# Séance 10: Arbres de classifications et Forêts aléatoires

Sébastien Gadat

Laboratoire de Statistique et Probabilités
UMR 5583 CNRS-UPS

www.lsp.ups-tlse.fr/gadat

Dixième partie X

# Agrégation de modèles

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Introduction

- Algorithmes très récents (années 2000) pour la classification et la régression
- Utilise des stratégies adaptatives (*boosting*) ou aléatoires (*bagging*)
- Idée : Utiliser une combinaison ou une agrégation d'un grand nombre de modèles tout en évitant l'*overfitting*
- Bagging ou Random Forests : utiliser le hasard pour améliorer les performances d'algorithmes de « faibles »performances. C'est applicable à différents algorithmes et RF est un aménagement spécifique à CART.
- Boosting : utiliser une stratégie adaptative pour *booster* des performances, applicable là aussi à tout type d'algorithme (Réseau de neurones, cart, etc.)
- Ces stratégies améliorent principalement les performances des algorithmes non linéaires plus instables. Ils sont en général moins efficace dans le cas des méthodes linéaires stables.

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Bagging

- $Y$ variable à expliquer, $X^1, \dots X^p$ variables explicatives
- $\phi$ modèle appris sur un échantillon $z = \{(x_1, y_1) \dots (x_n, y_n)\}$
- On considère $B$ échantillons bootstrap $z_1, \dots z_B$ d'individus étiquetés, ces échantillons sont issus de $z$ par tirage aléatoire avec remise.
- Sur chacun des échantillons, on apprend un modèle $\phi_{z_i}$
- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \frac{1}{B} \sum_{i=1}^{B} \phi_{z_i}(x)$$

pour une variable quantitative

- On *prédit* $Y$ en agrégeant les différentes décisions sur chacun des $z_i$ par

$$\hat{\phi}(x) = \text{Décision majoritaire parmi les } \phi_{z_i}(x)$$

pour une variable qualitative

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Utilisation du Bagging

- On estime assez naturellement l'erreur de prédiction par *bootstrap out of bag*, ce qui prévient le sur-ajustement
- Pour CART, on peut choisir parmi différentes stratégies :
  - Construire des arbres complets sans élagage
  - Construire un arbre d'au plus $q$ feuilles
  - Construire des arbres complets et élaguer par validation croisée
- En pratique, on garde souvent la première stratégie, chacun des arbres a un faible biais mais une grande variance
- La moyenne réduit avantageusement cette variance
- Avantage : très simple à mettre en oeuvre
- Inconvénients :
  - Temps de calcul un petit peu important
  - Interprétabilité peu évidente

# Random Forests

- Amélioration du Bagging dans le cas spécifique de l'algorithme CART
- L'objectif est de rendre les modèles (arbres) construits plus indépendants entre eux
- Cette indépendance va permettre de rendre le vote des experts (différents CART) plus efficace
- Approche très fructueuse en grande dimension, par exemple dans le cadre des bio-puces, signaux, images ou courbes
- Avantages : encore une fois très simple à mettre en oeuvre et coût numérique peu important en regard des performances obtenues

# Random Forests

- Amélioration du Bagging dans le cas spécifique de l'algorithme CART
- L'objectif est de rendre les modèles (arbres) construits plus indépendants entre eux
- Cette indépendance va permettre de rendre le vote des experts (différents CART) plus efficace
- Approche très fructueuse en grande dimension, par exemple dans le cadre des bio-puces, signaux, images ou courbes
- Avantages : encore une fois très simple à mettre en oeuvre et coût numérique peu important en regard des performances obtenues

# Random Forests

- Amélioration du Bagging dans le cas spécifique de l'algorithme CART
- L'objectif est de rendre les modèles (arbres) construits plus indépendants entre eux
- Cette indépendance va permettre de rendre le vote des experts (différents CART) plus efficace
- Approche très fructueuse en grande dimension, par exemple dans le cadre des bio-puces, signaux, images ou courbes
- Avantages : encore une fois très simple à mettre en oeuvre et coût numérique peu important en regard des performances obtenues

# Random Forests

- Amélioration du Bagging dans le cas spécifique de l'algorithme CART
- L'objectif est de rendre les modèles (arbres) construits plus indépendants entre eux
- Cette indépendance va permettre de rendre le vote des experts (différents CART) plus efficace
- Approche très fructueuse en grande dimension, par exemple dans le cadre des bio-puces, signaux, images ou courbes
- Avantages : encore une fois très simple à mettre en oeuvre et coût numérique peu important en regard des performances obtenues

# Random Forests

- Amélioration du Bagging dans le cas spécifique de l'algorithme CART
- L'objectif est de rendre les modèles (arbres) construits plus indépendants entre eux
- Cette indépendance va permettre de rendre le vote des experts (différents CART) plus efficace
- Approche très fructueuse en grande dimension, par exemple dans le cadre des bio-puces, signaux, images ou courbes
- Avantages : encore une fois très simple à mettre en oeuvre et coût numérique peu important en regard des performances obtenues

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$, avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Algorithme de Random Forests

- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage, $x_i$ décrit par $p$ variables explicatives
- Pour $b = 1 \dots B$ (B représente le nombre d'arbres formés dans la forêt)
  - Tirer un échantillon aléatoire $z_b$ avec remise parmi $z$
  - Estimer un arbre sur $z_b$ avec *randomisation* des variables :
  - Pour la construction de chaque noeud de chaque arbre, on tire uniformément $q$ variables parmi $p$ pour former la décision associée au noeud
- En fin d'algorithme, on possède $B$ arbres que l'on moyenne ou qu'on fait voter pour la régression ou la classification
- En général, un choix optimal pour $q$ est à peu près $q = \sqrt{p}$

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Stratégie d'élagage de Random Forests

- Stratégie plus élémentaire que pour CART ou le Bagging
- On se limite en général à des arbres de très faibles profondeur (voire $q = 2$)
- Dans le cas du Bagging, il faut un arbre profond pour qu'il ne soit pas trop corrélé aux arbres formés par Bagging puisque seul les échantillons d'apprentissage changent
- Dans le cas de Random Forests, le tirage aléatoire des variables explicatives à chaque noeud aboutit à des arbres non corrélés
- Chacun des petits arbres est moins performant mais l'union fait la force : la stratégie d'agrégation est très performantes puisque ces arbres sont non corrélés.
- On estime l'erreur *via* la méthode out-of-bag pour prévenir le sur-ajustement
- On peut interpréter les résultats de RF en représentant le nombre de fois qu'une variable a été utilisée dans les $B$ arbres construits

# Évaluation de Random Forests

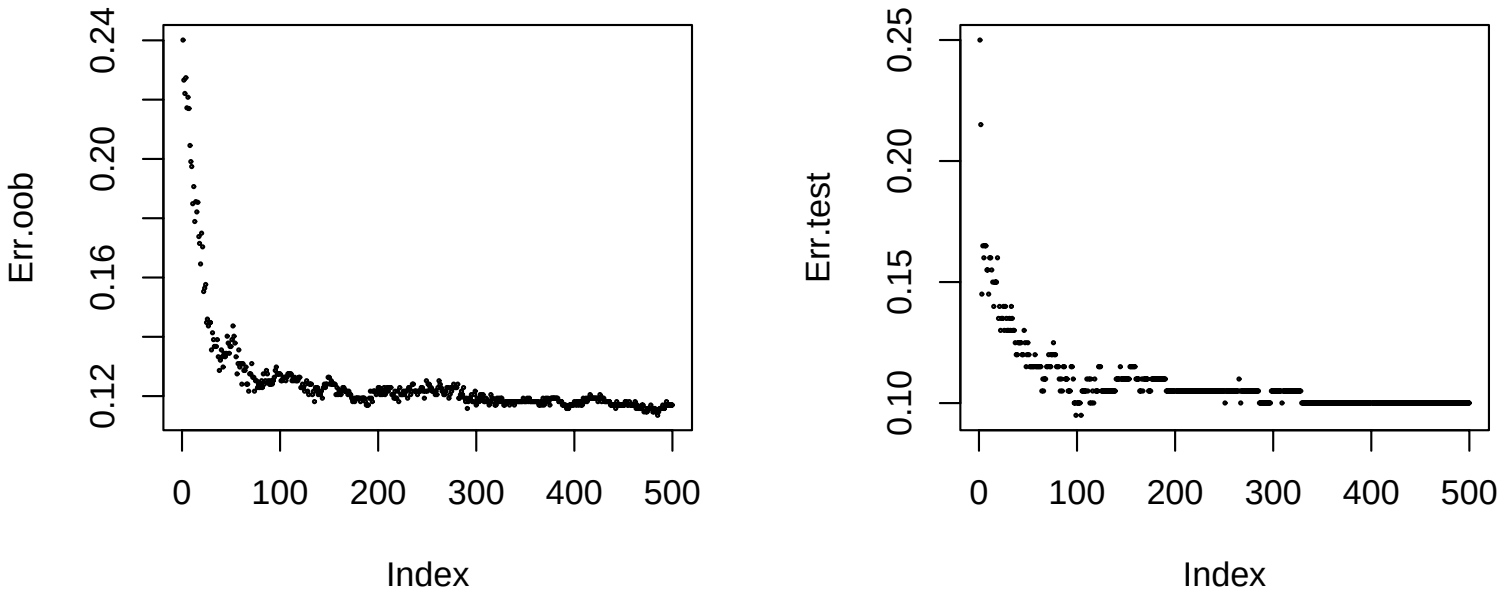


FIG.: Évolution du taux d'erreur oob et sur échantillon test en fonction du nombre d'arbres pour le jeu de données Visa Premier pour l'algorithme Random Forests

# Boosting

- Algorithme récent (1996 pour Adaboost)
- Idée : améliorer les performances d'un *classifieur* faible, c'est-à-dire un algorithme dont la probabilité de succès est légèrement faible supérieure à celle d'un choix aléatoire
- Algorithme itératif
- On pondère l'importance des observations mal-estimées lors de l'itération précédente
- Il existe de nombreuses versions de ces algorithmes, différant via les fonctions de coûts utilisées

# Boosting

- Algorithme récent (1996 pour Adaboost)
- Idée : améliorer les performances d'un *classifieur* faible, c'est-à-dire un algorithme dont la probabilité de succès est légèrement faible supérieure à celle d'un choix aléatoire
- Algorithme itératif
- On pondère l'importance des observations mal-estimées lors de l'itération précédente
- Il existe de nombreuses versions de ces algorithmes, différant via les fonctions de coûts utilisées

# Boosting

- Algorithme récent (1996 pour Adaboost)
- Idée : améliorer les performances d'un *classifieur* faible, c'est-à-dire un algorithme dont la probabilité de succès est légèrement faible supérieure à celle d'un choix aléatoire
- Algorithme itératif
- On pondère l'importance des observations mal-estimées lors de l'itération précédente
- Il existe de nombreuses versions de ces algorithmes, différant via les fonctions de coûts utilisées

# Boosting

- Algorithme récent (1996 pour Adaboost)
- Idée : améliorer les performances d'un *classifieur* faible, c'est-à-dire un algorithme dont la probabilité de succès est légèrement faible supérieure à celle d'un choix aléatoire
- Algorithme itératif
- On pondère l'importance des observations mal-estimées lors de l'itération précédente
- Il existe de nombreuses versions de ces algorithmes, différant via les fonctions de coûts utilisées

# Boosting

- Algorithme récent (1996 pour Adaboost)
- Idée : améliorer les performances d'un *classifieur* faible, c'est-à-dire un algorithme dont la probabilité de succès est légèrement faible supérieure à celle d'un choix aléatoire
- Algorithme itératif
- On pondère l'importance des observations mal-estimées lors de l'itération précédente
- Il existe de nombreuses versions de ces algorithmes, différant via les fonctions de coûts utilisées

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit » $c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit » $c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit » $c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(A(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit »$c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit » $c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit » $c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit »$c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- Le cadre est un algorithme de classification à 2 classes $\{-1; +1\}$
- $z = \{(x_1, y_1) \dots (x_n, y_n)\}$ échantillon d'apprentissage
- Initialiser des poids sur les variables

$$\omega = \{\omega_i = 1/n \quad i \in \{1 \dots n\}\}$$

- Tant que les poids $\omega$ ne convergent pas, procéder à l'itération $m$ :
  - Estimer $\mathbb{A}$ sur l'échantillon pondéré par $\omega$
  - Calculer le taux d'erreur apparent :

$$\hat{\mathcal{E}} = \frac{\sum_{i=1}^{n} \omega_i \delta(\mathbb{A}(x_i), y_i)}{\sum_{i=1}^{n} \omega_i}$$

  - Calculer la quantité « logit »$c_m = \log((1 - \hat{\mathcal{E}})/\hat{\mathcal{E}})$
  - Calculer les nouveaux poids $\omega_i = \omega_i e^{-c_m \delta(\mathbb{A}(x_i), y_i)}$

# Adaptative Boosting AdaBoost

- La règle de décision pour le boosting est alors l'agrégation :

$$\hat{\phi}(x) = \sum_{i=1}^{M} c_m \mathbb{A}_m(x)$$

- A noter que le poids d'un échantillon est inchangé si cet échantillon est bien classé à une itération
- On peut utiliser AdaBoost avec comme algorithme de base CART. En général on obtient un algorithme beaucoup plus performant et pour des arbres de profondeur $2$, on obtient mieux en effectuant un boosting sur CART qu'en construisant un arbre complexe beaucoup plus profond.
- Il existe également une version de l'algorithme pour la régression en utilisant comme fonction de coût un terme quadratique (au lieu de $\delta$).

# Adaptative Boosting AdaBoost

- La règle de décision pour le boosting est alors l'agrégation :

$$\hat{\phi}(x) = \sum_{i=1}^{M} c_m \mathbb{A}_m(x)$$

- A noter que le poids d'un échantillon est inchangé si cet échantillon est bien classé à une itération
- On peut utiliser AdaBoost avec comme algorithme de base CART. En général on obtient un algorithme beaucoup plus performant et pour des arbres de profondeur $2$, on obtient mieux en effectuant un boosting sur CART qu'en construisant un arbre complexe beaucoup plus profond.
- Il existe également une version de l'algorithme pour la régression en utilisant comme fonction de coût un terme quadratique (au lieu de $\delta$).

# Adaptative Boosting AdaBoost

- La règle de décision pour le boosting est alors l'agrégation :

$$\hat{\phi}(x) = \sum_{i=1}^{M} c_m \mathbb{A}_m(x)$$

- A noter que le poids d'un échantillon est inchangé si cet échantillon est bien classé à une itération
- On peut utiliser AdaBoost avec comme algorithme de base CART. En général on obtient un algorithme beaucoup plus performant et pour des arbres de profondeur $2$, on obtient mieux en effectuant un boosting sur CART qu'en construisant un arbre complexe beaucoup plus profond.
- Il existe également une version de l'algorithme pour la régression en utilisant comme fonction de coût un terme quadratique (au lieu de $\delta$).

# Adaptative Boosting AdaBoost

- La règle de décision pour le boosting est alors l'agrégation :

$$\hat{\phi}(x) = \sum_{i=1}^{M} c_m \mathbb{A}_m(x)$$

- A noter que le poids d'un échantillon est inchangé si cet échantillon est bien classé à une itération
- On peut utiliser AdaBoost avec comme algorithme de base CART. En général on obtient un algorithme beaucoup plus performant et pour des arbres de profondeur $2$, on obtient mieux en effectuant un boosting sur CART qu'en construisant un arbre complexe beaucoup plus profond.
- Il existe également une version de l'algorithme pour la régression en utilisant comme fonction de coût un terme quadratique (au lieu de $\delta$).

# Comparaison Random Forests/Boosting

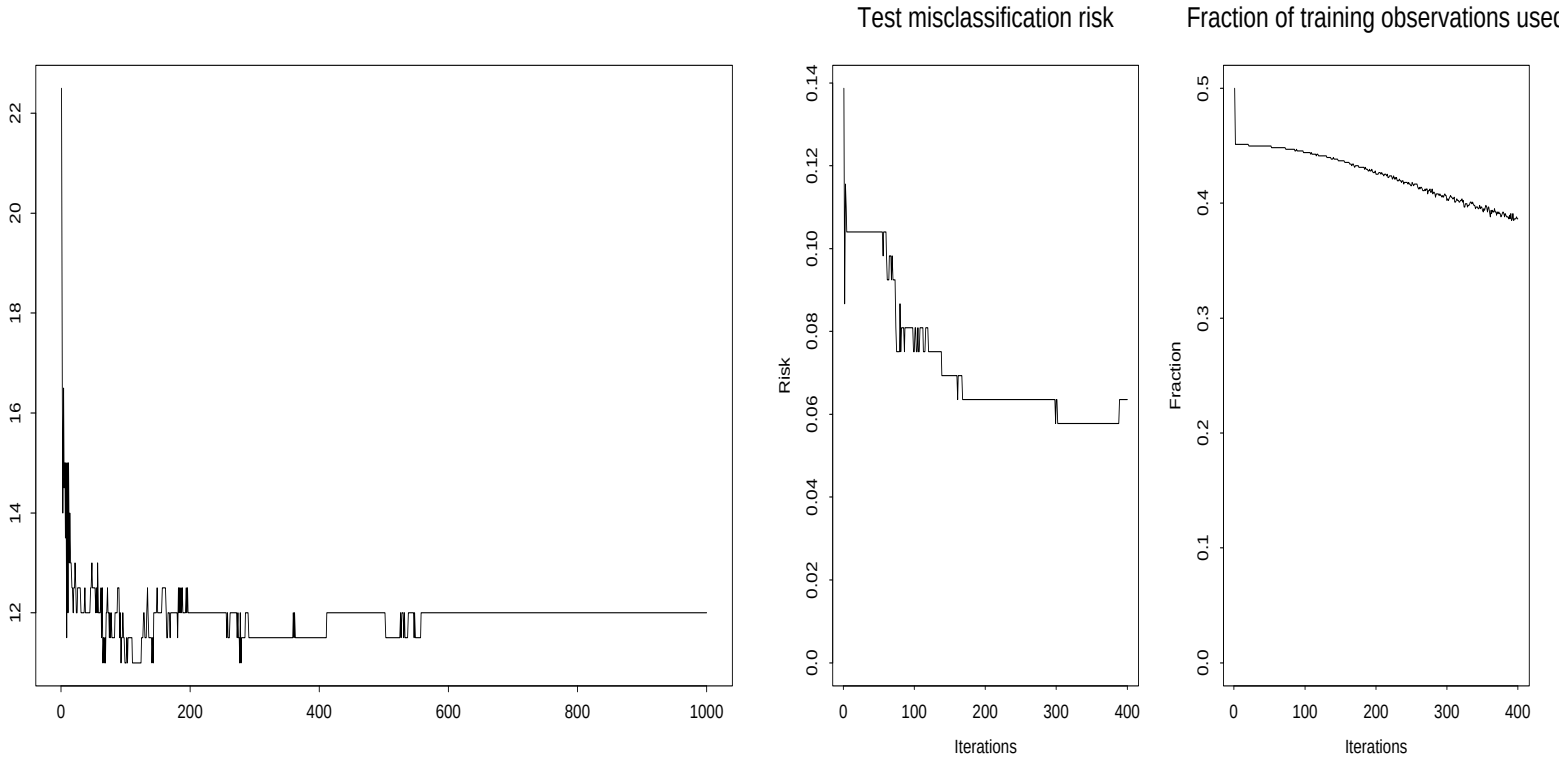


FIG.: Évolution de l'erreur de RF en fonction du nombre d'arbres, ainsi que l'erreur d'AdaBoost (*via* CART) en fonction du nombre d'itérations