Bagging – Random Forest - Boosting

Techniques ensemblistes pour l'analyse prédictive

Jimmy Vandel
jimmy.vandel@univ-lille.fr

Ricco Rakotomalala
Université Lumière Lyon 2
http://tutoriels-data-mining.blogspot.com/

Principes des techniques ensemblistes

- Faire coopérer plusieurs modèles
- Procédure de vote simple/pondéré lors du classement

2 grandes stratégies :

- Construire différents types de modèles sur les mêmes données
- Construire des modèles de même nature sur des versions différentes des données

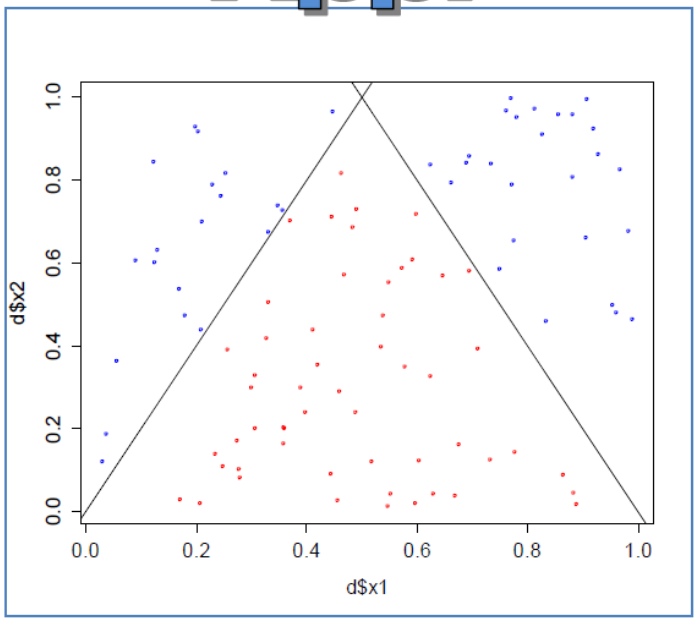
➢ Il faut que les modèles se **complètent** pour bénéficier d'une coopération.
S'ils classent tous de la même manière → le gain est nul !

❖ **On se place ici exclusivement dans une problématique de classification**

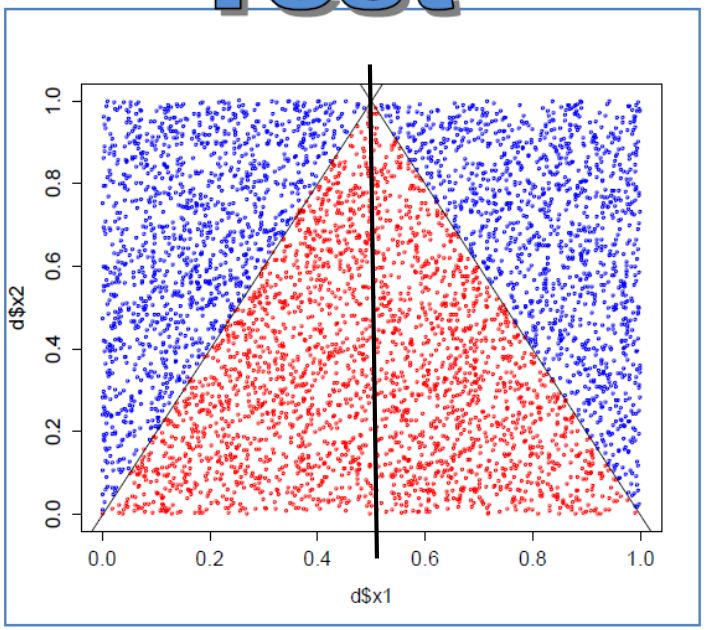
Données utilisées :

- n_{app}= 100 individus pour apprendre
- n_{test}= 500 pour l'évaluation
- p= 10 variables prédictives (uniquement 2 utiles)
- $\{x_1, x_2\}$
- Classification en 2 classes, sans bruit

App.



Test

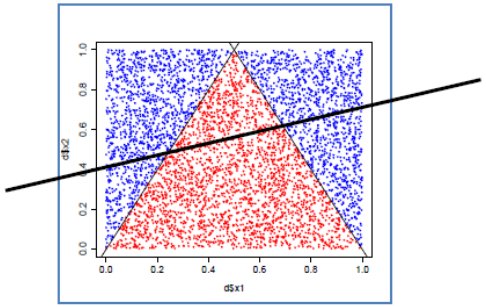


Y = «bleu»
si x2>(2*x1) pour x1<0.5
ou x2>(2−2*x1) pour x1≥0.5
«rouge» autrement

Composantes de l'erreur :

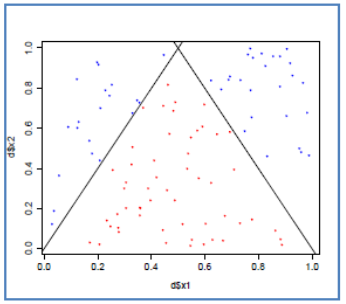
On construit un modèle à partir d'un échantillon que l'on souhaite performant sur la population. Deux composantes pèsent sur l'erreur de prédiction :

- **Biais** : incapacité du modèle à traduire le concept reliant Y à X



Un classifieur linéaire ne peut pas fonctionner ici. Impossible de trouver une droite permettant de séparer les points bleus des rouges.

- **Variance** : sensibilité aux fluctuations d'échantillonnage (aux données de prédiction)



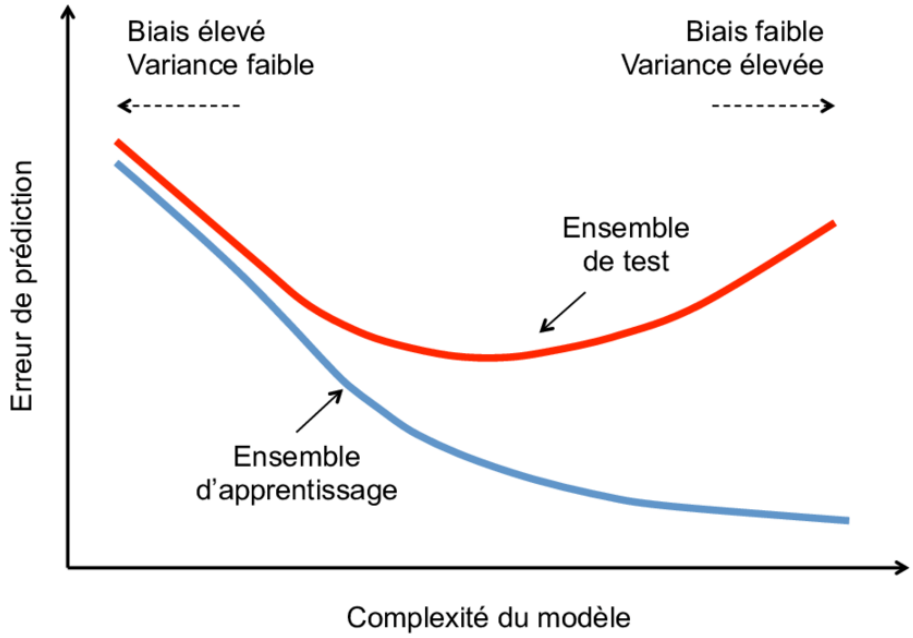
Le faible effectif de l'échantillon d'apprentissage ne permet pas de trouver avec exactitude les «bonnes» frontières.

Composantes de l'erreur :

Les modèles «simples» (ex. linéaires, peu de paramètres à estimer) présentent un biais fort, mais une faible variance

VS

Les modèles «complexes» (nombreux paramètres à estimer) présentent un biais faible, mais une forte variance



Sans oublier le rôle perturbateur que peuvent avoir les variables qui n'ont aucun rapport avec le problème étudié (récurent en grande dimension)

PLAN

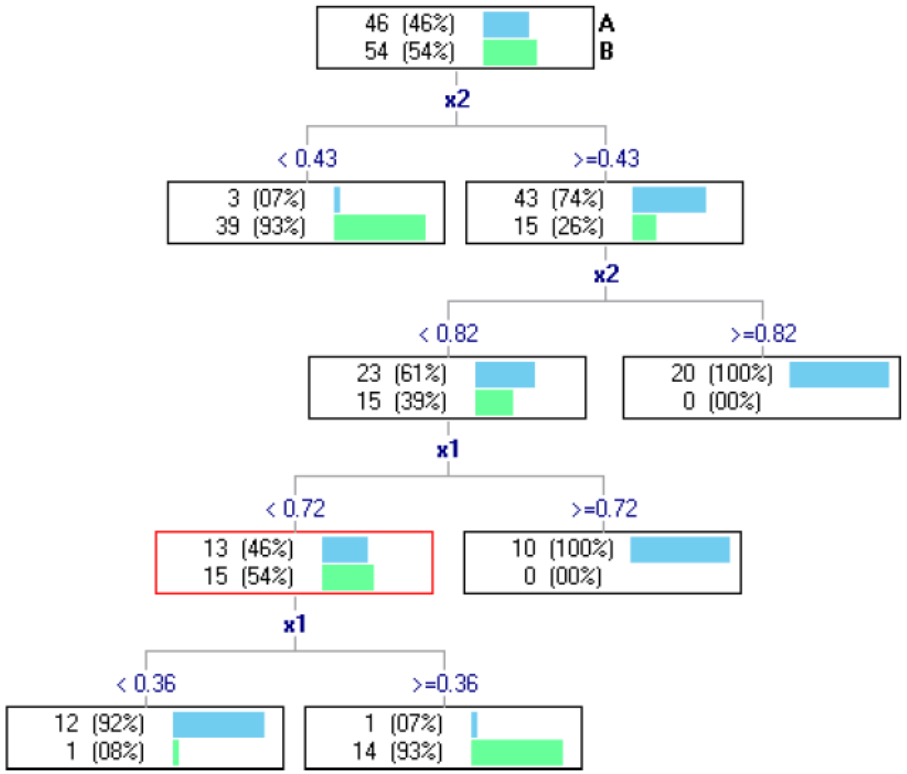
1. Arbres de décision
2. Bagging
3. Random Forest
4. Boosting
5. Bilan

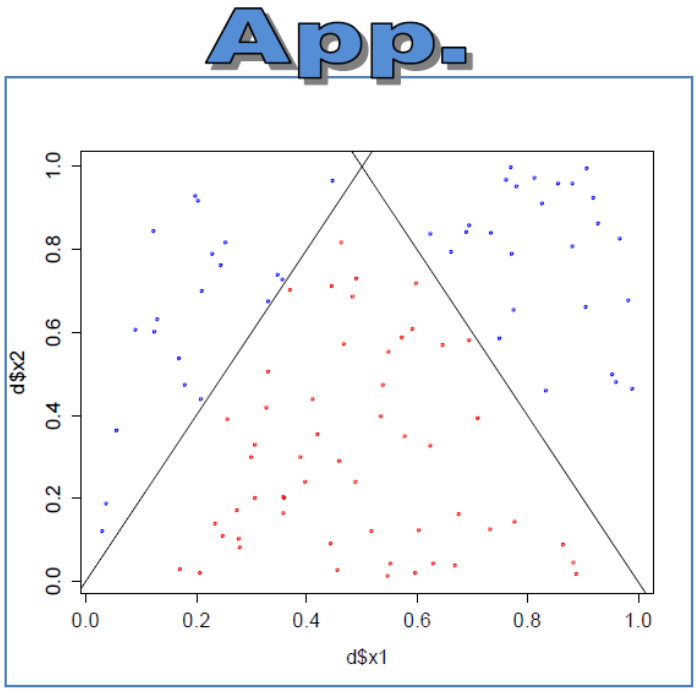
PLAN

1. **Arbres de décision**
2. Bagging
3. Random Forest
4. Boosting
5. Bilan

Arbres de décision

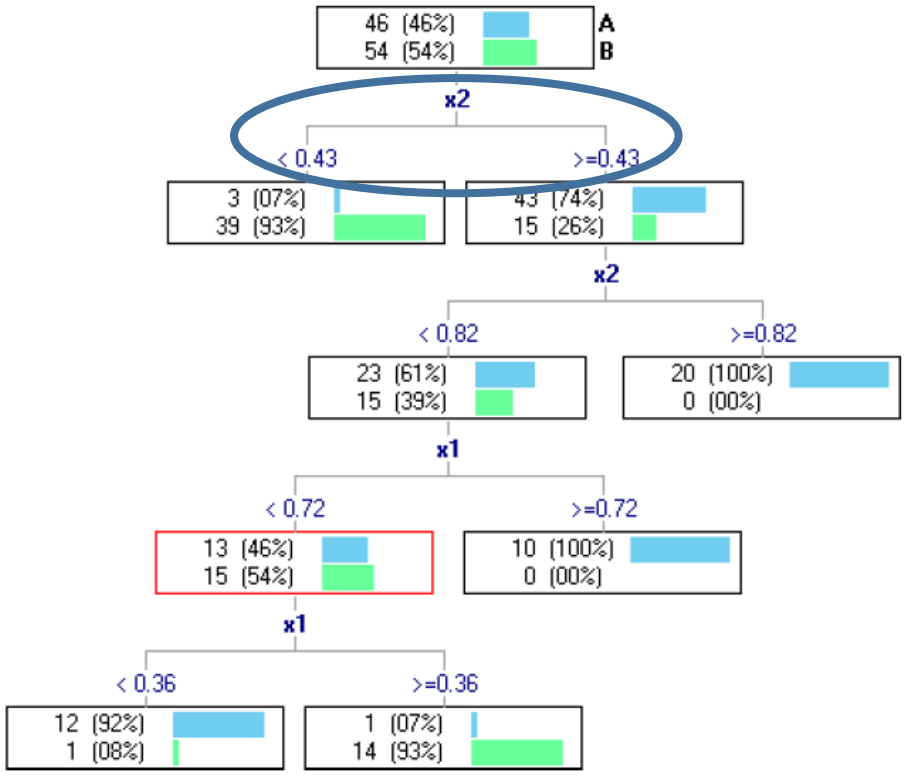
→ Algorithme récursif de découpage de l'espace de représentation.
Découpage forcément parallèle aux axes.
Modèle linéaire par morceaux, non-linéaire au final.

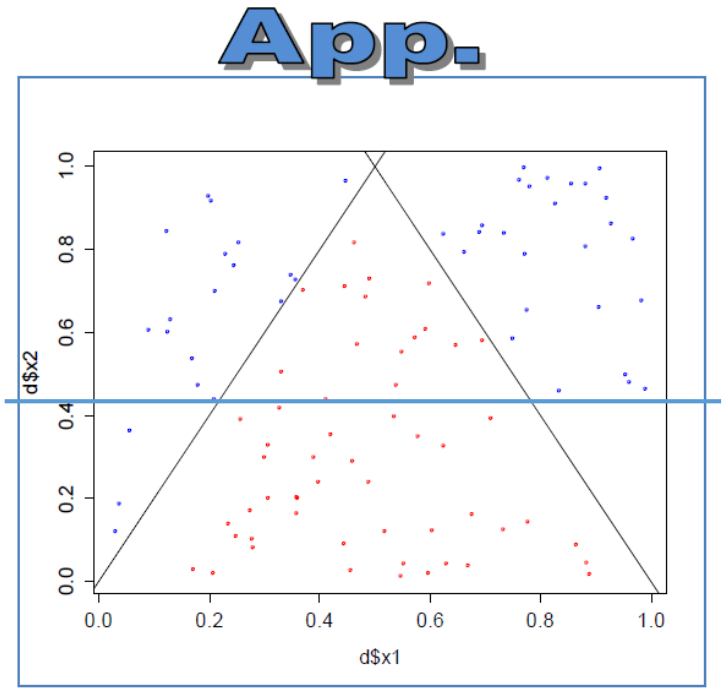




Arbres de décision

→ Algorithme récursif de découpage de l'espace de représentation.
Découpage forcément parallèle aux axes.
Modèle linéaire par morceaux, non-linéaire au final.



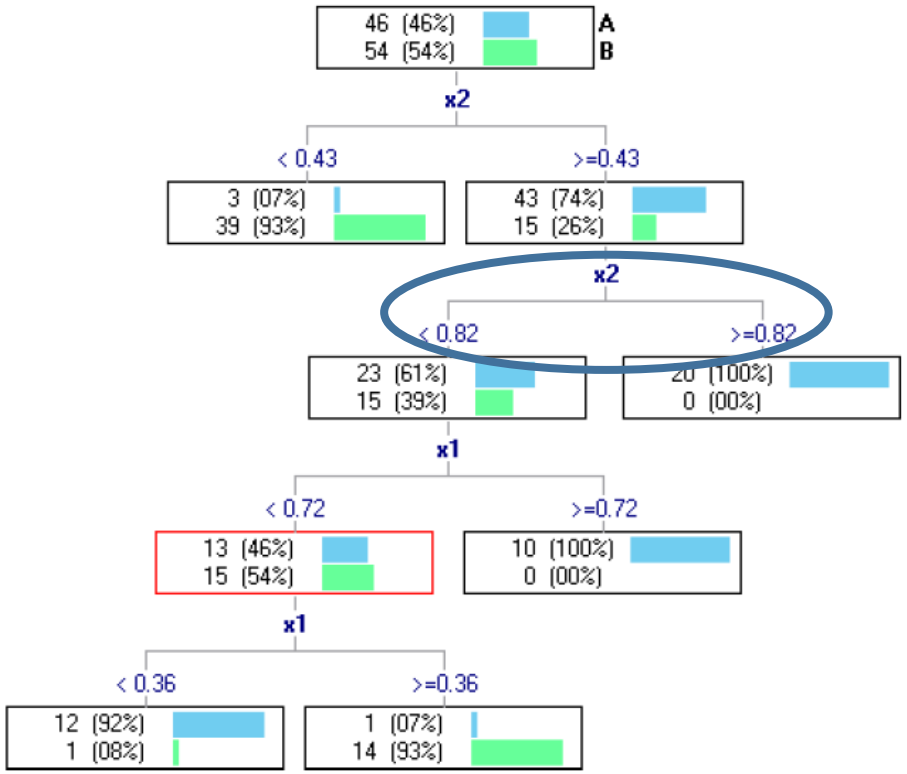


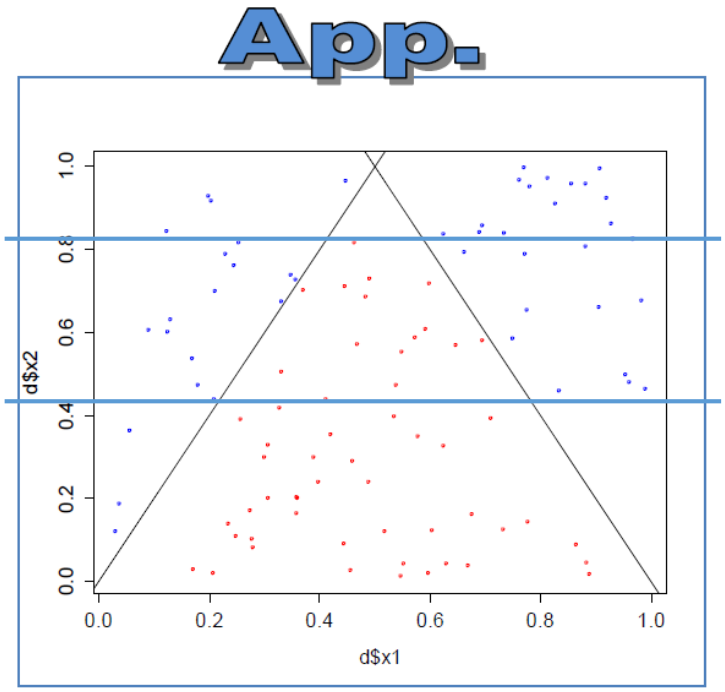
Arbres de décision

Algorithme récursif de découpage de l'espace de représentation.

Découpage forcément parallèle aux axes.

Modèle linéaire par morceaux, non-linéaire au final.



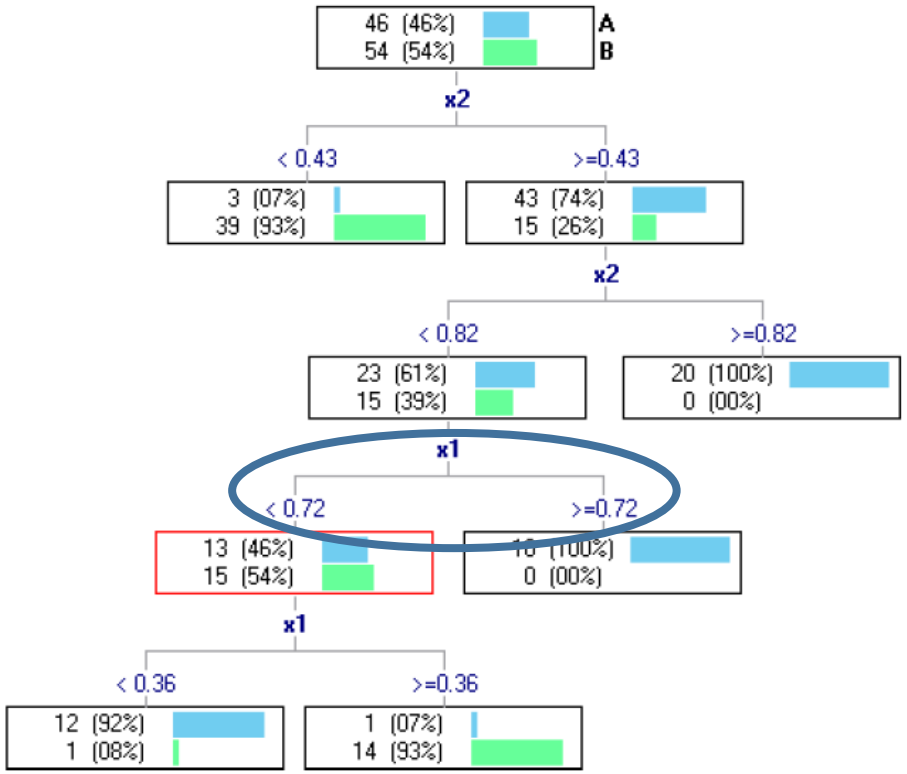


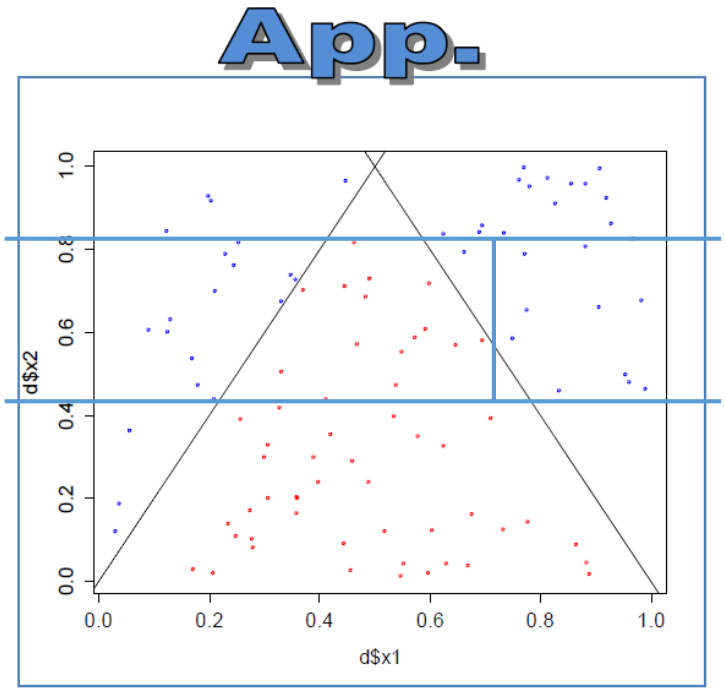
Arbres de décision

Algorithme récursif de découpage de l'espace de représentation.

Découpage forcément parallèle aux axes.

Modèle linéaire par morceaux, non-linéaire au final.



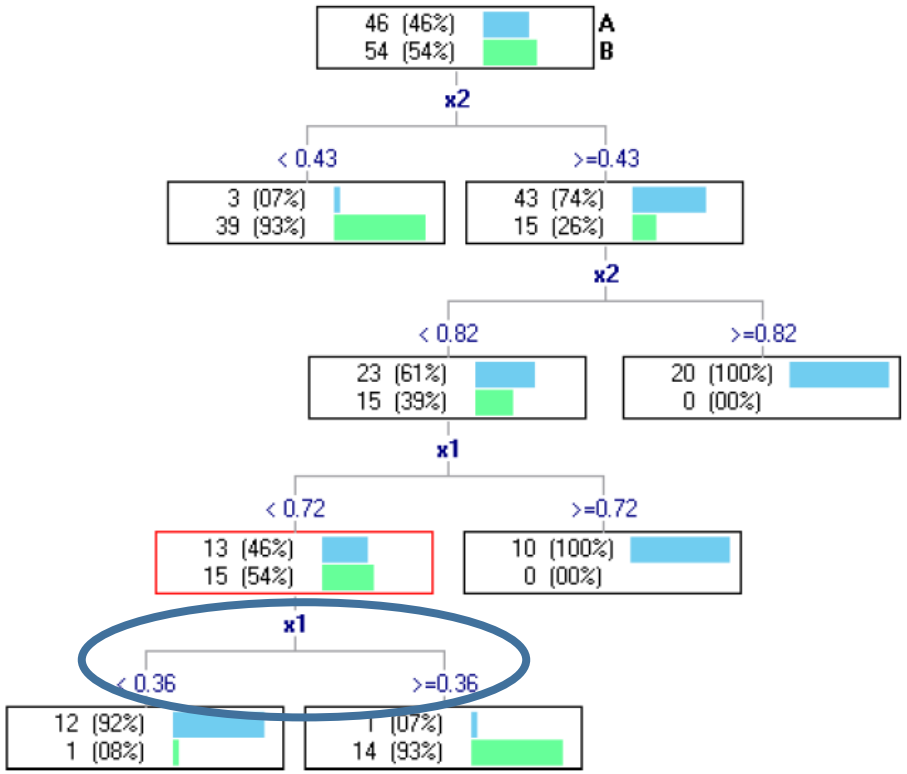


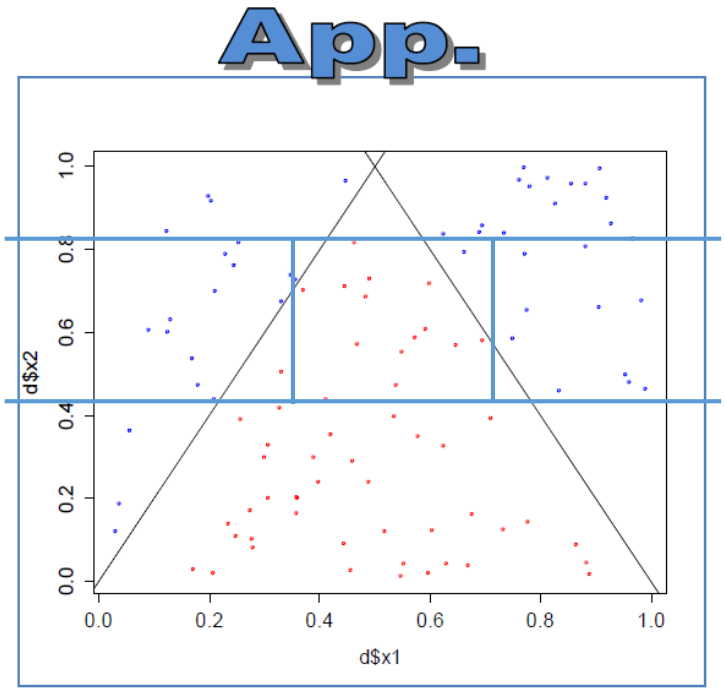
Arbres de décision

Algorithme récursif de découpage de l'espace de représentation.

Découpage forcément parallèle aux axes.

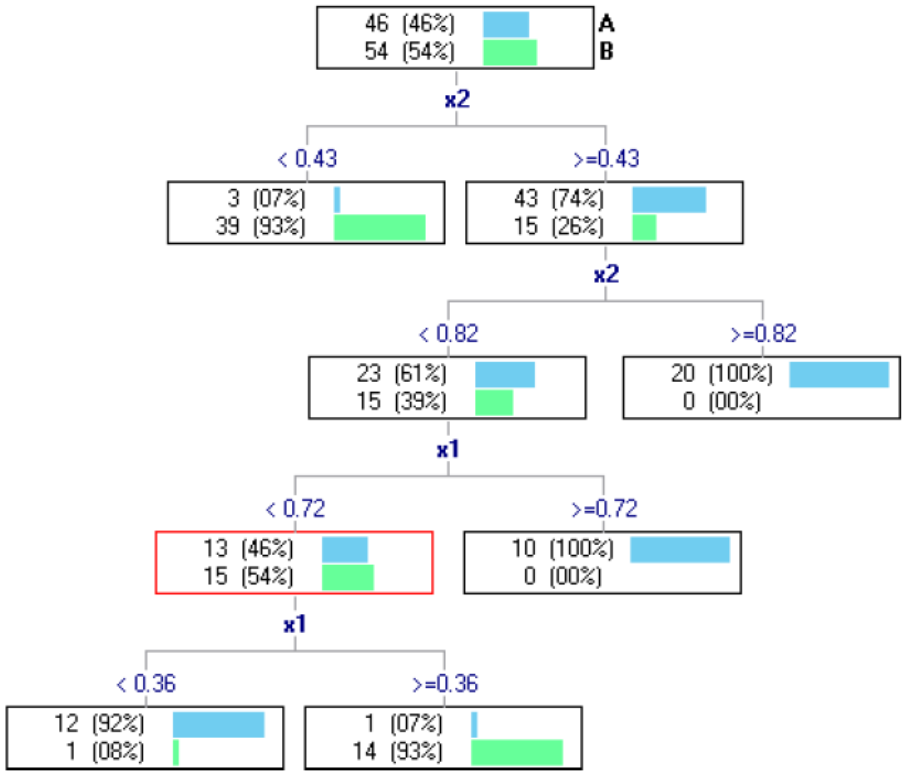
Modèle linéaire par morceaux, non-linéaire au final.

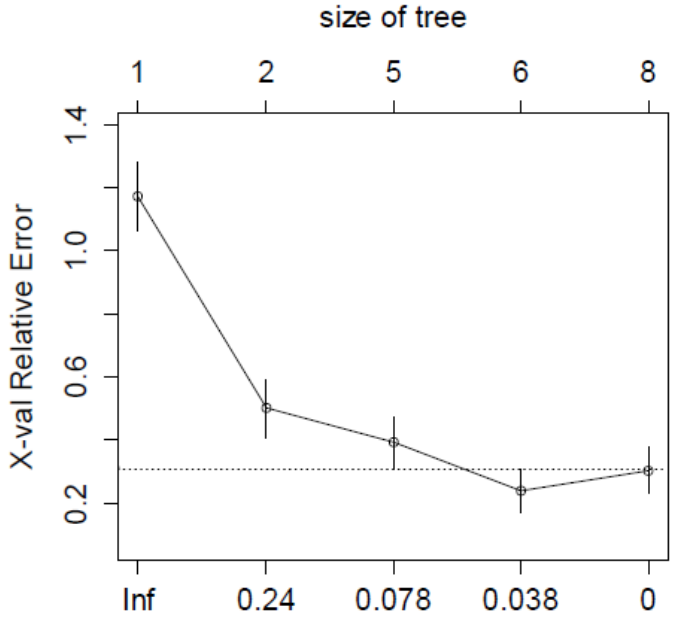




Arbres de décision

Algorithme récursif de découpage de l'espace de représentation.
Découpage forcément parallèle aux axes.
Modèle linéaire par morceaux, non-linéaire au final.





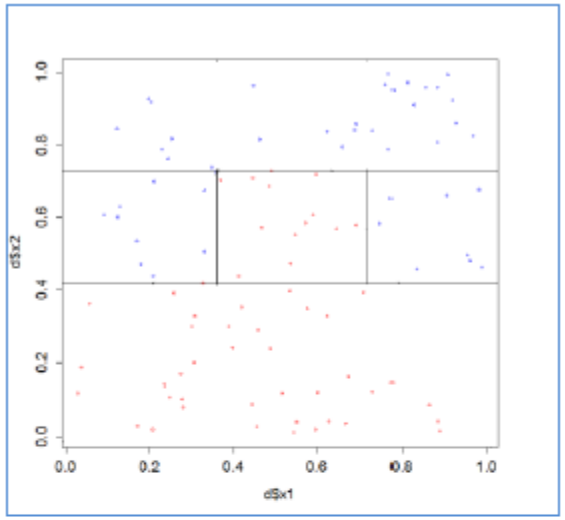
→ détecter la profondeur «optimale»

Arbre profond : biais faible, variance forte
Arbre court : biais fort, variance faible

Arbres de décision

```
library(rpart)
arbre <- rpart(y ~ ., data = d.train)
print(arbre)
```

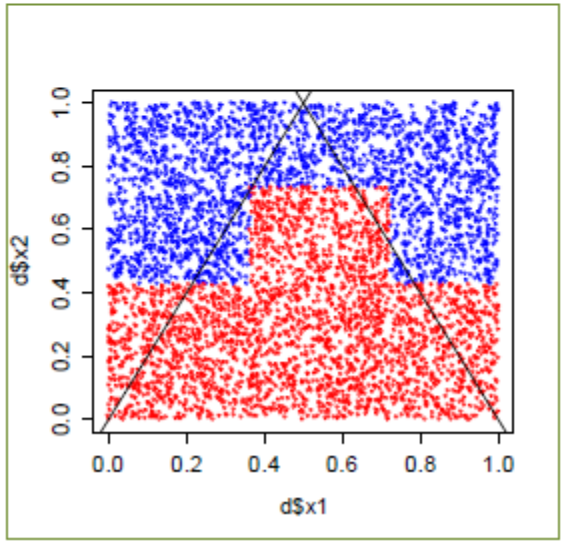
```
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 100 46 2 (0.46000000 0.54000000)
   2) x2>=0.4271354 58 15 1 (0.74137931 0.25862069)
     4) x2>=0.733562 27  1 1 (0.96296296 0.03703704) *
     5) x2< 0.733562 31 14 1 (0.54838710 0.45161290)
      10) x1>=0.7221232 8  0 1 (1.00000000 0.00000000) *
      11) x1< 0.7221232 23  9 2 (0.39130435 0.60869565)
        22) x1< 0.3639227 10  1 1 (0.90000000 0.10000000) *
        23) x1>=0.3639227 13  0 2 (0.00000000 1.00000000) *
   3) x2< 0.4271354 42  3 2 (0.07142857 0.92857143) *
```



TRAIN

(5 feuilles dans l'arbre = 5 zones sont définies)

→ La modélisation est contrainte par le nombre d'observations disponibles dans TRAIN



TEST

$\varepsilon = 0.1632$

Arbres de décision

```
library(rpart)
arbre <- rpart(y ~ ., data = d.train)
print(arbre)
```

```
n= 100

node), split, n, loss, yval, (yprob)
      * denotes terminal node

 1) root 100 46 2 (0.46000000 0.54000000)
   2) x2>=0.4271354 58 15 1 (0.74137931 0.25862069)
     4) x2>=0.733562 27  1 1 (0.96296296 0.03703704) *
     5) x2< 0.733562 31 14 1 (0.54838710 0.45161290)
      10) x1>=0.7221232 8  0 1 (1.00000000 0.00000000) *
      11) x1< 0.7221232 23  9 2 (0.39130435 0.60869565)
        22) x1< 0.3639227 10  1 1 (0.90000000 0.10000000) *
        23) x1>=0.3639227 13  0 2 (0.00000000 1.00000000) *
   3) x2< 0.4271354 42  3 2 (0.07142857 0.92857143) *
```

→ La modélisation est contrainte par le nombre d'observations disponibles dans TRAIN

Si on tente de forcer les segmentations, des «mauvaises» variables peuvent s'insérer

```
arbre.p <- rpart(y ~ ., data = d.train,
control=list(cp=0,minsplit=2,minbucket=1))
print(arbre.p)
```

```
1) root 100 46 2 (0.46000000 0.54000000)
  2) x2>=0.4271354 58 15 1 (0.74137931 0.25862069)
    4) x2>=0.733562 27  1 1 (0.96296296 0.03703704)
      8) x4>=0.1231597 26  0 1 (1.00000000 0.00000000) *
      9) x4< 0.1231597 1  0 2 (0.00000000 1.00000000) *
    5) x2< 0.733562 31 14 1 (0.54838710 0.45161290)
     10) x1>=0.7221232 8  0 1 (1.00000000 0.00000000) *
     11) x1< 0.7221232 23  9 2 (0.39130435 0.60869565)
       22) x1< 0.3639227 10  1 1 (0.90000000 0.10000000)
         44) x4>=0.02095698 9  0 1 (1.00000000 0.00000000) *
         45) x4< 0.02095698 1  0 2 (0.00000000 1.00000000) *
       23) x1>=0.3639227 13  0 2 (0.00000000 1.00000000) *
  3) x2< 0.4271354 42  3 2 (0.07142857 0.92857143)
    6) x1< 0.1139017 3  0 1 (1.00000000 0.00000000) *
    7) x1>=0.1139017 39  0 2 (0.00000000 1.00000000) *
```

Arbres de décision

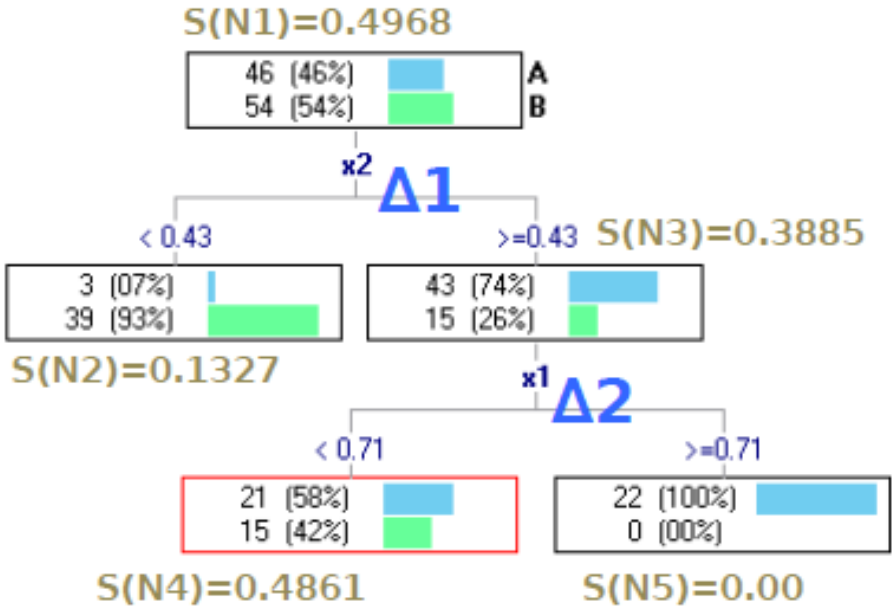
Décomposition du gain d'information

Indice de Gini

- Indice de concentration
- Mesure d'impureté
- Entropie quadratique

$$S(noeud) = \sum_{k=1}^{K} p_k (1 - p_k)$$

$$S(N1) = \frac{46}{100}\left(1 - \frac{46}{100}\right) + \frac{54}{100}\left(1 - \frac{54}{100}\right) = 0.4968$$

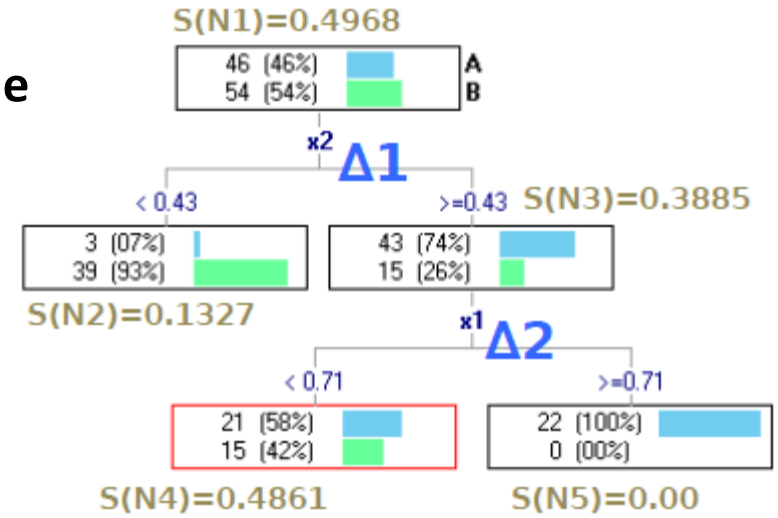


Arbres de décision

Décomposition du gain d'information

Information (variance) expliquée par le modèle

$$SCE = SCT - SCR$$



$$SCT = S(N1) = \frac{46}{100}\left(1-\frac{46}{100}\right)+\frac{54}{100}\left(1-\frac{54}{100}\right) = 0.4968$$

$$SCR = \frac{42}{100}\times S(N2)+\frac{36}{100}\times S(N4)+\frac{22}{100}\times S(N5)$$
$$= \frac{42}{100}\times 0.1327+\frac{36}{100}\times 0.4861+\frac{22}{100}\times 0.00 = 0.2307$$

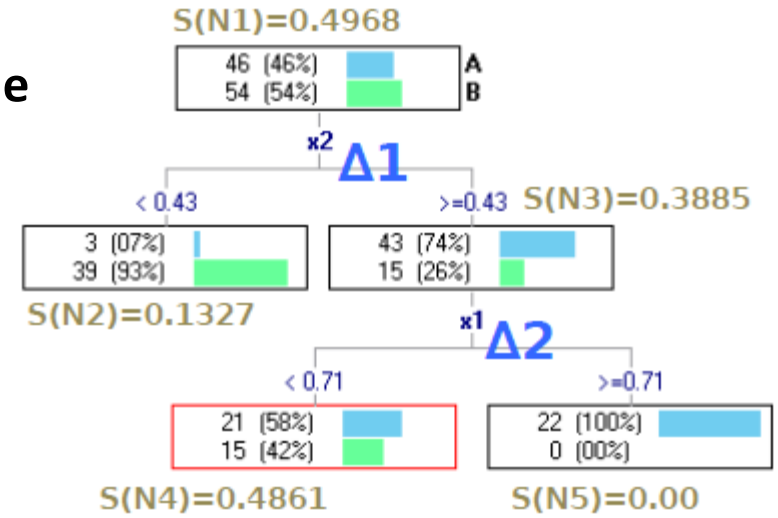
$$SCE = SCT - SCR = 0.4968 - 0.2307 = 0.2661$$

Arbres de décision

Décomposition du gain d'information

Information (variance) expliquée par le modèle

→ Elle peut être décomposée selon les segmentations dans l'arbre



$$\Delta 1 = \frac{42+58}{100}\left\{S(N1) - \left[\frac{42}{100}\times S(N2) + \frac{58}{100}\times S(N3)\right]\right\} = 0.2187$$

$$\Delta 2 = \frac{36+22}{100}\left\{S(N3) - \left[\frac{36}{58}\times S(N4) + \frac{22}{58}\times S(N5)\right]\right\} = 0.0474$$

SCE=Δ1+Δ2

Le gain Δ induit par chaque segmentation peut être replacé dans le contexte de la qualité globale du modèle SCE. L'importance diminue à mesure qu'on s'éloigne de la racine (parce que le poids du sommet est faible).

Arbres de décision

Bilan

Avantages

- connaissances «intelligibles» --validation d'expert (si arbre pas trop grand)
- traduction directe de l'arbre vers une base de règles
- sélection automatique des variables pertinentes, robuste face aux variables redondantes

- non paramétrique
- traitement indifférencié des différents types de variables prédictives
- robuste face aux données aberrantes, solutions pour les données manquantes
- rapidité et capacité à traiter des très grandes bases

- enrichir l'interprétation des règles à l'aide des variables non sélectionnées
- possibilité pour le praticien d'intervenir dans la construction de l'arbre

Inconvénients

- problème de stabilité sur les petites bases de données
- recherche «pas-à-pas» : difficulté à trouver certaines interactions
- peu adapté au «scoring»
- performances moins bonnes en général par rapport aux autres méthodes

PLAN

1. Arbres de décision
2. **Bagging**
3. Random Forest
4. Boosting
5. Bilan

Bagging

Bootstrap Aggregating

→ Faire coopérer (voter) B arbres construits sur des échantillons bootstrap. B est un paramètre de l'algorithme.

Apprentissage

Entrée : B nombre de modèles, ALGO algorithme d'apprentissage, Ω un ensemble de données de taille n avec y cible à K modalités, X avec p prédicteurs.

MODELES = {}
Pour b = 1 à B Faire
  Tirage avec remise d'un échantillon de taille n → Ω_b
  Construire un modèle M_b sur Ω_b avec ALGO
  Ajouter M_b dans MODELES
Fin Pour

Classement

Pour un individu i* à classer,
  Appliquer chaque modèle M_b de MODELES → $\hat{y}_b(i*)$

Prédiction bagging → $\hat{y}_{bag}(i*) = \arg\max_k \left[\sum_{b=1}^{B} I(\hat{y}_b(i*) = y_k) \right]$

→ Ce qui correspond à un vote à la majorité simple

Bagging

Pourquoi ?

Intérêt de la coopération. Faire coopérer des modèles n'a d'intérêt que si les modèles ne classent pas tous de la même manière (si le vote est systématiquement unanime, autant n'avoir qu'un seul modèle).

Biais et variance. Biais du bagging = biais du modèle sous-jacent.
Le bagging réduit avant tout la variance. Il faut donc que les modèles sous-jacents aient un biais faible, capturant la complexité des relations entre Y et X.

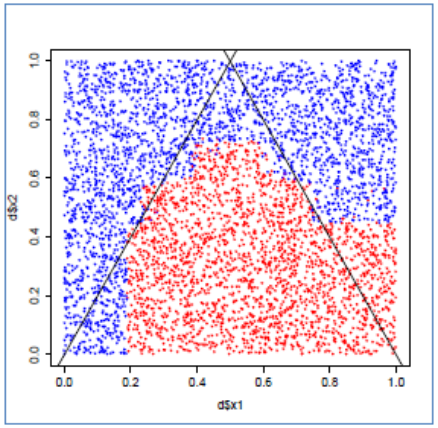
Apprenants faibles. Le bagging ne sait pas tirer parti des apprenants faibles. Il faut que les modèles sous-jacents soient de bonne qualité.

Sur-apprentissage (overfitting). Augmenter B n'aboutit pas au sur-apprentissage. En pratique, une centaine suffit, mais on peut l'ajuster à l'étude.

Arbres de décision. Le bagging peut s'appliquer à tout type de modèle. Les arbres sont un bon modèle : réduire le biais individuel en créant des arbres profonds, la variance (de chaque arbre) est alors compensée par la coopération.

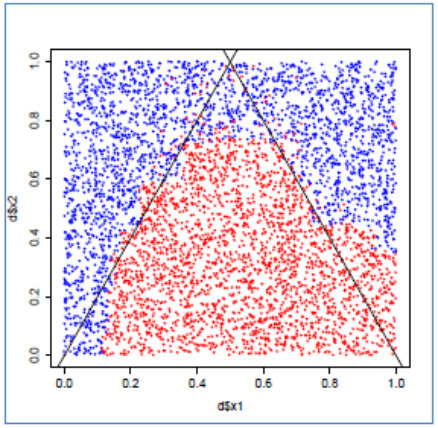
Bagging

Et sur nos données ?




```
#100 arbres générés par bagging
model.bagging <- bagging(y ~ ., data = d.train,
mfinal=100)
```

$\varepsilon = 0.156$




```
model.bagging.2 <- bagging(y ~ ., data = d.train,
mfinal=100,control=list(cp=0,minsplit=2,minbucket=1))
```

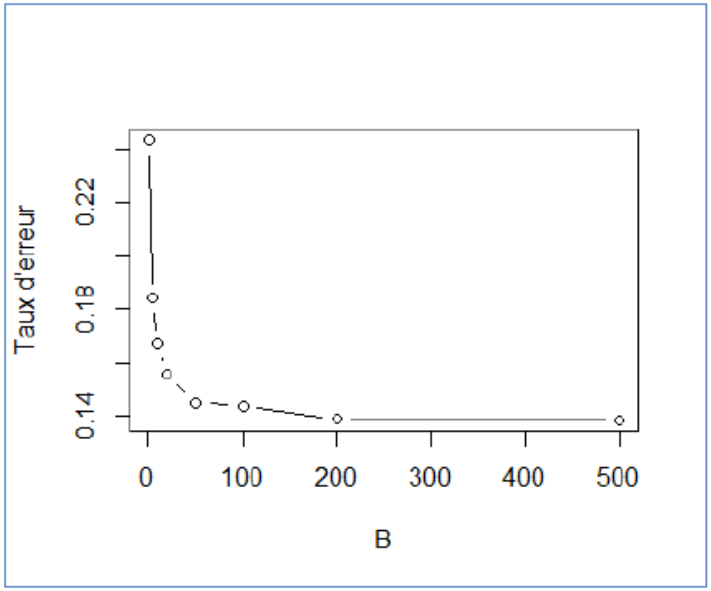
$\varepsilon = 0.1224$

→ En créant des arbres individuels plus profonds (moins biaisés), on améliore la qualité de la prédiction du bagging.

Bagging

Impact du nombre de modèles (B) sur la qualité de la prédiction

```
#bagging et nombre de réplications
B <- c(1,5,10,20,50,100,200,500)
#une session pour un nombre d'arbres égal à b
une_session <- function(b){
  model.bagging <- bagging(y ~ ., data = d.train, mfinal=b)
  p.bagging <- predict(model.bagging,newdata=d.test)
  return(erreur(d.test$y,p.bagging$class))
}
#mesurer l'erreur en répétant l'opération 20 fois
errors <- replicate(20,sapply(B,une_session))
m.errors <- apply(errors,1,mean)
plot(B,m.errors,xlab="B",ylab="Taux d'erreur",type="b")
```



→ Au-delà d'un certain nombre de réplications, il n'y a plus d'amélioration notable.

→ Mais il n'y a pas de dégradation non plus (pas d'overfitting).

Bagging

Classer un nouvel individu

Dans une classification Y ∈ {+, -}, on a besoin d'une estimation viable de la quantité P(Y = + / X).

Solution 1 Utiliser la fréquence des votes pour estimer la probabilité a posteriori :

$$\hat{P}(Y = + / X) = \frac{\sum_{b=1}^{B} I(\hat{y}_b = +)}{B}$$

Solution 2 Si le modèle M_b fournit une estimation de la probabilité P_b d'appartenance à la classe « + », on peut alors définir :

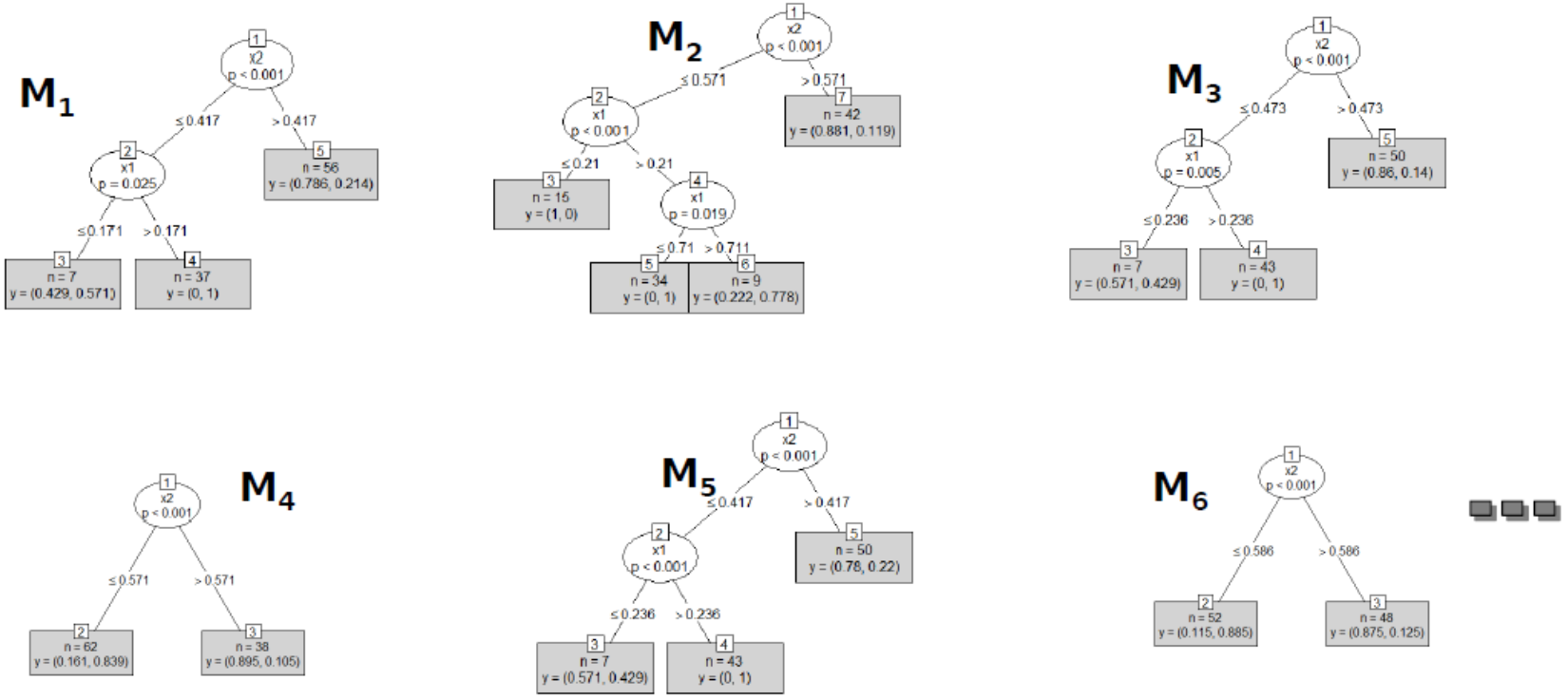
$$\hat{P}(Y = + / X) = \frac{\sum_{b=1}^{B} \hat{P}_b(Y = + / X)}{B}$$

Rq: Solution 2 est préférable, d'autant plus lorsque B est faible.

Bagging

Importance des variables

- On a une multitude d'arbres, on n'a plus un modèle «lisible» directement. L' interprétation du modèle prédictif devient difficile.



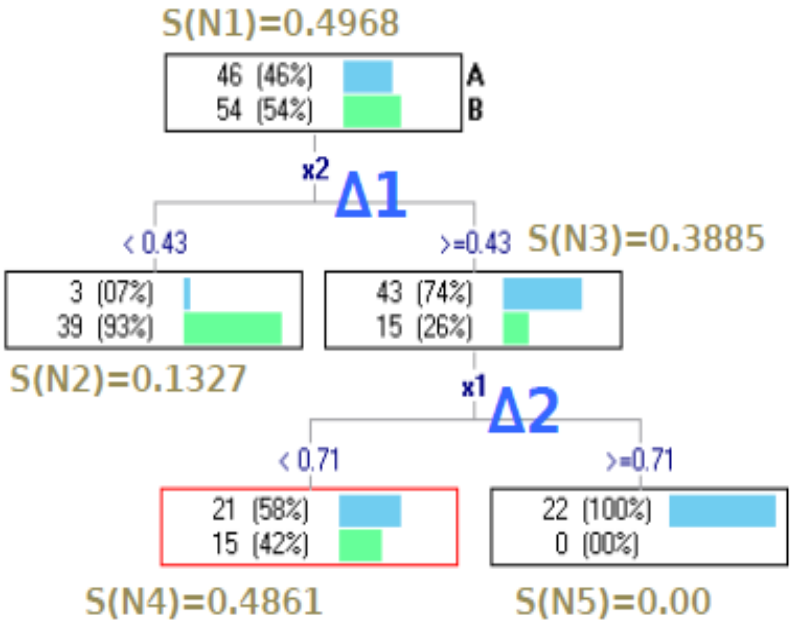
→ On ne peut pas inspecter chaque arbre pour distinguer les variables qui jouent un rôle, et dans quelle mesure elles jouent un rôle.

Bagging

Importance des variables

- Exploiter la mesure d'importance lors de la segmentation dans l'arbre.

- Additionner les contributions des variables dans les arbres pour lesquels elles apparaissent.

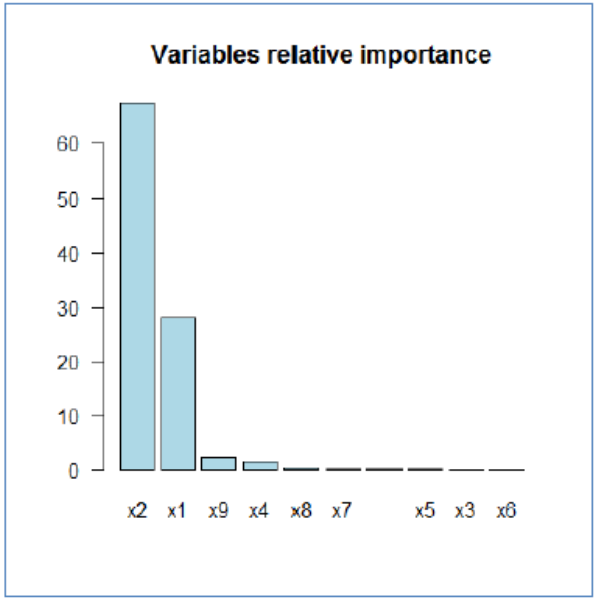


Bagging

Importance des variables

- Exploiter la mesure d'importance lors de la segmentation dans l'arbre.

- Additionner les contributions des variables dans les arbres pour lesquels elles apparaissent.




```
#importance des variables
#library(adabag)
importanceplot(model.bagging)
```

→ Exemple pour nos données où l'on «sait» que X2 et X1 sont pertinentes, X3 à X10 en revanche n'ont aucun impact dans la séparabilité des classes.

Bagging

Estimation de l'erreur OOB

→ Estimation de l'erreur OOB (out-of-bag error estimation) : mesurer l'erreur directement durant l'apprentissage, sans avoir à passer par un échantillon test ou par une technique de type validation croisée.

Pour l'arbre M_b, certains individus sont tirés plusieurs fois, d'autres ne sont pas inclus dans l'échantillon bootstrap → prédire ces individus avec le modèle M_b

arbres

individus

i	M_1	M_2	M_3	M_4	M_5	Y^	Y	ERR
1	.	+	.	+	–	+	+	0
2	.	.	.	–	–	–	–	0
3	+	.	–	–	–	–	+	1
4	.	+	+	.	.	+	+	0
5	+	–	–	.	.	–	+	1
6	.	.	.	.	+	+	–	1
7	–	.	.	–	.	–	–	0
8	–	+	.	.	+	+	+	0
9	.	.	+	.	.	+	+	0
10	+	.	.	+	.	+	–	1
...								

Prédiction OOB par vote à la majorité des modèles (dans la ligne).

B=5 arbres
Y={+, -}

`.` : individu utilisé
+ : OOB prédit +
- : OOB prédit -

ERR_{OOB} = Proportion(ERR)

Err_{OOB} est une estimation viable de l'erreur en prédiction du modèle bagging.

Bagging

Notion de marge

→ Correspond à l'écart entre la proportion de vote pour la bonne classe et le maximum de vote pour une autre classe.
Rq : pour un problème binaire {+ , -}, c'est l'écart entre la probabilité d'appartenance à la bonne classe moins le complémentaire à 1.

Pour un individu ***i*** dans la classe ***k****

$$m(\omega) = \frac{\sum_{b=1}^{B} I(\hat{y}_b(i) = k^*)}{B} - \max_{k \neq k^*} \left[\frac{\sum_{b=1}^{B} I(\hat{y}_b(i) = k)}{B} \right]$$

Augmenter cette marge → décision plus tranchée pour le classement (moins de variance)

Les techniques ensemblistes visent à augmenter la marge

PLAN

1. Arbres de décision
2. Bagging
3. **Random Forest**
4. Boosting
5. Bilan

Random forest

Un constat

Le bagging est, de manière générale, dominé par le boosting. Breiman propose une alternative au boosting . Il proposer en 2001 les forêts aléatoires qui s'avèrent aussi bon en général que le boosting.

Pour que le bagging soit efficace, il faut :

- Des arbres individuellement performants
- De grande profondeur (taille min. des feuilles = 1) -> faible biais
- Fortement différents les uns des autres pour qu'ils puissent se compléter → «décorrélation» des arbres.

➢ Introduire une perturbation «aléatoire» dans la construction des arbres, en jouant sur le mécanisme de sélection des variables de segmentation sur les nœuds.

Random forest

Apprentissage

Entrée : B nombre de modèles, ALGO algorithme d'apprentissage d'arbre, Ω un ensemble de données de taille n avec y cible à K modalités, X avec p prédicteurs, m nombre de variables à traiter sur chaque nœud avec, par défaut, $m = \sqrt{p}$

MODELES = {}
Pour b = 1 à B Faire
  Tirage avec remise d'un échantillon de taille n → Ω_b
  Construire un arbre M_b sur Ω_b avec ALGO
    Pour chaque segmentation :
      Choisir m variables au hasard parmi p
      Segmenter avec la meilleure variable parmi m
  Ajouter M_b dans MODELES
Fin Pour

Classement

Pour un individu **i*** à classer,
  Appliquer chaque modèle M_b de MODELES → $\hat{y}_b(i^*)$

Prédiction Random Forest ➔ $\hat{y}_{rf}(i^*) = \arg\max_k \left[\sum_{b=1}^{B} I(\hat{y}_b(i^*) = y_k) \right]$

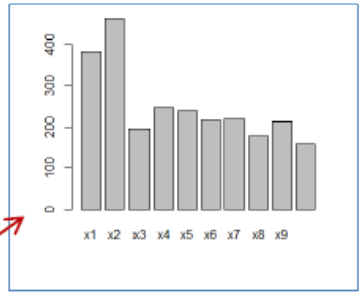
➔ Ce qui correspond à un vote à la majorité simple

Random forest

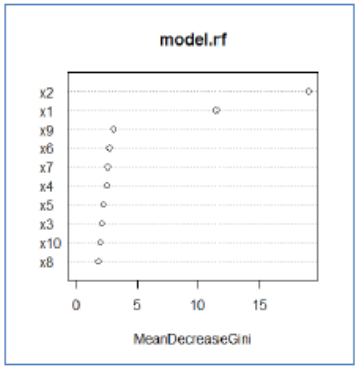
Exemple sous R

```
#random forest - B = 200 arbres
library(randomForest)
model.rf <- randomForest(y ~ ., data = d.train, ntree = 200)

#liste des variables utilisées
barplot(varUsed(model.rf),names.arg=paste("x",1:10,sep=""))

#importance des variables
varImpPlot(model.rf)

#matrice de confusion OOB
print(mc <- model.rf$confusion)

#taux d'erreur OOB
print((mc[2,1]+mc[1,2])/sum(mc[1:2,1:2])) # (7+8)/100 = 0.15

#prédiction sur échantillon test
p.rf <- predict(model.rf,newdata = d.test,type="class")

#taux d'erreur
print(erreur(d.test$y,p.rf)) # 0.149

#représentation des frontières dans (x1,x2)
nuage(d.test,p.rf)
```

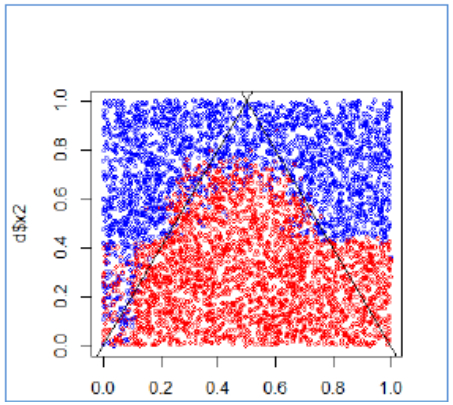
```
   1  2 class.error
1 38  8   0.1739130
2  7 47   0.1296296
```



Une variable peut apparaître plusieurs fois dans un arbre.



Une autre piste basée sur les OOB existe pour estimer l'importance.



Random forest

Détails sur les arbres

	left daughter	right daughter	split var	split point	status	prediction
1	2	3	1	0.7333904	1	0
2	4	5	7	0.5242057	1	0
3	6	7	8	0.6437400	1	0
4	8	9	3	0.3753580	1	0
5	10	11	7	0.7430234	1	0
6	12	13	5	0.8513703	1	0
7	0	0	0	0.0000000	-1	1*
8	14	15	1	0.1846170	1	0
9	16	17	6	0.4151308	1	0
10	0	0	0	0.0000000	-1	2*
11	18	19	6	0.4038910	1	0
12	0	0	0	0.0000000	-1	1*
13	0	0	0	0.0000000	-1	2*
14	0	0	0	0.0000000	-1	1*
15	0	0	0	0.0000000	-1	2*
16	20	21	10	0.4015341	1	0
17	22	23	10	0.1524530	1	0
18	24	25	2	0.6465817	1	0
19	26	27	2	0.6180417	1	0
20	28	29	3	0.9030539	1	0
21	0	0	0	0.0000000	-1	2*
22	0	0	0	0.0000000	-1	2*
23	0	0	0	0.0000000	-1	1*
24	0	0	0	0.0000000	-1	2*
25	0	0	0	0.0000000	-1	1*
26	0	0	0	0.0000000	-1	2*
27	0	0	0	0.0000000	-1	1*
28	0	0	0	0.0000000	-1	1*
29	0	0	0	0.0000000	-1	2*

#accès à un arbre individuel
print(getTree(model.rf,k=4))

- L'arbre a 15 feuilles
- La variable de segmentation à la racine est x1
- Une variable peut apparaître plusieurs fois (par ex. x2, x6, etc.)
- Des variables non pertinentes peuvent être intégrées (par ex. x7, x8, x3, x5, x6, etc.)

Random forest

Bilan

Avantages

- Bonnes performances en prédiction
- Paramétrage simple (**B** et **m**)
- Pas de problème d'overfitting (on peut augmenter B)
- Mesure de l'importance des variables
- Evaluation de l'erreur intégrée (OOB)
- Possibilité de programmation parallèle

Inconvénients

- Problème si le nombre de variables pertinentes très faible
- Déploiement d'un tel modèle peut être compliqué

PLAN

1. Arbres de décision
2. Bagging
3. Random Forest
4. **Boosting**
5. Bilan

Boosting

→ Toujours apprendre sur différentes versions des données.
Mais mieux diriger l'apprentissage en se focalisant sur les individus mal classés à l'étape précédente.

Boosting. Combiner des weaklearner permet de produire une modèle performant, nettement meilleur que chaque modèle pris individuellement.

Pondération des individus. A l'étape (n+1), donner une pondération plus élevée aux individus mal classés. La construction des modèles est séquentielle.

Pondération des modèles. Système de vote pondéré (selon la performance) en classement.

Biais et variance. Orienter l'apprentissage à chaque étape réduit le biais ; en les combinant, il réduit la variance.

Overfitting. Augmenter B n'aboutit pas au sur-apprentissage.

Arbres de décision. Peut s'appliquer à tout type de modèle.

Boosting

Adaptative boosting (AdaBoost)

Un algorithme avant tout pour les problèmes binaires Y = {+, -}

Apprentissage

Entrée : B nombre de modèles, ALGO algorithme d'apprentissage, Ω un ensemble de données de taille n avec y cible à K modalités, X avec p prédicteurs.

MODELES = {}

Les individus sont uniformément pondérés $\omega^1_i = 1/n$

Pour b = 1 à B Faire

- Construire un modèle M_b sur $\Omega(\omega^b)$ avec ALGO (ω^b pondération à l'étape b)
- Ajouter M_b dans MODELES
- Calculer le taux d'erreur pondéré pour M_b : $\varepsilon_b = \sum_{i=1}^{n} \omega_i^b \times I(y_i \neq \hat{y}_i)$
- Si $\varepsilon_b > 0.5$ ou $\varepsilon_b = 0$, arrêt de l'algorithme
- Sinon
  - Calculer $\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b}$
  - Les poids sont remis à jour $\omega_i^{b+1} = \omega_i^b \times \exp\left[\alpha_b . I(y_i \neq \hat{y}_i)\right]$
  - Et normalisés pour que la somme fasse 1

Fin Pour

Remarque : Si ($\varepsilon_b > 0.5$), on passe à des poids négatifs, d'où l'arrêt. Cette condition, naturelle dans un cadre binaire (K = 2), devient très restrictive en multi-classes (K > 2). L'apprenant doit faire largement mieux que « weak ».

Boosting

Adaptative boosting (AdaBoost)

→ Utilisation d'un vote pondéré pour le classement

Classement

Pour un individu i* à classer,

Appliquer chaque modèle M_b de MODELES → $\hat{y}_b(i^*)$

Prédiction boosting → $\hat{y}_{M1}(i^*) = \arg\max_k \left[\sum_{b=1}^{B} \alpha_b . I(\hat{y}_b(i^*) = y_k) \right]$

→ Ce qui correspond à un vote pondéré

Remarque : Plus le modèle est performant (ε_b petit), plus son poids sera élevé (α_b grand), plus il pèsera dans le processus de décision.

$$\alpha_b = \ln \frac{1-\varepsilon_b}{\varepsilon_b}$$

Boosting

Et au-delà du cadre binaire ?

→ La condition ($\varepsilon_b < 0.5$) pour pouvoir poursuivre est très (trop) restrictive quand ($K > 2$).

Solution 1 ➢ Décomposer la modélisation en un ensemble de problèmes binaires

A. Stratégie 1 contre les autres: il y a **K** apprentissages à effectuer
→ en prédiction, on choisit la conclusion qui a le score le plus élevé

B. Stratégie 1 contre 1: il y a **K(K-1)/2** apprentissages à effectuer
→ en prédiction, on choisit la classe qui présente le plus de victoires

Solution 2 ➢ Modifier le calcul de α $\alpha_b = \ln\frac{1-\varepsilon_b}{\varepsilon_b} + \ln(K-1)$

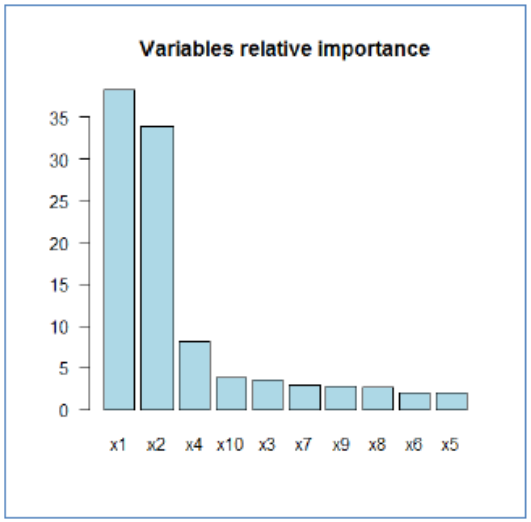
La contrainte devient : $\varepsilon_b < 1 - \frac{1}{K}$

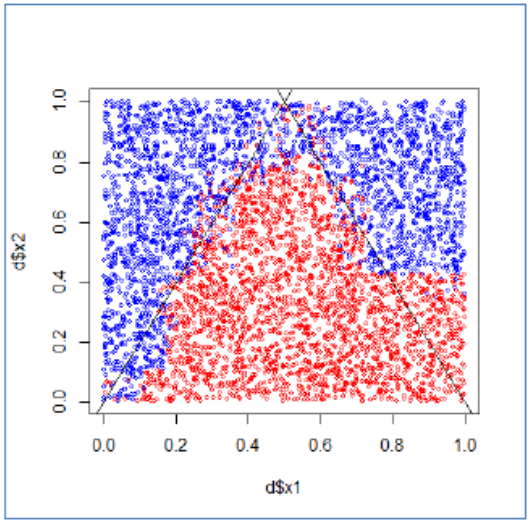
→ D'autres approches existent, mais elles sont peu diffusées, peu présentes dans les logiciels.

Boosting

Et sur nos données ?

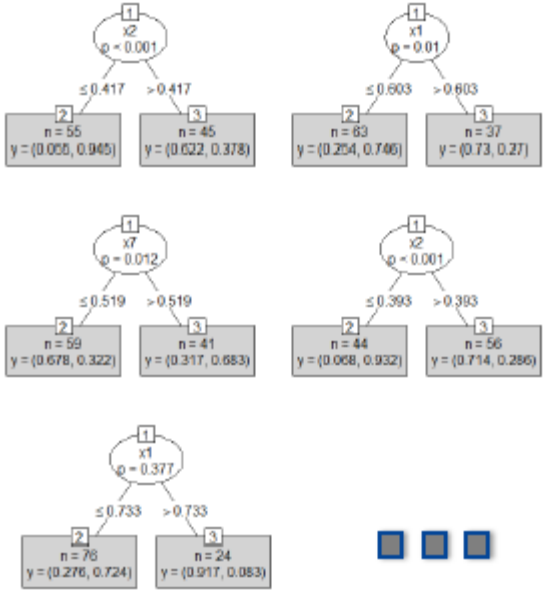
```
library(adabag)

#boosting avec 100 arbres
model.boosting <- boosting(y ~ ., data = d.train, mfinal=100)

#importance des variables :
#somme des contributions des variables dans chaque arbre
#pondérée par l'importance de l'arbre αb
importanceplot(model.boosting)

#prédiction sur échatillon test
p.boosting <- predict(model.boosting,newdata=d.test)

#taux d'erreur
print(erreur(d.test$y,p.boosting$class)) # 0.1242

#frontières dans le plan
nuage(d.test,factor(p.boosting$class))
```





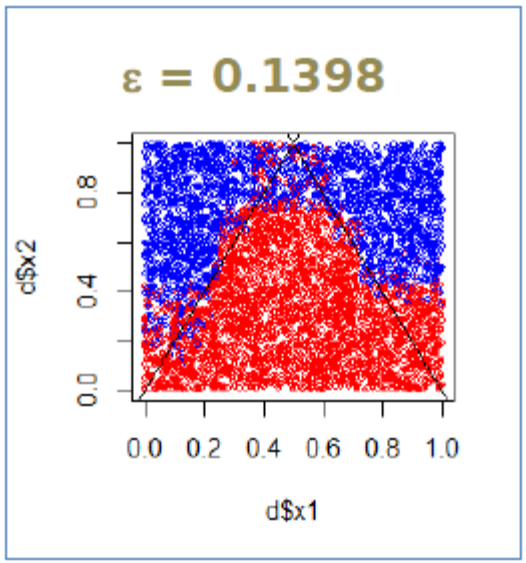
Boosting

Decision stump



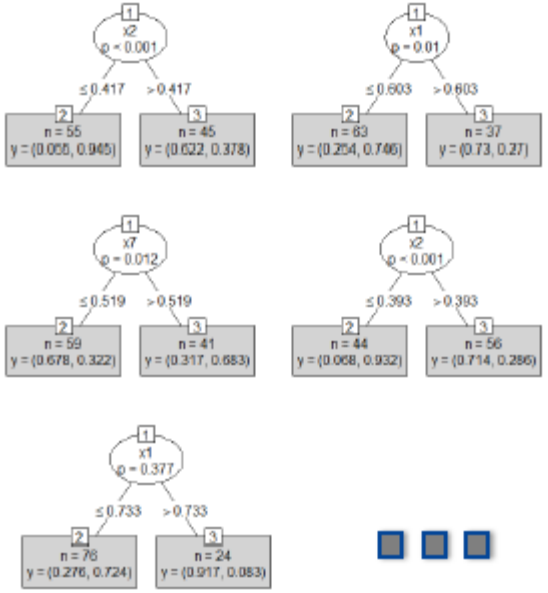
→ Le boosting agit sur le biais. On peut se limiter à des modèles très simples comme les «decisionstump» (arbre à une seule segmentation) qui ont un fort biais mais une très faible variance.

```
library(adabag)

#paramètres de construction de l'arbre
parametres = list(cp=0,maxdepth=1,minbucket=1)

#boosting avec 100 decision stump
stump.boosting <- boosting(y ~ ., data = d.train,
          mfinal=100, control=parametres)
```



Boosting

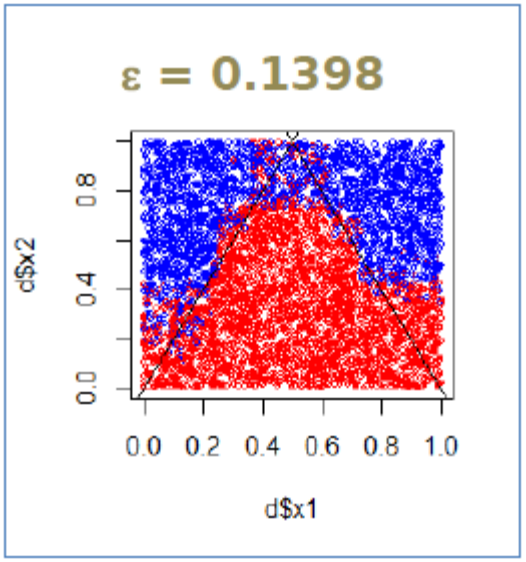
Decision stump



- On ne tient pas compte des interactions dans le modèle de base, pourtant le boosting améliore les performances
  → les variables apparaissent tour à tour ce qui induit une sorte de frontière «floue».

- Avec un arbre à 2 niveaux, on tiendrait compte des interactions d'ordre 2 entre les variables. Mais ce moindre biais peut faire perdre en variance.

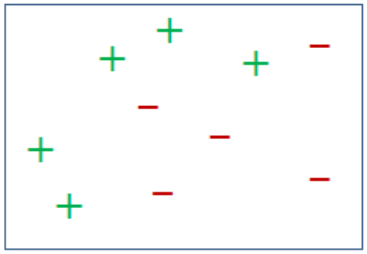
- Si tous les descripteurs sont binaires, le boosting avec des decision stump correspond à une combinaison linéaire des indicatrices



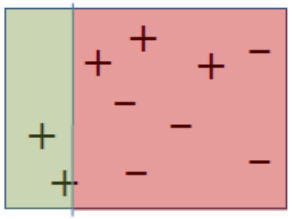
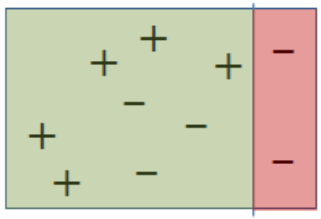
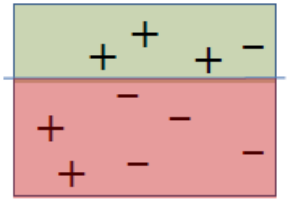
Boosting

Exemple de decision stump

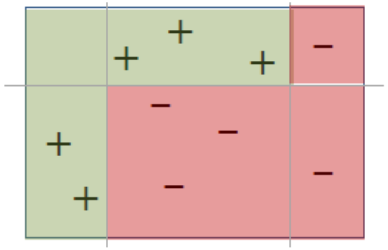
Un arbre à un niveau ne saura pas discerner parfaitement les ‘’+’’ des ‘’–’’



Mais une combinaison de 3 arbres à un niveau saura le faire

H = signe (0.42 x  + 0.65 x  + 0.92 x )

La combinaison produit un modèle qui, lui, sait discerner parfaitement les classes



Boosting

Bilan

Avantages

- Bonnes performances en prédiction
- Paramétrage simple (**B**)
- Importance des variables
- Joue sur le biais et la variance
- Ne nécessite pas des grands arbres
- On peut jouer sur la profondeur pour tenir compte des interactions

Inconvénients

- Pas de parallélisation possible
- Problème si M_b trop simple : underfitting
- Problème si M_b trop complexe : overfitting
- Problème si points aberrants ou bruités, poids exagérés
- Déploiement d'un tel modèle reste compliqué

PLAN

1. Arbres de décision
2. Bagging
3. Random Forest
4. Boosting
5. **Bilan**

Bagging, Random Forest, Boosting

Bilan

- Les méthodes ensemblistes qui appliquent répétitivement une méthode d'apprentissage sur des versions différentes des données (ré-échantillonnage, repondération) sont maintenant bien connues, les gains en performances sont reconnues.

- Le seul véritable frein est le manque de lisibilité du méta-modèle, malgré l'indicateur «importance des variables», qui empêche une interprétation sophistiquée des relations de causes à effet, nécessaire dans certains domaines.