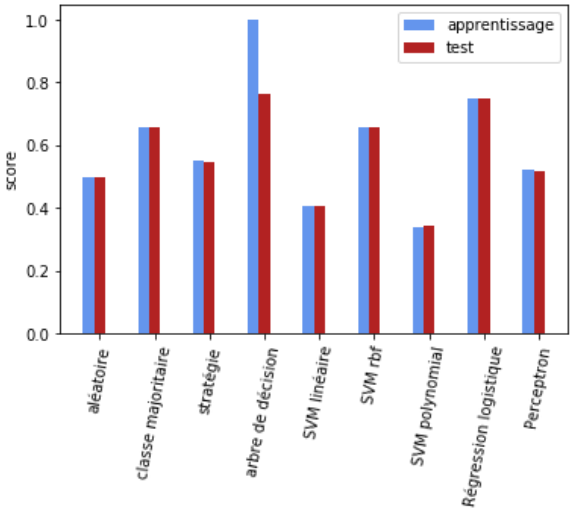
	accuracy	validation
DER_mass_MMC	0.798219	0.710941
DER_mass_transverse_met_lep	0.751117	0.662935
DER_mass_vis	0.736059	0.656128

Figure 7 : Score d'accuracy d'un arbre de décision en fonction de l'attribut utilisé

On observe que l'attribut "DER_mass_MMC" est l'attribut qui permet d'obtenir les meilleurs résultats. Le score de bonnes prédictions en validation est de 71%. Cela qui signifie que plus tard si on augmente le nombre d'attributs sur lequel le modèle apprend, il faudra obtenir un score supérieur à 71% sinon cela n'aura aucun intérêt d'augmenter le nombre d'attributs.

3. Scores de références sur 23 attributs

Nous avons mis en place plusieurs classifieurs simples, c'est-à-dire que nous avons gardé les paramètres par défaut présent dans la bibliothèque scikit-learn. L'étude des paramètres n'est pas nécessaire car nous cherchons à mettre en évidence un score minimum à obtenir lors de la conception du modèle d'apprentissage final.



	Score app	Score test
aléatoire	0.499785	0.497311
classe majoritaire	0.658585	0.656128
stratégie	0.550435	0.548225
arbre de décision	1.000000	0.761378
SVM linéaire	0.407810	0.407985
SVM rbf	0.658856	0.655248
SVM polynomial	0.339560	0.341587
Régression logistique	0.749191	0.747055
Perceptron	0.519852	0.519518

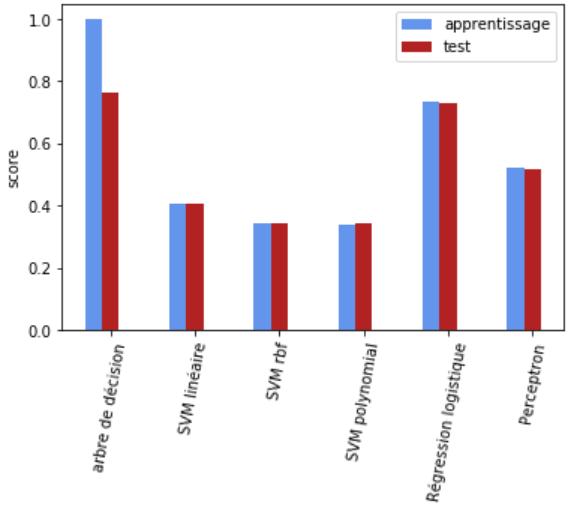
Figure 8 : Score en apprentissage et en test de différents classifieurs - 23 attributs

Sur les résultats ci-dessous, on peut observer les scores en apprentissage et en test sur une base de données déséquilibrée sans les attributs redondants trouvés grâce à la matrice de corrélation.

On remarque que le SVM linéaire, le SVM polynomial et le Perceptron sont en dessous du classifieur majoritaire (65%) et de l'arbre de décision sur un seul attribut (70%). On peut donc considérer qu'avec les paramètres par défauts, ils ne sont pas suffisamment efficaces.

L'arbre de décision et la régression logistique obtiennent des scores supérieurs aux scores de références obtenus dans les parties 1 et 2, donc ils pourront être pris comme nouveaux scores de références.

Suite à ces expériences, il nous a semblé important de réaliser les mêmes expériences mais cette fois sur une base d'apprentissage équilibrée. C'est-à-dire que l'apprentissage s'effectuera sur une base de données qui contiendra autant d'exemples étiquetés "bosons de Higgs" que d'exemples étiqueté "non boson de Higgs". Le test, quant à lui, s'effectuera sur une base de données déséquilibrée puisqu'en pratique il est rare d'observer un boson de Higgs.



	Score app	Score test
arbre de décision	1.000000	0.765876
SVM linéaire	0.407810	0.407985
SVM rbf	0.342083	0.343872
SVM polynomial	0.339560	0.341587
Régression logistique	0.735289	0.732108
Perceptron	0.519852	0.519518

Figure 9 : Score de bonnes classifications en apprentissage et en test sur données équilibrées de différents classifieurs avec 23 attributs

Il est intéressant de voir que l'équilibrage de la base de données n'a pas eu beaucoup d'impact sur les résultats contrairement à ce qui était attendu, en particulier sur l'arbre de décision qui y est sensible en général.

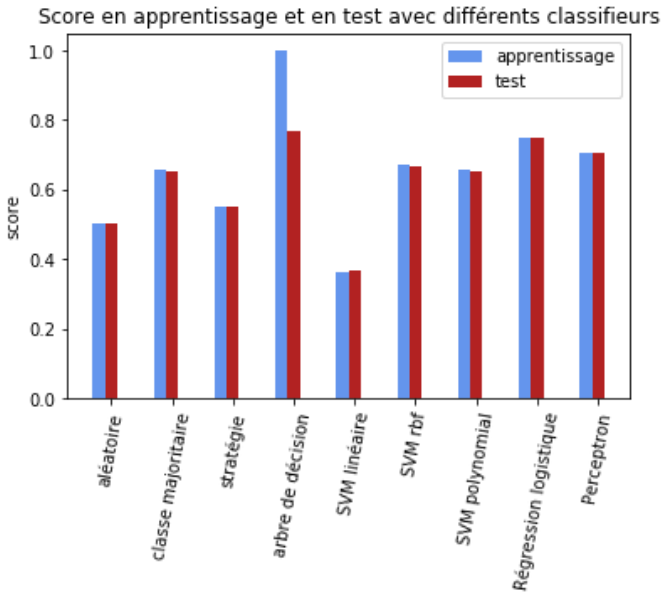
En ce qui concerne le SVM plusieurs problèmes s'imposent d'eux-mêmes. Premièrement, pour obtenir un score de références rapidement nous avons limité le nombre d'itérations lors de l'apprentissage à 500. Il est possible qu'avec plus d'itérations les scores obtenu seraient meilleurs. Deuxièmement, les valeurs des différents attribuent sont très éparses, ce qui pose problème au SVM. Il faudra donc envisager de normaliser les données pour chaque attribut pour améliorer les résultats du SVM.

L'équilibrage de la base de données n'a pas eu beaucoup d'influence sur les scores de la régression logistique et du perceptron.

Pour l'instant, l'arbre de décision est le modèle qui obtient le meilleur score (76,5% en test sur un jeu de données équilibrés). On peut donc considérer ce score comme notre score de référence, cependant lors des traitements précédents nous n'avons pas travaillé sur toute la base de données. Il est possible qu'avec les trente attributs de départ nos modèles soient meilleurs.

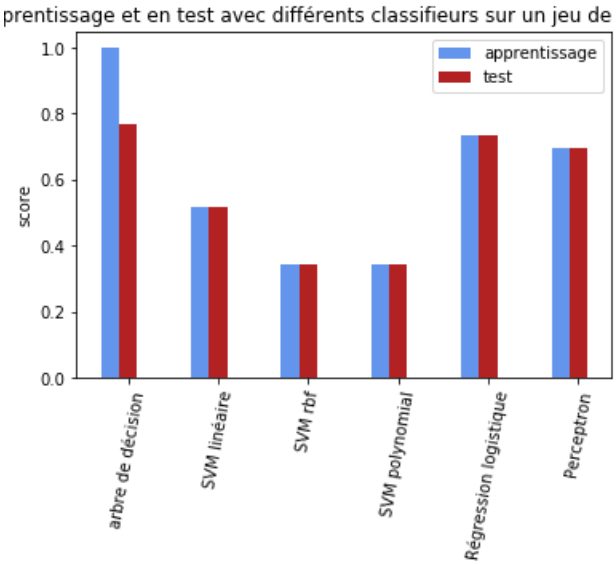
2. Scores de références avec 30 attributs

Nous avons donc réitéré les apprentissages précédents sur la base de données initiale sans les attributs en rapport avec le concours.



	Score app	Score test
aléatoire	0.501259	0.502689
classe majoritaire	0.658851	0.653732
stratégie	0.551008	0.549912
arbre de décision	1.000000	0.768723
SVM linéaire	0.362898	0.368474
SVM rbf	0.670219	0.665343
SVM polynomial	0.658829	0.653708
Régression logistique	0.750697	0.748949
Perceptron	0.706522	0.707482

Figure 10 : Score en apprentissage et en test sur une base de données déséquilibrée avec 30 attributs



	Score app	Score test
arbre de décision	1.000000	0.769670
SVM linéaire	0.515675	0.518136
SVM rbf	0.341540	0.342748
SVM polynomial	0.341661	0.342858
Régression logistique	0.734811	0.736703
Perceptron	0.695233	0.695077

Figure 11 : Score en apprentissage et en test sur une base de données équilibrée avec 30 attributs

Nous observons globalement les mêmes scores que précédemment, ce qui prouve que les attributs que nous avions supprimé pour les apprentissages précédents n'ont qu'une très faible influence sur la création de la plupart des modèles. Cependant nous remarquons que sur la base de données déséquilibré le SVM polynomial est bien meilleur (65,3% de bonnes classifications) que sur la base de données équilibrée ou sur la base de données avec 23 attributs (34% de bonnes classifications). Le perceptron est lui aussi meilleurs

sur la base de données possédant 30 attributs (70%) que sur celle en possédant 23 (52%). Cela nous montre qu'en fonction des modèles utilisés il n'est pas toujours judicieux de retirer des attributs de la base de données même s'ils sont fortement corrélés avec d'autres.

3. Conclusion-Ouverture

Grâce aux nouvelles analyses réalisées nous sommes en possession d'un score minimum à obtenir lors de la création du modèle final. Nos futures modèles devront au moins être aussi performants que l'arbre de décisions sur la base de données équilibrée à 30 attributs (76.9%) pour être acceptables. On peut chercher à améliorer les scores obtenus en paramétrant mieux nos modèles, en utilisant des modèles plus sophistiqués et en réalisant un prétraitement des données(normalisation, suppression des valeurs manquante…).

TP3 – Analyses approfondies

Précédemment nous avons calculé l'accuracy, c'est-à-dire le pourcentage de bonnes prédictions. Cependant le but de ce projet est de détecter un évènement "bosons de Higgs" c'est-à-dire qu'il est préférable de détecter des évènement "bosons de Higgs" qui n'en sont pas (faux positifs) plutôt que de ne pas les détecter (faux négatifs). Dans cette optique, nous utilisons également le score AMS (Approximate Median Significance) qui permet de tenir compte lors de l'apprentissage de nos modèles du taux de vrais positifs et de faux positifs.

Afin de trouver le modèle optimal nous avons décidé de faire varier les hyper-paramètres de différents classifieurs dans le but d'obtenir la meilleure combinaison possible. Pour cela nous avons séparé notre base de données en trois ensembles, un ensemble d'apprentissage qui permet d'apprendre le modèle, un ensemble de validation qui permet de trouver les hyper-paramètres optimaux de chaque modèle et enfin un ensemble de test afin de comparer les modèles entre eux. De plus pour rendre nos scores plus fiables nous avons réalisé une cross-validation sur les ensembles d'apprentissage et de validation.

1. Arbre de décisions

Dans les tps précédents nous avons remarqué que l'arbre de décisions obtenait un très bon score, nous avons donc cherché à l'améliorer. Pour cela nous avons fait varier la profondeur de l'arbre.

	acc train	acc val	ams train	ams val
2	0.754778	0.754150	1.523584	0.760833
6	0.798839	0.793871	1.780144	0.881271
10	0.817576	0.802654	1.924483	0.926998
14	0.838468	0.796576	2.058230	0.906113
18	0.876771	0.784306	2.358431	0.853497
22	0.923529	0.773837	3.024329	0.806327
26	0.959928	0.767829	4.165128	0.771874
30	0.982319	0.764006	6.237912	0.749371
34	0.993191	0.761778	9.756549	0.736290

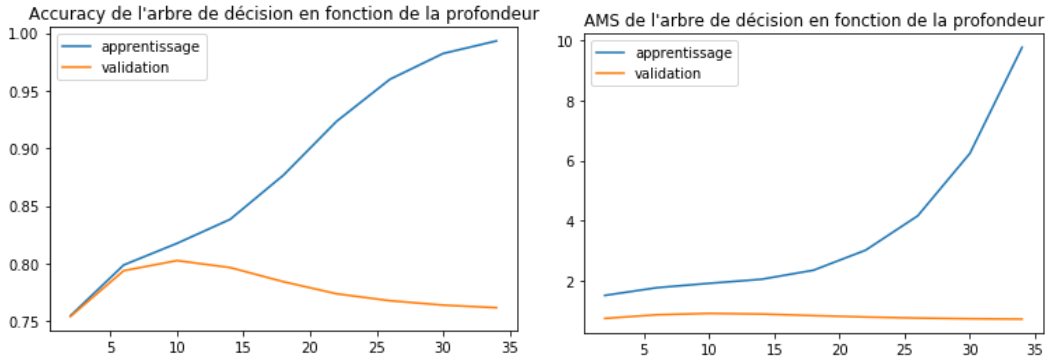


Figure 12 : Score d'accuracy et d'AMS d'un arbre de décision en fonction de la profondeur

Nous observons que plus l'arbre est profond plus les scores d'AMS et d'accuracy en apprentissage augmentent. Cependant, ce n'est pas le cas des scores en validation, cela indique un sur-apprentissage du modèle. Il faut donc bien choisir la profondeur de l'arbre sous peine d'obtenir des scores décevants en

phase de test. La profondeur optimale est 10 dans notre cas, le score d'accuracy est alors de 0.80 en validation et le score d'AMS est de 0.93 en validation.

Une fois l'hyper-paramètre de profondeur choisis (10) on peut tester le modèle sur un ensemble de données qu'il n'a encore jamais vu (ensemble de test). Nous obtenons un score d'accuracy de 0.8125 et d'AMS de 1.0473.

2. Régression logistique

Dans le tp 2 nous avons également observé que la régression logistique obtenait un bon score. Nous avons essayé d'améliorer ce score en cherchant ses hyper-paramètres optimaux. Pour cela nous avons commencé par observer l'influence du paramètre C lorsqu'on utilisait la pénalité l2.

La variable C permet d'insuffler une pénalité à tous les points qui serait du mauvais côté de la frontière de décisions trouvé par le modèle. Ainsi plus la valeur de la variable C est grande plus la pénalité est important et plus le modèle va essayer de corriger son erreur.

	acc train	acc val	ams train	ams val
0.01	0.735073	0.735001	1.613163	0.805926
0.10	0.735166	0.735013	1.613926	0.806101
1.00	0.735175	0.735025	1.614009	0.806192
10.00	0.735172	0.735078	1.613934	0.806228
50.00	0.735191	0.735075	1.613794	0.806300

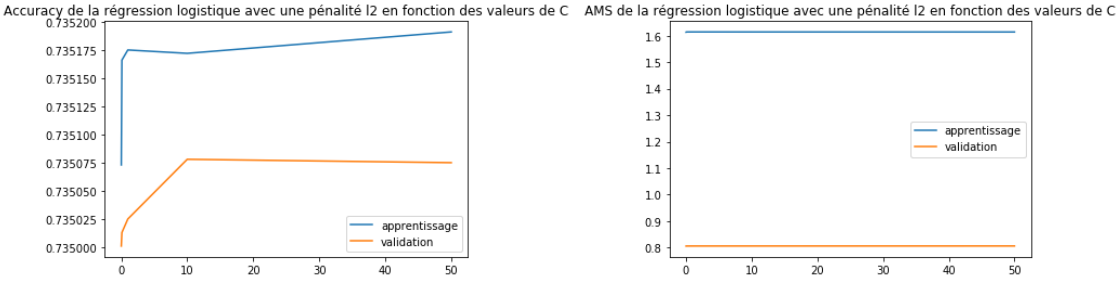


Figure 13 : Score d'accuracy et d'AMS pour la régression logistique en fonction de la valeur de C

La valeur du paramètre C semble ne pas avoir beaucoup d'influence sur les résultats obtenus, en effet les scores varient seulement de 0.0001. On peut imaginer que le modèle n'arrive pas à trouver une frontière permettant de classer tous les points. Le meilleur score d'AMS en validation est de 0.8063 pour C qui vaut 50.

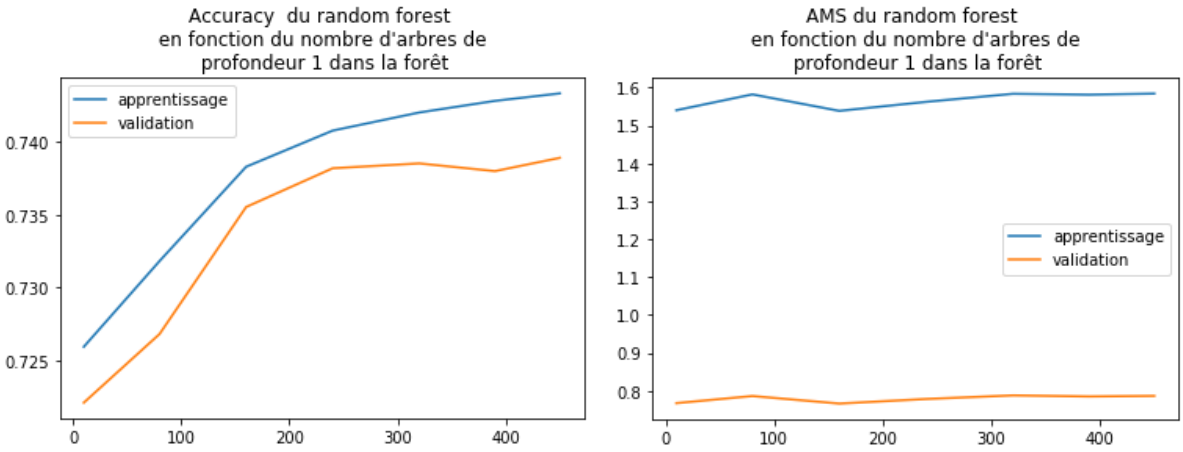
Nous avons envisagé que la pénalité puisse influencer nos scores, nous avons refait l'expérience avec une pénalité l1. Cependant les scores obtenus sont à 0.0001 près les mêmes qu'avec la pénalité l2. Nous en concluons que pour la régression logistique dans notre cas les valeurs des hyper-paramètres n'ont pas d'influence sur les scores.

Nous avons donc appris un modèle ayant une pénalité l2 et une valeur de C égale à 50 afin de l'évaluer sur les données de test. Nous obtenu un score d'accuracy 0,735 et d'AMS de 0,893.

On peut remarquer malgré l'optimisation des hyper-paramètres que le score d'accuracy de la régression logistique reste inférieur au score de référence que nous avions obtenu au tp 2. En effet, nous avions considéré que le score de référence était de 0.769, ce score était obtenu avec un arbre de décision sur un ensemble de donné équilibré.

3. Forêt d'arbres aléatoires

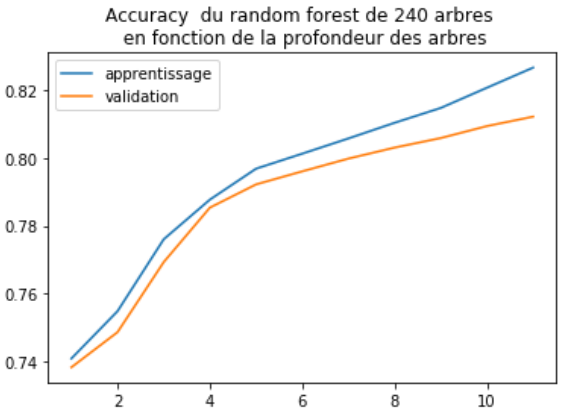
Dans le but d'éviter le sur-apprentissage de l'arbre de décision nous avons décidé d'entraîner une forêt d'arbres de décisions. Nous avons commencé par entraîner une forêt contenant des arbres de profondeur 1. Nous avons observé l'influence du nombre d'arbres dans la forêt sur les scores.

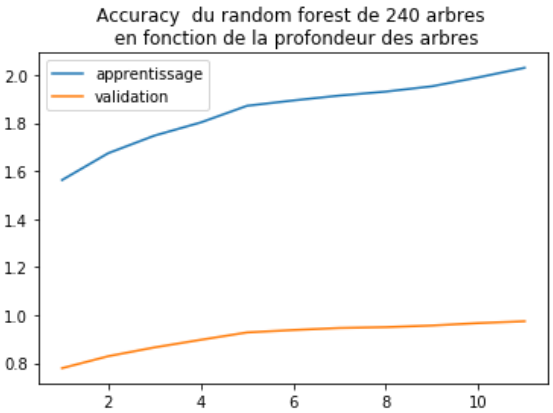


	acc train	acc val	ams train	ams val
10	0.725933	0.722095	1.540513	0.767624
80	0.731820	0.726809	1.582381	0.786351
160	0.738296	0.735532	1.538832	0.766524
240	0.740779	0.738193	1.562680	0.778584
320	0.742023	0.738525	1.583939	0.788065
390	0.742821	0.737996	1.581568	0.785215
450	0.743335	0.738910	1.584428	0.786728

Figure 14 : Scores en accuracy et AMS sur un random forest en fonction du nombre d'arbres de profondeur 1 dans la forêt

Nous observons que plus le nombre d'arbres augmente plus les scores d'accuracy et d'AMS augmentent. Cependant le score d'accuracy en validation se stabilise autour de 0,74 et le score d'AMS en validation oscille autour de 0.79. Ces scores sont moins bons que les scores obtenus par l'arbre de décision de profondeur 10. Nous avons donc décidé de faire varier la profondeur des arbres d'une forêt de 240 arbres dans le but de trouver la profondeur optimale.



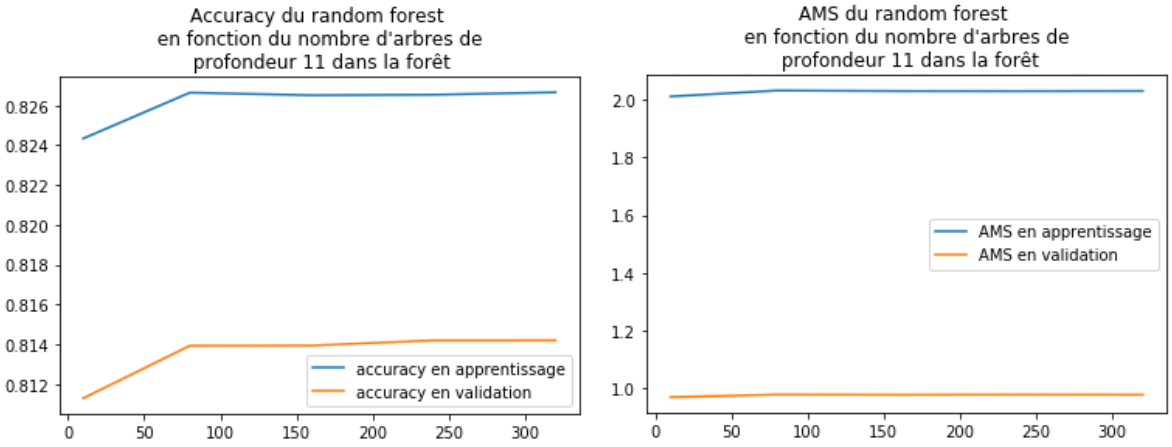


	acc train	acc val	ams train	ams val
1	0.740779	0.738193	1.562680	0.778584
2	0.754771	0.748575	1.674717	0.828592
3	0.775993	0.769253	1.747886	0.865261
4	0.787701	0.785334	1.802701	0.896912
5	0.796829	0.792191	1.872088	0.927593
6	0.801259	0.796043	1.894331	0.937661
7	0.805756	0.799803	1.914702	0.946143
8	0.810374	0.803055	1.931236	0.949606
9	0.814758	0.805869	1.953456	0.956430
10	0.820727	0.809400	1.990489	0.966611
11	0.826662	0.812185	2.030416	0.974106

Figure 15 : Scores d'accuracy et d'AMS d'un random forest de 240 arbres en fonction de la profondeur des arbres

Nous observons que plus la profondeur des arbres augmente meilleurs sont les scores que ce soit en apprentissage ou en validation. On peut également noter que les forêts aléatoires sur-apprennent beaucoup moins que les arbres de décision. Cependant plus la profondeur des arbres est élevées plus le temps d'apprentissage est grand. Nous avons donc décidé de ne pas continuer d'augmenter leur profondeur.

Nous avons donc décidé de faire varier le nombre d'arbres dans une forêt aléatoire contenant des arbres de profondeur 11.



	acc train	acc val	ams train	ams val
10	0.824335	0.811291	2.011777	0.968937
80	0.826638	0.813931	2.032375	0.977687
160	0.826508	0.813937	2.030499	0.977021
240	0.826534	0.814197	2.030260	0.977554
320	0.826654	0.814201	2.030785	0.977162

Figure 16 : Scores d'accuracy et d'AMS d'un random forest en fonction du nombre d'arbres de profondeur 11 dans la forêt

On voit que l'accuracy augmente et se stabilise autour de 0.814 en validation et l'AMS se stabilise autour de 0.97 en validation. Il n'est donc pas nécessaire d'augmenter le nombre d'arbres car le gain serait trop infime par rapport au temps de calcul pour être significatif.

Pour des raisons de temps de calcul, nous avons réalisé notre test sur une forêt aléatoire de 240 arbres ayant une profondeur de 11. Le score d'accuracy est de 0,8196 et le score d'AMS est de 1,0972.

4. AdaBoost

Dans le but de contrer le sur-apprentissage nous avons entraîné un classifieur adaboost. Ce classifieur consiste à enchaîner pleins de classifieurs simples les uns aux autres. Nous avons fait varier le nombre de classifieurs simples afin de trouver quel nombre permet d'obtenir les meilleurs résultats. Les classifieurs simples que nous avons utilisés sont des arbres de décision de profondeur 3.

	acc train	acc val	ams train	ams val
10	0.822764	0.810697	2.054752	0.986826
60	0.836546	0.819903	2.232534	1.047249
110	0.839592	0.819447	2.262543	1.046682
160	0.841158	0.818098	2.280249	1.045058
210	0.842303	0.817941	2.291583	1.043620

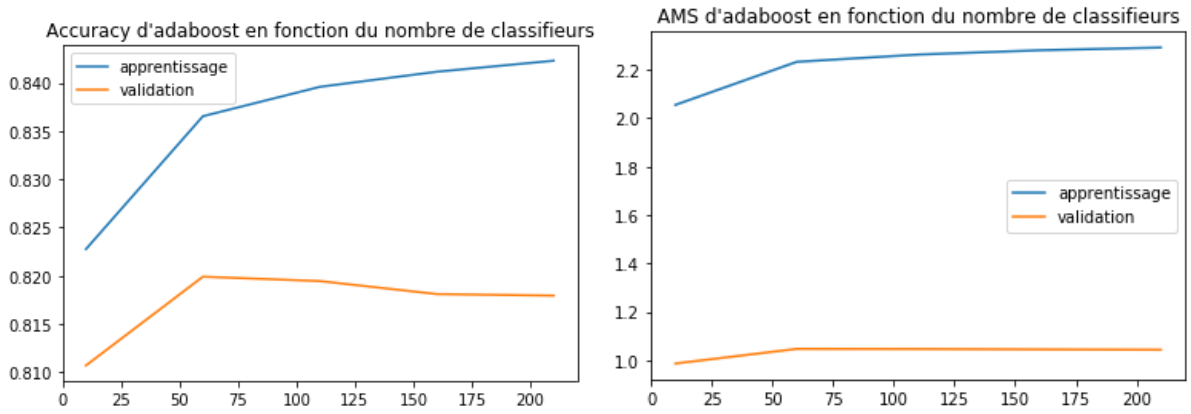


Figure 17 : Score d'accuracy et d'AMS d'AdaBoost en fonction du nombre de classifieurs faibles

Nous remarquons que les scores d'accuracy en apprentissage augmentent en fonction du nombre de classifieurs simples utilisé, alors que le score d'accuracy en validation augmente jusqu'à 0,8199 pour 60 classifieur puis diminue progressivement. On observe le même schéma pour le score d'AMS, le meilleur score en validation est de 1,047. Cela traduit un sur-apprentissage, cependant il est moins important que celui observé avec les arbres de décision.

Nous avons entrainé un adaboost contenant 60 classifieurs simples de type arbre de décision de profondeur 3 afin de pouvoir tester les performances du modèle sur l'ensemble de test. Le modèle obtient un score d'accuracy de 0,8356 en test et un score d'AMS en test de 1,2214.

5. Conclusion

Dans la dernière partie du projet nous avons implémenté quatre modèles de classification différents que nous avons cherché à optimiser en trouvant les hyper- paramètres qui permettent d'obtenir les meilleurs résultats. Le tableau suivant rappelle pour chaque modèle quels sont les scores d'accuracy et d'AMS qui ont été obtenus sur l'ensemble de test.

	Accuracy en test	AMS en test
Arbre de décision	0,8125	1,0473
Régression logistique	0,7352	0,8933
Forêt aléatoire	0,8196	1,0972
AdaBoost	**0,8356**	**1,2214**

On observe que le modèle qui obtient les meilleurs scores est adaboost. La forêt aléatoire et l'arbre de décision obtiennent des scores un peu moins bons, mais meilleurs que le score de référence (0.769) alors que la régression logistique obtient un score moins inférieur.

En conclusion, pour résoudre le problème du boson de higgs nous recommandons donc d'utiliser un classifieur adaboost contenant 60 arbres de décision d'une profondeur de 3.