

STA211 : Méta-algorithmes

Vincent Audigier

14 mai, 2019

- 1 Introduction
- 2 Bagging
 - 2.1 Régression
 - 2.2 Classification
- 3 Boosting
- 4 Comparaison
- 5 Exemples
 - 5.1 Bagging
 - 5.2 Forêts aléatoires
 - 5.3 Boosting
- 6 Conclusion
- Références

1 Introduction

Différentes méthodes supervisées ont été présentées dans cette partie. Il est possible d'en améliorer les performances en utilisant des méthodes d'agrégation qui consistent à combiner plusieurs prédictions fournies par différents modèles. Ce type d'approche a déjà été abordé au travers des forêts aléatoires dans la mesure où une forêt peut être vue comme la combinaison des règles de prédictions fournies par plusieurs arbres. Nous présentons ici deux procédures d'agrégation très classiques : le bagging et le boosting.

2 Bagging

Nous présentons d'abord le bagging dans le cadre d'un problème de régression, puis développons ensuite ses spécificités dans le cadre de la classification.

2.1 Régression

Le Bagging est aussi appelé *agrégation bootstrap*. Il s'agit d'une procédure qui permet de réduire la variance d'une méthode supervisée. Le fondement de cette méthode repose sur le résultat suivant : si on se donne un n -échantillon (T_1, \dots, T_n) , tel que $\text{Var}[T_i] = \sigma^2$ pour tout $1 \leq i \leq n$, alors $\text{Var}[\sum_{i=1}^n T_i] = \sigma^2/n$. Autrement dit, si on dispose de n réalisations indépendantes d'une variable aléatoire T , alors la moyenne de ces n valeurs est n fois moins variable que T . Notons que l'espérance de cette moyenne ne sera quant à elle pas modifiée. En appliquant ce résultat à différents prédicteurs on espère obtenir un prédicteur ``moyen'' avec moins de variance que chacun des prédicteur pris isolément (mais avec un biais similaire). Ceci est vrai à condition de disposer de plusieurs prédicteurs indépendants les uns des autres.

Le principe du Bagging est alors le suivant : on constitue B échantillons bootstrap à partir des données. Pour cela, on peut soit faire du ré-échantillonnage avec remise des n individus de façon à constituer B échantillons de taille n , soit constituer des échantillons de taille inférieure (via des tirages avec ou sans remise). Ces B échantillons sont constitués indépendamment les uns des autres. Par la suite on construit un modèle pour

chaque échantillon bootstrap (par exemple un arbre non élagué). On agrège ensuite les modèles en moyennant les prédictions de chacun. Notons que pour certains modèles, faire la moyenne des prédictions revient à faire la moyenne des paramètres, puis à retenir les prédictions correspondantes, mais ce n'est pas toujours le cas, en particulier pour un modèle non-paramétrique tel qu'un arbre ceci n'a même pas de sens.

Le tirage d'échantillons indépendants ne suffit pas à garantir l'indépendance entre les B prédictions obtenues dans la mesure où toutes restent construites à partir du même jeu de données. Ainsi, on priviliege généralement des prédicteurs instables (i.e. dont la prédiction varie beaucoup en fonction de l'échantillon d'apprentissage considéré), tels que les arbres, de façon à limiter le lien entre les B différentes prédictions. Par ailleurs, dans la mesure où le bagging a pour objectif de réduire la variabilité de la prédiction, il ne sera généralement pas utile de l'appliquer sur des modèles stables. Pour autant, il n'est pas nécessairement judicieux de considérer des prédicteurs très fortement variables car le prédicteur *baggé* (i.e. agrégé par bagging) pourrait avoir une variance importante, bien que celle-ci ait été diminuée par rapport au prédicteur de base. De ce point de vue, considérer des prédicteurs de base non biaisés (et donc avec une variance importante) n'est pas forcément un bon choix.

Le fait de n'utiliser qu'une partie de l'échantillon d'apprentissage via le bootstrap permet d'utiliser les individus non sélectionnés en tant qu'échantillon test et donc d'estimer une erreur de prédiction appelée erreur *out-of-bag* (littéralement, *en dehors du bootstrap*) ou simplement OOB. Plus précisément, cette erreur se calcule de la façon suivante : on fixe une observation (X_i, Y_i) de l'échantillon d'apprentissage. On considère ensuite parmi les B prédicteurs construits, l'ensemble des prédicteurs \hat{f} construits sans cette observation, i.e. pour lesquels cette observation est ``Out-Of-Bag''. On agrège alors les prédictions correspondantes pour fournir une unique prédiction \hat{Y}_i de Y_i . On répète l'opération pour toutes les observations et on calcule ensuite l'erreur quadratique moyenne, correspondant à l'erreur OOB :

$$\left(\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2 \right)$$

Ainsi, la procédure de Bagging fournit directement une estimation de l'erreur de généralisation.

Le bénéfice à utiliser le bagging tient d'une part au fait de moyenner les règles de prédiction : on gagnera généralement à considérer un très grand nombre de prédicteurs, car plus B est grand, plus la variance des valeurs prédites sera petite. D'autre part, le bagging peut être avantageux en présence de valeurs aberrantes. Ceci tient au fait que certaines valeurs aberrantes présentes dans les données ne se retrouvent que dans certains échantillons bootstrap et que la moyenne des prédictions bootstrap fait perdre leur influence à ces valeurs aberrantes.

2.2 Classification

Dans un cadre de classification, le bagging reste construit de façon très similaire : on ré-échantillonne les données B fois, on se donne un classifieur pour chaque échantillon puis on agrège les règles de classification ensemble. Quelques spécifiés aux variables réponses qualitatives doivent néanmoins être mentionnées.

En particulier, il est fréquent de s'intéresser davantage à la probabilité d'appartenance à une classe, qu'à la classe prédite elle-même. En effet, on peut prédire qu'un client sera un bon payeur, mais savoir que la probabilité qu'il le soit vaille 0.99 ou 0.51 est également riche d'information. Pour obtenir cette probabilité à partir du bagging, il serait tentant de vouloir calculer pour chaque individu les fréquences relatives des classes prédites par les B prédicteurs. Ceci n'est en général pas pertinent. Par exemple, supposons deux prédicteurs tels que chacun donne une probabilité de 0.8 pour que la sortie associée à une entrée x_0 vaille

'1'. Dans ce cas, chacun prédira '1' pour la sortie, et donc la probabilité d'appartenance à la classe '1' serait estimée par 1, ce qui est très approximatif. Il vaut mieux faire la moyenne des probabilités données par les B prédicteurs. Par exemple, dans le cas des arbres, on considérera la proportion de chaque classe au sein de la feuille à laquelle appartient l'entrée x_0 .

Par ailleurs, même si l'objectif est purement prédictif et que l'on ne s'intéresse pas à la probabilité d'affecter un individu à une classe plutôt qu'une autre, il pourra être également préférable d'agréger les probabilités prédites pour chaque prédicteur puis de retenir la classe la plus probable, plutôt que de considérer la classe majoritaire pour les différents prédicteurs. Ceci permet d'obtenir des prédicteurs moins variables, en particulier quand le nombre de prédicteurs est petit (Hastie, Tibshirani, and Friedman (2009)).

Enfin précisons qu'il est également possible de calculer une erreur OOB dans un cadre de classification. Pour cela, on procède de façon similaire au cadre de la régression en choisissant un critère d'erreur adapté à la nature qualitative de la réponse, en l'occurrence l'erreur de mauvais classement.

3 Boosting

Le boosting est une autre procédure d'agrégation qui comme le bagging consiste à effectuer une moyenne des prédictions de différents prédicteurs (en régression) ou un vote à la majorité (en discrimination).

Néanmoins, il s'agit là d'une moyenne pondérée. Il existe différentes méthodes de boosting, nous présentons ici la procédure *AdaBoost* qui est l'une des plus répandues. Celle-ci est généralement utilisée dans le cadre de la classification, mais des versions dans le cadre de la régression ont aussi été proposées comme AdaBoost.R ou L2 Boosting (c.f. Duffy and Helmbold (2002)). Le principe de cette procédure d'agrégation est de construire une famille d'estimateurs construits de manière récursive : chaque estimateur est une version adaptative du précédent en donnant plus de poids aux observations mal ajustées ou mal prédites.

L'estimateur construit à l'étape b concentrera donc ses efforts sur les observations mal ajustées par l'estimateur à l'étape $b - 1$. Le poids des individus à chaque itération sert ensuite à calculer une erreur de mauvais classement qui permettra de définir le poids accordé à chaque version de l'estimateur lors de l'agrégation.

Plus précisément l'algorithme AdaBoost est le suivant (Tufféry (2015)):

1. Initialiser les poids des n observations de l'échantillon d'apprentissage : $p_i^{(0)} = 1/n$ pour $1 \leq i \leq n$
2. Itération : pour b de 1 à B
 - Ajuster le classifieur f_b sur l'échantillon d'apprentissage pondéré par les poids $(p_i^{(b-1)})_{1 \leq i \leq n}$.
 - Calculer le taux d'erreur e_b pondéré de \hat{f}_b : $e_b = \sum_{i=1}^n p_i^{(b-1)} I(y_i \neq \hat{f}_b(x_i))$
 - Calculer le coefficient $\alpha_b = \log\left(\frac{1-e_b}{e_b}\right)$
 - Mettre à jour les poids des individus mal classés selon : $p_i^{(b)} = \exp(\alpha_b)p_i^{(b-1)}$ si $y_i \neq \hat{f}_b(x_i)$ et $p_i^{(b)} = p_i^{(b-1)}$ sinon
 - Normaliser les poids : $p_i^{(b)} = \frac{p_i^{(b)}}{\sum_{i=1}^n p_i^{(b)}}$
3. Le classifieur boosté est alors donné par $\sum_{b=1}^B \alpha_b \hat{f}_b$.

A l'itération b Le boosting tend à prédire au mieux les individus les plus difficiles à prédire, ceci au détriment des individus plus facile à prédire. Ainsi, l'itération $b + 1$ tend à inverser la tendance en ré-affectant un poids important à ces derniers, et ainsi de suite. Ainsi, les prédicteurs construits à chaque itération sont localement

optimaux seulement (i.e. performants pour un sous-ensemble d'individus), mais le classifieur agrégé lui est globalement optimal (performant pour l'ensemble des individus). On notera en effet que α_b est d'autant plus petit que le taux de mauvais classement est grand. Ainsi, le classifieur boosté accorde un poids plus important aux modèles les plus performants globalement.

Il existe d'autres méthodes de boosting, citons par exemple la procédure arc-x4⁶¹ dans laquelle les poids sont mis à jour différemment selon les individus, ou Real AdaBoost qui met à jour les poids en fonction de l'estimation de la probabilité de succès, plutôt que de l'erreur de mauvais classement. De façon générale, cet algorithme peut être paramétré à plusieurs niveaux :

- choix du critère d'erreur pour la pondération des individus
- mise à jour du coefficient α : ceci peut être effectué sur la base des pondérations courantes des individus, ou des précédentes également de façon à "insister" sur la correction des individus mal classés
- gestion des individus "trop" influents pour l'algorithme : on peut les écarter au départ, ou ne plus actualiser leur poids au cours de l'algorithme, ou leur affecter un poids nul à l'itération suivant la détection de leur influence trop importante)
- méthode d'agrégation des prédicteurs : par exemple on peut vouloir imposer certains coefficients α_b à zéro pour les modèles trop mal ajustés
- conservation ou non de l'intégralité des observations : on peut en effet envisager un ré-échantillonnage des individus à chaque itération selon le poids qui leur est affecté. C'est ce qui est proposé dans l'algorithme Arcing afin de permettre de rompre le caractère cyclique des pondérations.

Contrairement au bagging, le boosting permet de réduire à la fois le biais et la variance. Ainsi, une procédure de boosting basée sur un classifieur avec une variance faible mais un biais important peut devenir performante en termes de prédiction. C'est par exemple le cas des arbres à deux feuilles (appelés *stumps*).

Ajoutons que les pondérations des modèles sont fonction des données en boosting. Ainsi, le boosting est sujet à des problèmes de sur-apprentissage, problèmes qui sont exacerbés en présence de données fortement bruitées. Le nombre d'itérations en boosting est donc un paramètre important qu'il convient de régler. Des méthodes de validation-croisée sont notamment proposées dans les logiciels afin de choisir ce paramètre.

4 Comparaison

Bagging et boosting sont deux façons différentes d'agrégier des prédicteurs. Ils diffèrent selon différents points et présentent donc des avantages et inconvénients différents (résumés dans le Tableau 4.1 extrait de Tufféry (2007)). Tout d'abord, le boosting utilise toutes les données de l'échantillon d'apprentissage, alors que le Bagging s'appuie sur différents échantillons bootstrap. Ainsi, le boosting est purement déterministe tandis que le bagging est aléatoire. Par ailleurs, les modèles agrégés dans le bagging accordent la même importance à toutes les observations (en moyenne), alors que les modèles agrégés en boosting sont spécifiques à certains individus. Aussi, le boosting est séquentiel, contrairement au bagging. Ceci est préjudiciable en termes de coût algorithmique car la procédure ne peut plus être parallélisée. Ajoutons que les pondérations des modèles sont fixes en bagging, alors qu'elles sont fonction des données en boosting ce qui conduit à des problèmes de sur-ajustement. Enfin, une différence importante entre bagging et boosting est que le premier réduit uniquement la variance tandis que le second réduit à la fois le biais et la variance.

Table 4.1: Comparatif du bagging et boosting

Bagging	Boosting
Caractéristiques	
Le bagging est aléatoire	Le boosting est adaptatif et généralement déterministe
On utilise des échantillons bootstrap	On utilise généralement l'échantillon initial complet
Chaque modèle produit doit être performant sur l'ensemble des observations	Chaque modèle produit doit être performant sur certaines observations ; un modèle performant sur les valeurs aberrantes sera moins performant sur les autres individus
Dans l'agrégation, tous les modèles ont le même poids	Dans l'agrégation, les modèles sont généralement pondérés selon leur qualité d'ajustement
Avantages et inconvénients	
Technique de réduction de la variance par moyenne de modèle	Peut diminuer la variance et le biais du classifieur de base, mais la variance peut augmenter avec un classifieur de base stable
Perte de lisibilité quand le classifieur de base est un arbre de décision	Perte de lisibilité quand le classifieur de base est un arbre de décision
Inopérant sur les stumps (sauf double randomisation comme dans les forêts aléatoires)	Efficace sur les stumps
Possibilité de paralléliser l'algorithme	Algorithme séquentiel ne pouvant pas être parallélisé
Pas de sur-apprentissage : supérieur au boosting en présence de 'bruit'	Risque de sur-apprentissage mais globalement supérieur au bagging sur les données non ``bruitées''
En résumé, le bagging est souvent plus efficace que le boosting...	... mais quand le boosting est efficace, il dépasse le bagging.

5 Exemples

Afin d'illustrer ces méta-algorithmes, nous reprenons le jeu de données German Credit où l'on souhaite prédire la variable *Creditability* indiquant si un client est un bon payeur ou non. Il s'agit donc d'un problème de classification où la variable réponse a deux modalités. Le jeu de données est découpé en échantillon d'apprentissage et échantillon test (selon les proportions respectives 2/3 et 1/3).

```
#l'objet don correspond au jeu de données German Credit
set.seed(235)
id<-sample(seq(nrow(don)), size=ceiling(nrow(don)*2/3))
test<-don[-id,]
ech<-don[id,]
```

5.1 Bagging

Dans un premier temps, on construit un arbre CART à partir du jeu d'apprentissage puis on évalue ses performances sur le jeu de données test. L'AUC obtenue vaut 0.675 tandis que le taux de mauvais classement vaut 0.285.

```
set.seed(235)
library(ROCR)
res.tree<-rpart(Creditability~, data=ech) #ajustement de l'arbre non élagué
res.prune<-prune(res.tree, cp=0.046) #élagage
res.pred<-predict(res.prune, newdata = test)[,1] #calcul des probabilités d'être bon payeur sur l'échantillon test
AUC_CART<-performance(prediction(res.pred, test$Creditability), measure="auc")@y.values[[1]] #calcul de l'AUC
Err_CART<-performance(prediction(res.pred, test$Creditability), measure ="err")@y.values[[1]][4] #calcul de l'erreur de mauvais classement
cat("AUC CART : ", AUC_CART); cat(" Erreur CART : ", Err_CART) #affichage
```

```
## AUC CART : 0.6751502
```

```
## Erreur CART : 0.2852853
```

Dans un deuxième temps, on applique la procédure de bagging en ré-échantillonnant avec remise le jeu d'apprentissage. Sur chacun des $B = 200$ échantillons bootstrap de taille $n = 666$, un arbre non élagué avec 5 individus par feuille est ajusté. Ceci peut par exemple être mis en oeuvre à l'aide du package R ipred.

```
set.seed(235)
library(ipred)
bag<-bagging(Creditability~.,
              data=ech,
              nbagg=200,
              coob=TRUE,
              control=rpart.control(minbucket = 5))
```

L'agrégation selon les probabilités d'appartenance aux deux classes conduit à une AUC de 0.7686 sur l'échantillon test. Comme attendu, l'AUC est plus faible en agrégeant les prédictions fournies par les B arbres selon le vote majoritaire (AUC de 0.767).

```
#aggregation des proba
test$bag1<-predict(bag,test,type="prob",aggregation="average")
pred1<-prediction(test$bag1[,1],test$Creditability)
AUC_proba<-performance(pred1,"auc")@y.values[[1]]

#aggregation des valeurs prédictées
test$bag2<-predict(bag,test,type="prob",aggregation="majority")
pred2<-prediction(test$bag2[,1],test$Creditability)
AUC_pred<-performance(pred2,"auc")@y.values[[1]]

cat("AUC selon probabilités : ", AUC_proba);cat("AUC selon prédictions : ", AUC_pred)
```

```
## AUC selon probabilités : 0.7686052
```

```
## AUC selon prédictions : 0.7666094
```

Enfin, on peut vérifier que l'erreur OOB est proche du taux de mauvais classement obtenu sur l'échantillon test.

```
cat("erreur test : ",performance(pred1,measure ="err")@y.values[[1]][251]);cat("erreur
OOB : ",bag$err)

## erreur test : 0.2552553

## erreur OOB : 0.2353823
```

NB : le nombre d'échantillons bootstrap a ici été choisi a priori à titre illustratif. Il conviendra en pratique de régler ce paramètre. Ceci pourra être effectué en examinant l'évolution de l'erreur OOB en fonction du nombre d'échantillons considérés, mais on utilisera en général plutôt une procédure de validation croisée.

5.2 Forêts aléatoires

Les forêts aléatoires sont une procédure de bagging particulière dans le sens où non seulement les individus sont ré-échantillonnés (comme classiquement en bagging), mais où en plus les variables explicatives utilisées pour construire les noeuds sont choisies parmi un sous-ensemble de variables tiré au hasard. Ceci a pour effet de diminuer la liaison entre les différents arbres construits et améliore de ce fait les performances du prédicteur baggé. La prédiction par forêts aléatoires peut par exemple être effectuée à l'aide du package randomForest. Ici, avec 200 arbres on obtient une AUC de 0.80 sur l'échantillon de test.

```
library(randomForest)
set.seed(235)
res.rf<-randomForest(y=ech$Creditability,x=ech[,-c(21,22)],ntree = 200)
cat("AUC Random_Forest : ",
    performance(prediction(predict(res.rf,test,type="prob",aggregation="average"))[,1],
                test$Creditability),"auc")@y.values[[1]])
```

```
## AUC Random_Forest : 0.8054506
```

5.3 Boosting

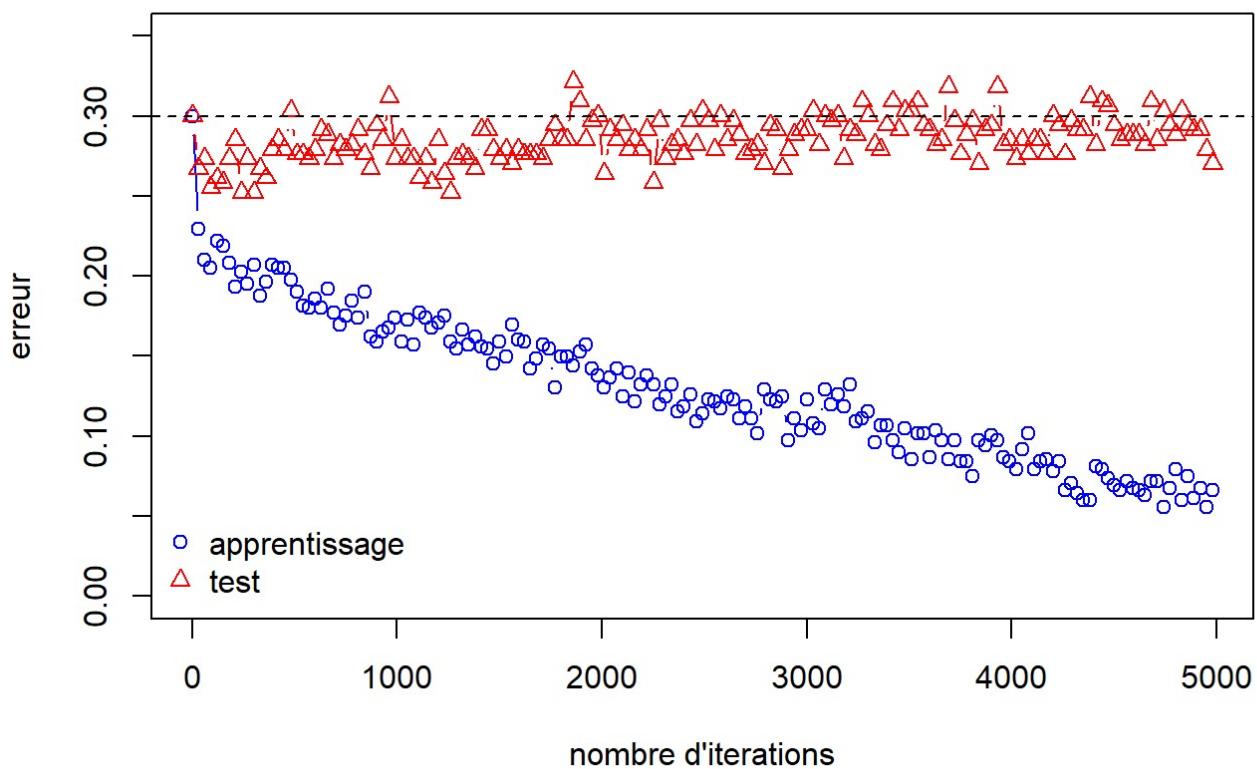
A présent, on propose de comparer le bagging au boosting en utilisant comme prédicteur de base des stumps qui ont l'avantage d'être peu coûteux algorithmiquement. Le boosting peut être par exemple réalisé à l'aide du package R gbm ou ada. On commence par inspecter l'influence du nombre d'itérations de la procédure sur l'erreur en considérant l'échantillon d'apprentissage ou l'échantillon test.

```
library(gbm)
set.seed(235)
B<-5000#nombre d'iterations
ech$Creditability.bin<-as.numeric(ech$Creditability)-1#0 = good
test$Creditability.bin<-as.numeric(test$Creditability)-1
model <- gbm(Creditability.bin~,data=ech[,-21],distribution="adaboost",interaction.depth=1,shrinkage=1,n.trees=B)

#représentation graphique de l'influence de B
boucle <- seq(1,B,by=30)
errapp <- errtest <-rep(0,length(boucle))

k <- 0
for (i in boucle){
  k <- k+1
  prev_app <- predict(model,newdata=ech[,-21],n.trees=i)
  errapp[k] <- sum(as.numeric(prev_app>0)!=ech[,-21]$Creditability.bin)/nrow(ech)
  prev_test<- predict(model,newdata=test[,-21],n.trees=i)
  errtest[k] <- sum(as.numeric(prev_test>0)!=test[,-21]$Creditability.bin)/nrow(test)
}

plot(boucle,errapp,type="b",col="blue",ylim=c(0,0.35),xlab="nombre d'iterations",
ylab="erreur",lty=1)
points(boucle,errtest,col="red",type="b",pch=2)
abline(0.3,0,lty=2)
legend("bottomleft",legend=c("apprentissage","test"),col=c("blue","red"),pch=c(1,2),bt
y="n")
```



La Figure ci-dessus illustre clairement le problème de sur-apprentissage : l'erreur d'apprentissage diminue sans cesse tandis que l'erreur de test se rapproche de 0.3 correspondant à la proportion de mauvais payeurs dans le jeu de données. Ainsi, si B est trop élevé, les performances du prédicteur baggé avoisinent celle d'un tirage au hasard.

En retenant les 120 premiers arbres, on obtient une AUC à 0.775 sur l'échantillon test.

```
AUC_boosting<-gbm.roc.area(test$Creditability.bin, predict(model,newdata=test[,-21],n.trees = 120))
cat("AUC boosting : ",AUC_boosting)
```

```
## AUC boosting : 0.7746781
```

Comparons à présent avec la procédure de bagging sur les stumps

```
set.seed(235)
stump<-bagging(Creditability~.,data=don[id,],nbagg=200,coob=TRUE,control=rpart.control(maxdepth = 1,cp=-1))
test$stump<-predict(stump,test,type="prob",aggregation="average")
pred<-prediction(test$stump[,1],test$Creditability)
AUC_bagging<-performance(pred,"auc")@y.values[[1]]
cat("AUC bagging : ",AUC_bagging)
```

```
## AUC bagging : 0.7040558
```

L'AUC est plus faible. Ceci n'est pas surprenant car le boosting permet de corriger à la fois le biais et la variance des stumps, tandis que le bagging ne diminue que la variance.

6 Conclusion

Les méta-algorithmes peuvent grandement améliorer les performances de prédiction d'un prédicteur de base. Ils sont souvent à la base des stratégies gagnantes des compétitions de sciences des données telles que le challenge Netflix. Néanmoins, il n'est pas garanti que le prédicteur agrégé soit plus performant que le prédicteur de base. Pour cette raison, il est important de s'assurer du gain à employer une procédure d'agrégation sinon on risque juste de perdre en interprétabilité, voire de perdre également en qualité de prédiction.

Ce document se limite à l'agrégation de modèles appartenant à une même famille, mais le cadre de la combinaison de modèles est plus large. Il est en effet possible de combiner des modèles issus de familles différentes, par exemple on pourrait combiner des prédictions par réseau neuronal et par régression logistique, on parle alors de *stacking*. Toute la difficulté est alors de définir le poids à accorder à chaque modèle. Dans un cadre de régression, on peut par exemple rechercher la combinaison linéaire des poids qui minimise le critère des moindres carrés entre les sorties observées et les sorties prédites (voir e.g. Hastie, Tibshirani, and Friedman (2009) pour plus de précisions).

Références

- Duffy, Nigel, and David Helmbold. 2002. "Boosting Methods for Regression." *Machine Learning* 47 (2): 153–200.
- Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. 2009. *The Elements of Statistical Learning*. Springer Series in Statistics. New York, NY, USA: Springer New York Inc. <https://web.stanford.edu/~hastie/Papers/ESLII.pdf> (<https://web.stanford.edu/~hastie/Papers/ESLII.pdf>).
- Tufféry, S. 2007. *Data Mining et Statistique décisionnelle: L'intelligence Des Données*. Editions Technip.
- . 2015. *Modélisation Prédictive et Apprentissage Statistique Avec R*: Éditions Technip.
- Wikistat. 2017. "Agrégation de Modèles — Wikistat." <https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-agreg.pdf> (<https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-app-agreg.pdf>).