Analyse des Forêts Aléatoires et Séléction de Variables

Kenson FAVEUR

2025-03-07

Introduction

Ce rapport explore l'utilisation des arbres de décision et des forets aléatoires pour détecter les spams. Nous analyserons également l'importance des variables et la séléction automatique des meilleures caractéristiques.

Chargement des données

Nous utilisons le jeu de données "spam" de la librairie " kernlab".

```
# Charger la librairie nécessaire
library(kernlab)

# Charger le jeu de données spam
data(spam)

# Séparer les données en apprentissage et test
set.seed(9146301)
ytable <- table(spam$type)
app <- c(sample(1:ytable[2], ytable[2]/2),
         sample((ytable[2] + 1):nrow(spam), ytable[1]/2))

spam.app <- spam[app, ]  # Données d'apprentissage
spam.test <- spam[-app, ]  # Données de test

# Vérification des dimensions
dim(spam.app)
```

```
## [1] 2300   58
```

```
dim(spam.test)
```

```
## [1] 2301   58
```

Commentaires

Nous chargeons les données de spam et les divisons en deux ensembles : L'ensemble d'apprentissage contient 2300 observations et 58 variables.L'ensemble de test contient 2301 observations et 58 variables. La somme des observations dans les deux ensembles est 4601, ce qui correspond probablement à la taille initiale du dataset spam.

Interprétation

La division assure un équilibrage en prenant la moitié des observations de chaque classe.

Il y a un léger déséquilibre numérique entre les ensembles d'apprentissage et de test (2300 vs. 2301), probablement dû à un nombre impair d'observations dans spam.

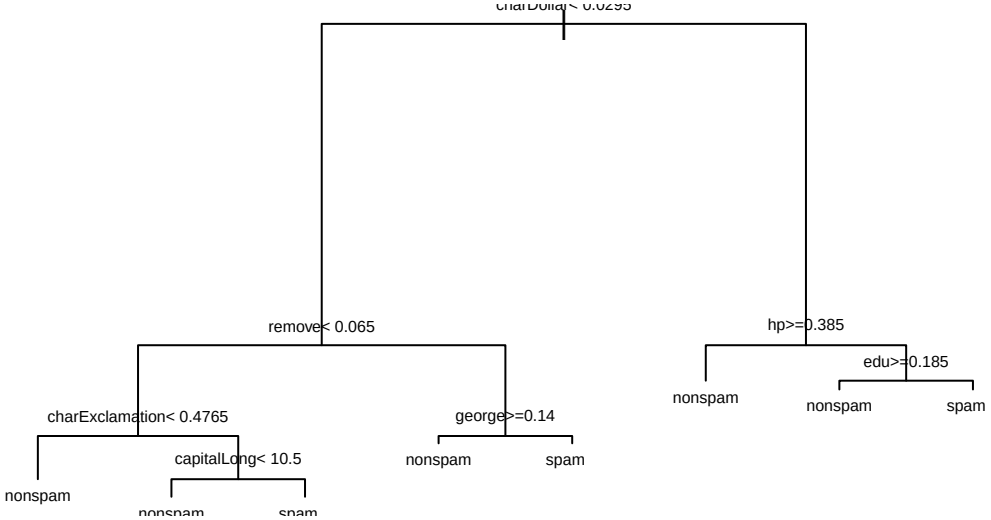
Cette séparation est essentielle pour entraîner un modèle sur spam.app et évaluer ses performances sur spam.test sans biais.

Arbres de décision (CART)

```r
# Charger la librairie rpart
library(rpart)

# Calculer l'arbre par défaut
t_def <- rpart(type ~ ., data = spam.app)

# Afficher l'arbre
plot(t_def)
text(t_def, cex = 0.5)
```



```r
# Calculer l'arbre maximal
tmax <- rpart(type ~ ., data = spam.app, control = rpart.control(minsplit = 1, cp = 0))

# Élagage pour obtenir l'arbre optimal
tprune <- prune(tmax, cp = tmax$cptable[which.min(tmax$cptable[, 4]), 1])

# Comparaison des erreurs
pred_def <- predict(t_def, spam.test, type="class")
err_def <- mean(pred_def != spam.test$type)

pred_max <- predict(tmax, spam.test, type="class")
err_max <- mean(pred_max != spam.test$type)

pred_prune <- predict(tprune, spam.test, type="class")
err_prune <- mean(pred_prune != spam.test$type)

# Afficher les erreurs
cat("Erreur arbre par défaut :", err_def, "\n")
```

```
## Erreur arbre par défaut : 0.1008257
```

```r
cat("Erreur arbre maximal :", err_max, "\n")
```

```
## Erreur arbre maximal : 0.08952629
```

```r
cat("Erreur après élagage :", err_prune, "\n")
```

```
## Erreur après élagage : 0.0890917
```

Interprétation

Arbre par défaut (t_def)

A une erreur plus élevée (10.08%), ce qui peut indiquer un modèle sous-ajusté (trop simple pour capturer les tendances du jeu de données).

Arbre maximal (tmax)

A une erreur plus faible (8.95%), car il capture plus de détails du jeu de données, mais peut souffrir de surajustement.

Arbre élagué (tprune)

A une erreur légèrement meilleure (8.91%), ce qui indique que l'élagage a réduit la complexité sans perte significative de performance.

Cela suggère qu'il supprime des branches inutiles (réduction du surajustement) tout en maintenant une bonne capacité de généralisation.

Forêts aléatoires

```r
# Charger la librairie randomForest
library(randomForest)

# Construire une RF (Bagging, mtry = nombre total de variables)
rf_bag <- randomForest(type ~ ., data = spam.app, mtry = ncol(spam.app) - 1)
pred_bag <- predict(rf_bag, spam.test)
err_bag <- mean(pred_bag != spam.test$type)

# RF par défaut
rf_def <- randomForest(type ~ ., data = spam.app)
pred_rf <- predict(rf_def, spam.test)
err_rf <- mean(pred_rf != spam.test$type)

# Étudier l'évolution de l'erreur OOB
rf_ntree <- randomForest(type ~ ., data = spam.app, ntree = 1000, do.trace = 100)
```

```
## ntree      OOB      1      2
##   100:   5.83%  3.80%  8.94%
##   200:   5.39%  3.30%  8.61%
##   300:   5.57%  3.44%  8.83%
##   400:   5.61%  3.30%  9.16%
##   500:   5.74%  3.52%  9.16%
##   600:   5.74%  3.52%  9.16%
##   700:   5.74%  3.44%  9.27%
##   800:   5.61%  3.37%  9.05%
##   900:   5.57%  3.37%  8.94%
##  1000:   5.65%  3.37%  9.16%
```

```r
# Afficher les erreurs
cat("Erreur bagging :", err_bag, "\n")
```

```
## Erreur bagging : 0.06736202
```

```r
cat("Erreur forêt aléatoire :", err_rf, "\n")
```

```
## Erreur forêt aléatoire : 0.04563233
```

Interprétation

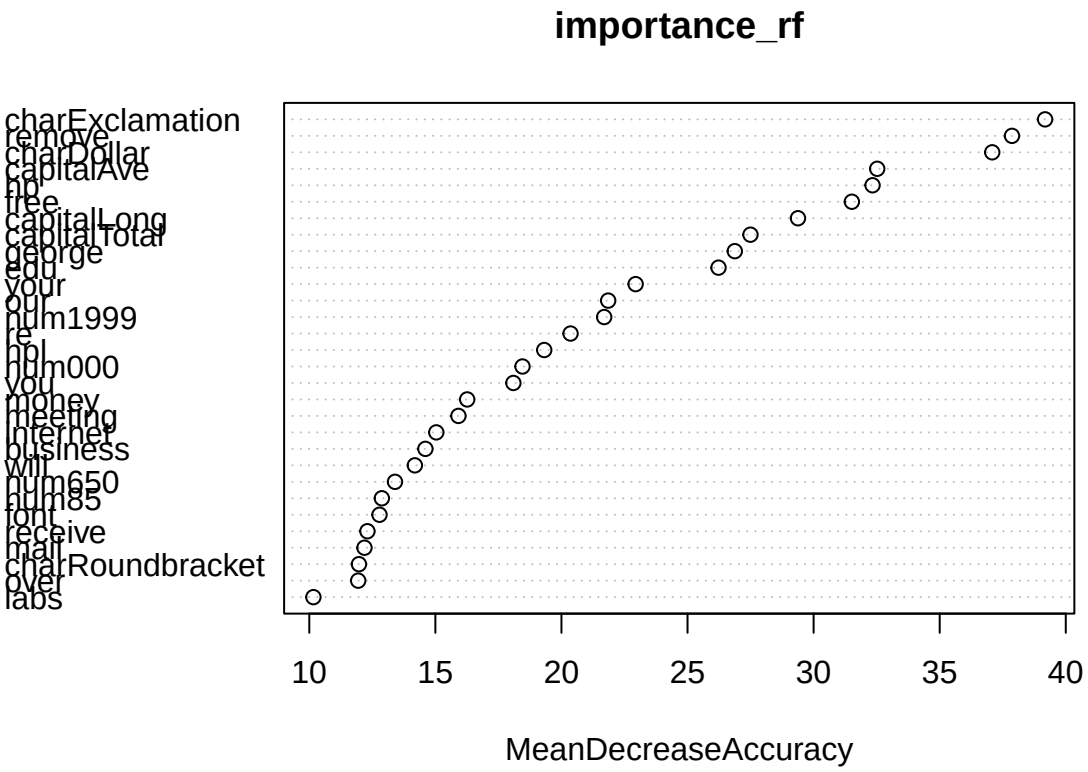
Phase de diminution initiale (0 à 200 arbres).L'erreur OOB passe de 5.83% à 5.39%, ce qui montre que l'ajout d'arbres améliore la performance.

L'erreur pour les deux classes (3.80% pour la classe 1 et 8.94% pour la classe 2) diminue progressivement.Phase de stabilisation (300+ arbres).Dès 300 arbres, l'erreur commence à fluctuer autour de 5.6%.Ajouter plus d'arbres n'apporte plus d'amélioration significative.On observe même une légère hausse de l'erreur après 500 arbres (surapprentissage possible).Effet du nombre d'arbres.Trop peu d'arbres (<100) : Risque de sous-ajustement, la forêt n'a pas encore appris assez d'informations.300-500 arbres : Bonne stabilité, compromis entre performance et coût de calcul. 1000 arbres : Pas de gain significatif par rapport à 500 arbres, mais temps de calcul plus long.

Comparaison des erreurs Bagging vs Forêt Aléatoire: Le bagging a une erreur plus élevée (6.74%).Le bagging construit plusieurs arbres en utilisant un bootstrap, mais chaque arbre utilise toutes les variables à chaque nœud. Résultat : Les arbres restent corrélés, ce qui limite l'amélioration de la performance.La forêt aléatoire améliore la performance (4.56%).Contrairement au bagging, la forêt aléatoire sélectionne aléatoirement un sous-ensemble de variables à chaque nœud.Cela réduit la corrélation entre les arbres, rendant le modèle plus robuste et performant.La réduction de l'erreur de 6.74% à 4.56% montre que cette diversification améliore la capacité de généralisation.

Importance des variables

```r
# Importance des variables
importance_rf <- randomForest(type ~ ., data = spam.app, importance = TRUE)
varImpPlot(importance_rf, type = 1)
```



Interprétation

Nous identifions les variables les plus importantes dans la classification des spams. Certaines caractéristiques, comme le nombre de **caractères spéciaux (ex: charDollar, charExclamation)**, influencent fortement la prédiction.

Interprétation des résultats

Les résultats montrent que : - Les **forêts aléatoires** obtiennent une **meilleure performance** que les arbres de décision seuls. - L'**élagage des arbres** améliore légèrement la précision en réduisant le sur-apprentissage. - Certaines variables comme **charDollar, remove, charExclamation** sont très importantes pour la classification.

Conclusion

L'utilisation des **forêts aléatoires** est plus efficace que les arbres classiques pour identifier les spams. De plus, la sélection des variables clés permet d'améliorer la robustesse du modèle.