Arbres de décision

masedki.github.io

ven. 13 mars 2020

Outline

Méthodes basées sur des arbres

- Nous décrivons ici des méthodes *basées sur des arbres* pour la classification et la régression.
- Cela implique de *stratifier* ou *segmenter* l'espace des prédicteurs en un certain nombre de régions simples.
- Comme les règles des partitionnement peuvent être résumées par un arbre, ce type d'approches sont connues comme des méthodes à *arbres de décision*.

Pours et contres

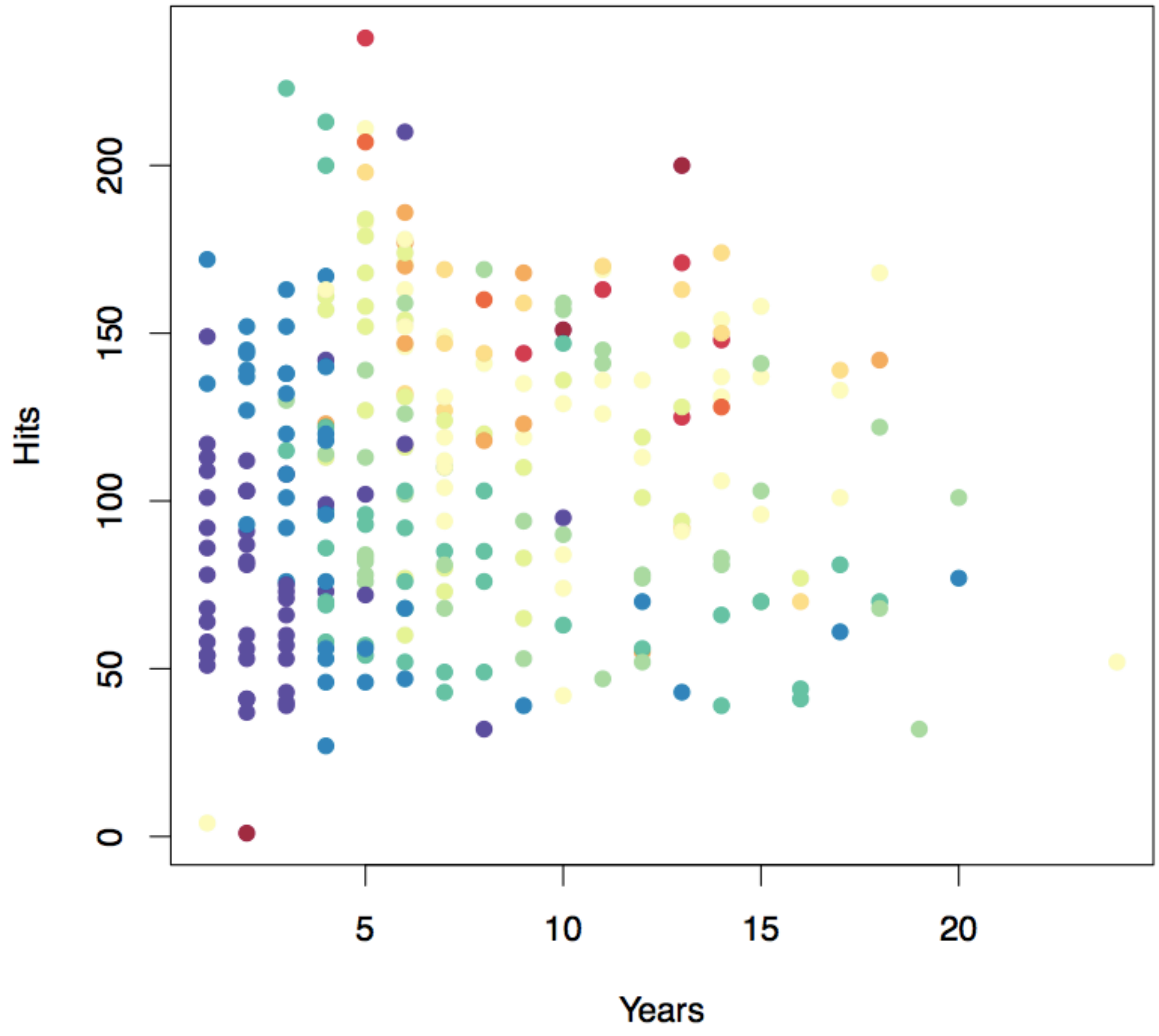
- Les méthodes basées sur des arbres sont simples et utiles pour l'interprétation.
- Cependant, elles ne sont pas capables de rivaliser avec les meilleures approches d'apprentissage supervisé en terme de qualité de prédiction.
- Nous discuterons aussi (dans l'avenir en M2) de *bagging*, *forêts aléatoires (random forests)*, et *boosting*. Ces méthodes développent de nombreux arbres de décision qui sont ensuite *combinés* pour produire une réponse consensus.

Les bases des arbres de décision

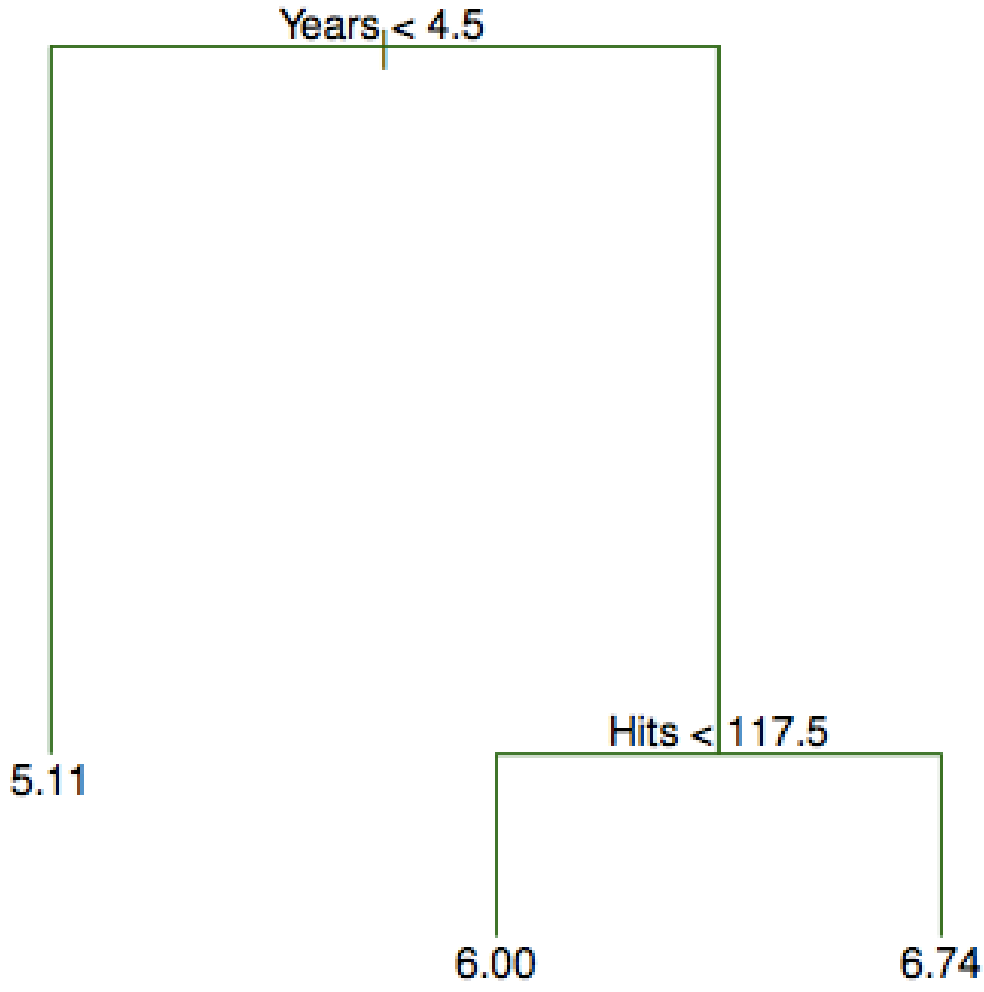
- Les arbres de décision sont utiles aussi bien pour des problèmes de régression que de classification.
- Nous commençons par présenter des problèmes de régression et nous viendrons ensuite à la classification.

Données de salaire au baseball: comment les stratifier ?

Le salaire est codé par des couleurs : les faibles valeurs sont en bleu, puis vert, les plus fortes valeurs en orange puis rouge.



L’arbre de décision sur ces données

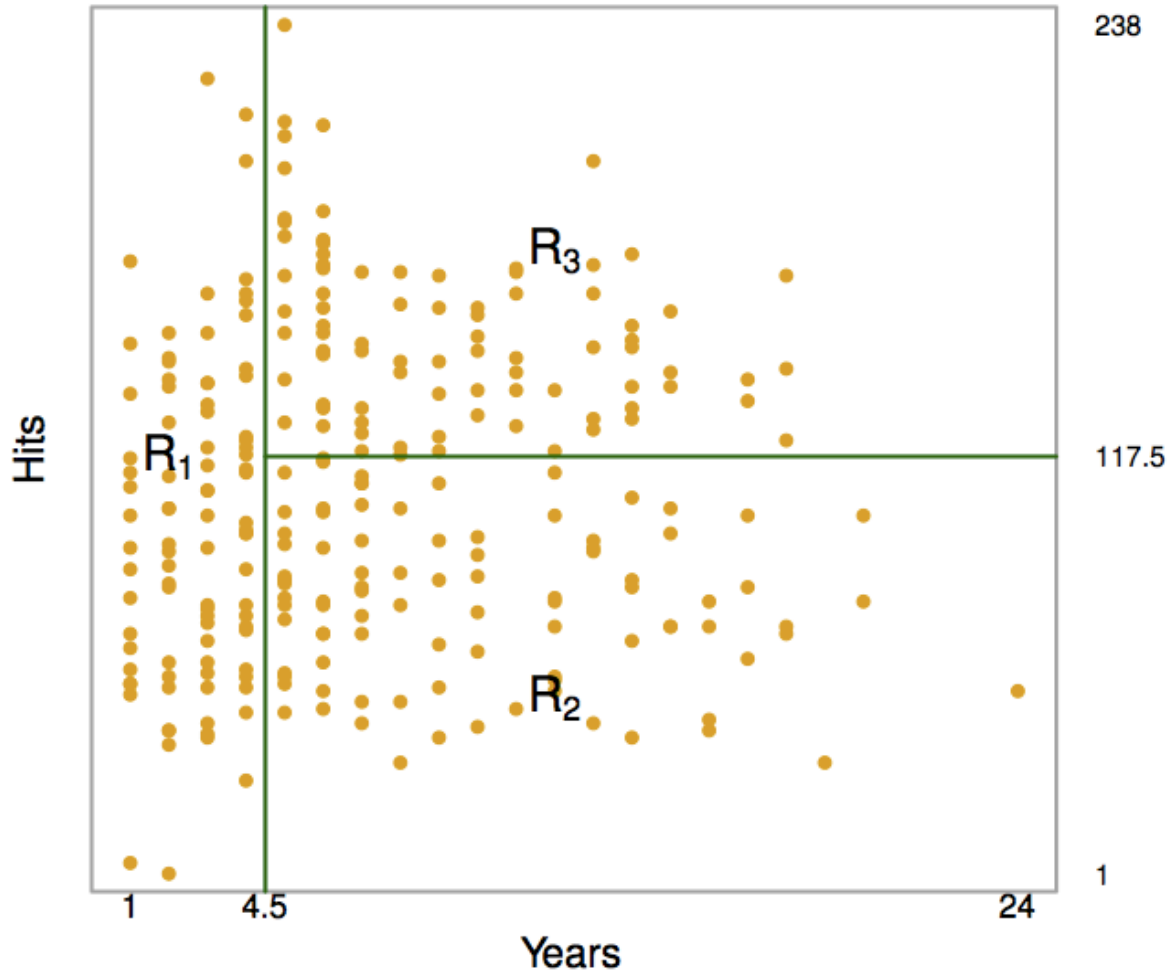


Détails de la précédente figure

- C'est un arbre de régression pour prédire le **log** des salaires des joueurs, basé sur
  - l'expérience (`Years`)
  - le nombre de succès (`Hits`)
- Pour chaque nœud interne, l'étiquette (de la forme $X_j < t_k$) indique la branche de gauche émanant du nœud et la branche droite correspond à $X_j \geq t_k$.
- Cet arbre a deux nœuds internes et trois nœuds terminaux ou feuilles. Le nœud le plus haut dans la hiérarchie est la racine.
- L'étiquette des feuilles est la réponse moyenne des observations qui satisfont aux critères pour la rejoindre.

Résultats

- En tout, l'arbre distingue trois classes de joueurs en partitionnant l'espace des variables explicatives en trois régions : $R_1 = \{X : \texttt{Years} < 4.5\}$, $R_2 = \{X : \texttt{Years} \geq 4.5, \texttt{Hits} < 117.5\}$ et $R_3 = \{X : \texttt{Years} \geq 4.5, \texttt{Hits} \geq 117.5\}$.



Interprétation des résultats

- `Years` est le facteur le plus important pour expliquer `Salary` : les joueurs de moindre expérience gagnent moins que les joueurs expérimentés
- Sachant qu'un joueur a peu d'expérience, le nombre de `Hits` l'année passée n'influence pas son salaire
- Mais, parmis les joueurs expérimentés, le nombre de `Hits` de l'année passée affecte son salaire (positivement)
- C'est sûrement une simplification de la réalité, mais comparé à un modèle de régression (linéaire par exemple), la fonction de régression est simple à décrire, interpréter et expliquer.

Détails sur la construction de l'arbre

Algorithme CART (Classification and Regression Trees)

1. Division de l'espace des prédicteurs en J régions distinctes, non recouvrantes: $R_1, R_2, \ldots R_J$.
2. Pour toute nouvelle observation des prédicteurs $X = x_0$, on regarde dans quelle région on tombe, disons R_ℓ. La prédiction est la moyenne des valeurs observées dans la partie de l'ensemble d'entrainement qui tombent dans R_ℓ.

Détails sur la construction de l'arbre (suite)

- Pour limiter l'espace des partitions possibles, les arbres de décision divisent l'espace en rectangles ou boîtes parallèles aux axes.
- Le but est de trouver les boîtes $R_1, \ldots, R_J$ qui minimisent un critère des moindres carrés, ici

$$SSE = \sum_{j=1}^{J} \sum_{i:x_i \in R_j} \left(y_i - \hat{y}_{R_j} \right)^2,$$

où $\hat{y}_{R_j}$ est la réponse moyenne sur les observations d'entrainement qui tombent dans R_j.

Détails sur la construction de l'arbre (suite)

- Malheureusement, il est impossible de traverser l'ensemble des partitionnements de l'espace des prédicteurs en *J* boîtes.
- Pour cette raison, on met en place un algorithme *glouton*, *top-down* qui construit l'arbre binaire de façon récursive.
- L'algorithme démarre à la racine de l'arbre et sépare ensuite l'espace des prédicteurs en ajoutant progressivement des nœuds.
- On parle d'algorithme *glouton* car à chaque étape de la construction de l'arbre, on construit la meilleur division possible du nœud en deux sous-nœuds.

L'algorithme de construction de l'arbre T_0 (phase 1)

Initialisation

Nœud racine : on place l'ensemble de l'échantillon d'estimation à la racine de l'arbre

Récurrence sur chaque nœud

On partionne chaque nœud en deux classes:

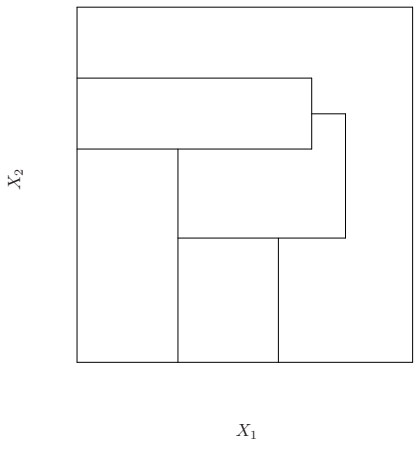
$$\mathcal{R}_1(j,s) = \{X : X_j \leq s\}, \quad \mathcal{R}_2(j,s) = \{X : X_j > s\}$$

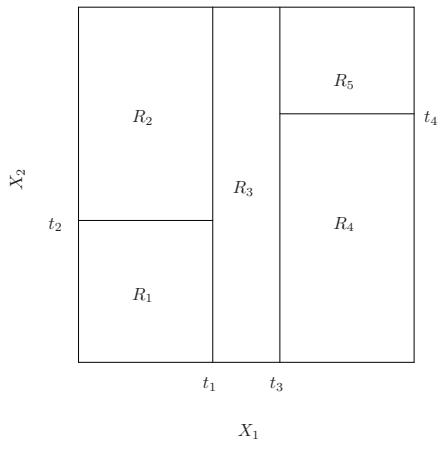
en cherchant j et s qui minimisent

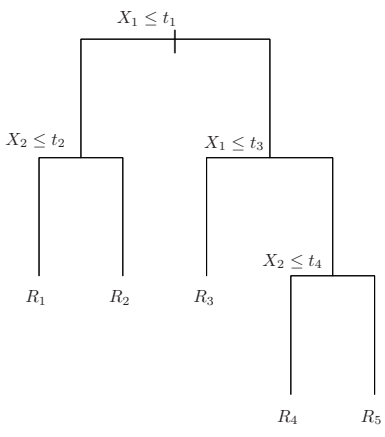
$$\text{RSS}_{\text{new}} = \sum_{i:x_i \in \mathcal{R}_1(j,s)} \left(y_i - \hat{y}_1\right)^2 + \sum_{i:x_i \in \mathcal{R}_2(j,s)} \left(y_i - \hat{y}_2\right)^2 \tag{1}$$

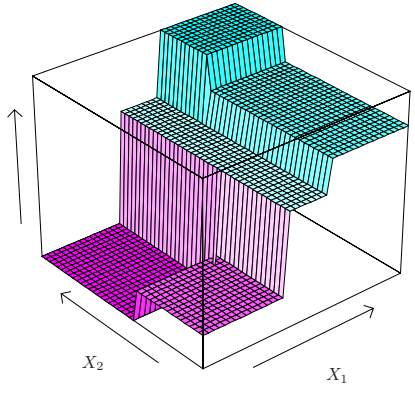
où $\hat{y}_m = \text{ave}\left(y_i \middle| x_i \in \mathcal{R}_m(j,s)\right)$ est la réponse moyenne des données d'apprentissage qui tombent dans la région $R_m(j,s)$ pour $m = 1$ ou $m = 2$.
Trouver le couple (j,s) optimal est un problème relativement facile lorsque le nombre de variables p n'est pas trop grand.

Exemples de récurrence binaire









Exemples de récurrence binaire

- En haut à gauche : exemple de partition qui ne peut être le résultat d'une partition binaire
- En haut à droite : résultat d'une partition binaire récursive
- En bas à gauche : l'arbre binaire correspondant à la partition en haut à droite
- En bas à droite : surface de prédiction associé à cet arbre

Algorithme (suite…)

Phase 1 : Construction de T_0

Initialisation

[…]

Récurrence sur chaque nœud

[…]

Terminaison

On arrête de diviser un nœud de T_0 lorsqu'il y a peu d'observations (disons 5).

Critère d'arrêt

- La récurrence jusqu'à 5 observations par noeud terminal est arbitraire
- Trop d'étapes de partitionnement : beaucoup de feuilles (noeuds terminaux), modèle trop complexe, petit biais mais grande variance, sur-apprentissage
- Peu d'étapes de partitionnement: peu de feuilles, modèle trop simple, grand biais mais petite variance, sous-apprentissage

Une première idée

- Division d'une région R en deux régions R_1 et R_2 on considère la somme des carrés des résidus avant la division

$$\text{RSS}_{\text{old}} = \sum_{i \in R} \left(y_i - \hat{y}\right)^2$$

où $\hat{y}$ est la moyenne de la variable réponse des données qui se situent dans la région R. Avec la division optimale, la réduction de RSS

$$\text{RSS}_{\text{old}} - \text{RSS}_{\text{new}}$$

- On peut choisir un seuil h et décider de la significativité d'une partition
- Si la réduction du RSS est supérieure à h on applique le *split* sinon on arrête

Une première idée (suite)

- L'idée est raisonnable mais trop *locale*
- Une partition peut être jugée non-significative et peut cacher d'autres partitions plus significatives

Sur-apprentissage

L'arbre T_0 obtenu est trop profond. Faire un compromis entre

- sur-apprentissage : trop profond
- arbre trop peu précis (grande erreur de prédiction): trop peu profond

Solution : élagage de T_0 appelé *Cost complexity pruning*

Élagage

Une stratégie consiste à construire un très grand arbre, puis à l'élaguer afin d'obtenir un sous-arbre.

- Comment détermine-t-on le meilleur moyen d'élaguer l'arbre ?
- Sélectionner un sous-arbre menant à l'erreur de test la plus faible.
- Nous pouvons estimer l'erreur de test en utilisant la validation croisée (chaque sous-arbre : explosion combinatoire !!).
- Sélectionner un petit ensemble de sous-arbres à prendre en compte.
- L'élagage du maillon le plus faible permet de considérer une séquence d'arbres indexés par un paramètre de réglage non négatif α.

Élagage : détails

Introduire un paramètre α qui règle le compromis, et minimiser le critère pénalisé *perte + pénalité* défini pour $T \subset T_0$ par

$$\mathcal{C}_\alpha(T) = \sum_{m=1}^{|T|} \sum_{x_i \in R_m(T)} \left(y_i - \hat{y}_m\right)^2 + \alpha|T|,$$

où

- $|T|$ est nombre de feuilles de T
- $\hat{y}_m = \text{ave}\left(y_i \middle| x_i \in \mathcal{R}_m(T)\right)$
- On notera T_α le sous-arbre qui minimise $C_\alpha(T)$ à α fixé
- Rôle de α ? Cas particuliers $\alpha = 0$ et $\alpha \to +\infty$!!

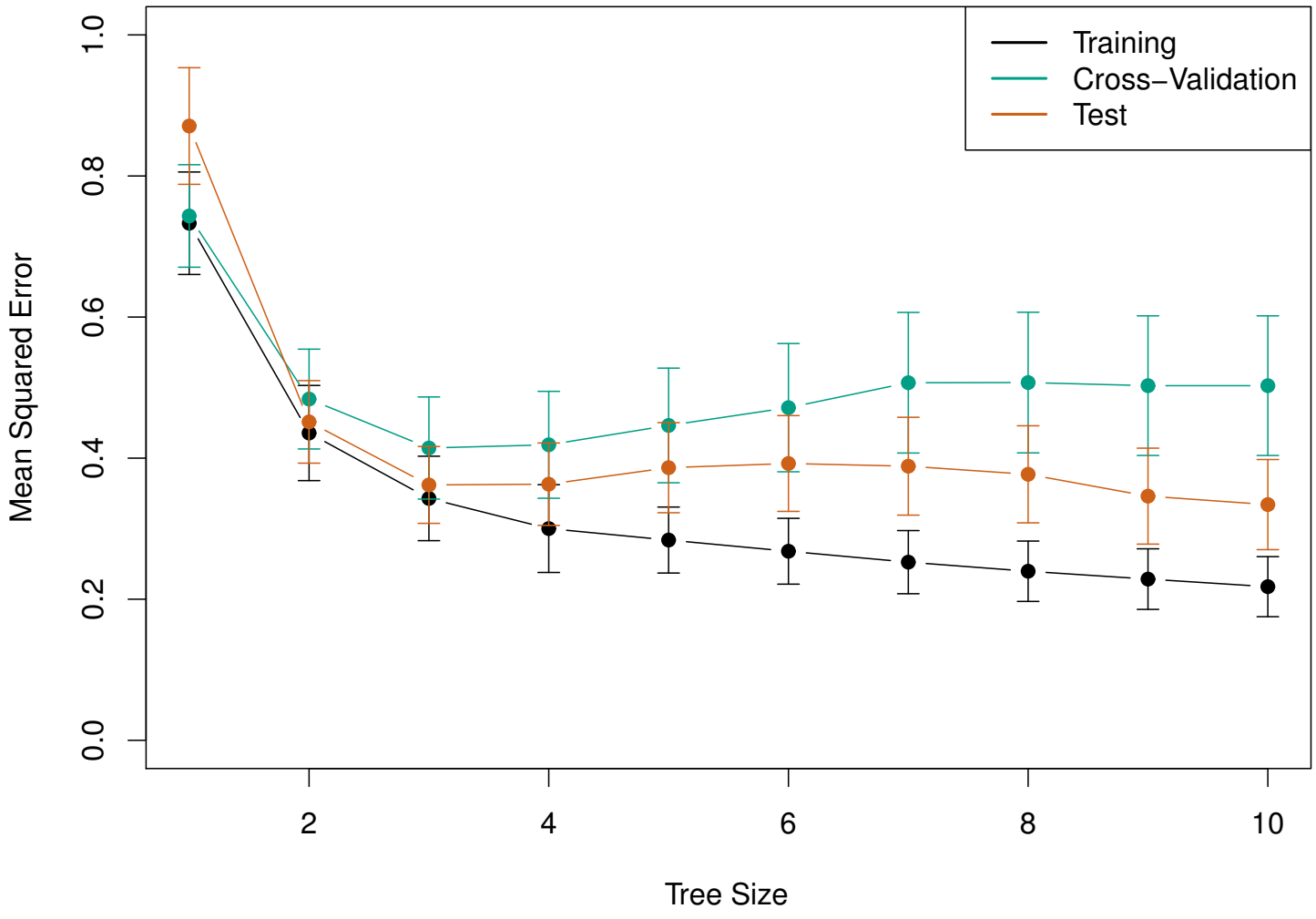
Élagage : Calcul des minima T_α du critère pénalisé

1. On construit une suite d'arbres itérativement
   - On part de T_0
   - À chaque étape, on supprime le nœud interne de tel sorte à produire la plus petite augmentation de
   $$\sum_{m=1}^{|T|} \sum_{x_i \in R_m(T)} \left(y_i - \hat{y}_m\right)^2$$
   - On s'arrête lorsque l'arbre est réduit à un seul nœud (racine)
2. Tous les minima $T = T_\alpha$ des fonctions $T \mapsto \mathcal{C}_\alpha(T)$ sont dans cette suite

Élagage : Choix de α par validation croisée K-folds

- Diviser le jeu de données d'apprentissage en K-folds
- Pour $k = 1, \ldots, K$:
  - Calculer les minima T_α du critère pénalisé sur l'ensemble du jeu de données privé du $k^{\text{ième}}$ fold
  - Pour chaque T_α, calculer l'erreur de prédiction moyenne des données du $k^{\text{ième}}$ fold comme une fonction $\text{err}_{-k}(\alpha)$ de α
- Choisir la valeur de α^* qui minimise la fonction moyenne $\frac{1}{K}\sum_{k=1}^{K} \text{err}_{-k}(\alpha)$
- Renvoyer T_{α^*} calculé par élagage sur l'ensemble du jeu de données d'apprentissage

Illustration : *Hitters* dataset



Exemple : coût de soins

- Compétition kaggle https://www.kaggle.com/mirichoi0218/insurance
- À l'aide de la fonction rpart, ajuster un arbre de décision sans élagage pour prédire la variable medv en fonction des autres variables présentes dans le jeu de données.
    - Utiliser la fonction rpart.control pour construire un arbre en continuant les découpages dans les feuilles qui contiennent au moins 5 observations (paramètre minsplit=5) et sans contrainte sur la qualité du découpage (paramètre cp=0)
    - Visualiser l'arbre obtenu à l'aide de la fonction rpart.plot
    - Évaluer l'erreur de prédiction du modèle sur le jeu de données test
- Découvrir l'élagage effectué automatiquement à l'aide de la fonction plotcp
- À l'aide de la fonction prune, extraire l'arbre obtenu par élagage correspondant à l'erreur minimale par validation croisée
- Tracer le nouvel arbre obtenu par élagage et évaluer son erreur de prédiction sur le jeu de données test

Arbres de classification

- Similaires aux arbres de régression, sauf qu'ils sont utilisés pour prédire une réponse catégorielle
- Pour un arbre de classification, on prédit à l'aide la classe la plus fréquente dans cette feuille parmi les données d'entrainement

Classification : différence avec la régression

- Rappelons qu'en régression, on vise à réduire les moindres carrés (ou somme des carrés des résidus) notés RSS qui sert à mesurer l'erreur du modèle
- En classification, on a besoin d'une d'une mesure d'erreur appropriée
- Réponse catégorielle $Y \in \{1, 2, \ldots, K\}$ donc la prédiction $\hat{f}(x) \in \{1, 2, \ldots, K\}$

Taux d'erreur pour la classification

- Si la feuille m représente la région $\mathcal{R}_m$ avec N_m observations, on définit

$$\hat{p}_{mk} = \frac{1}{N_m} \sum_{x_i \in \mathcal{R}_m} \mathbf{1}\{y_i = k\},$$

  la proportion d'observations du nœud m appartenant à la $k^{\text{ième}}$ classe.

- On assigne une nouvelle observation dans la région $\mathcal{R}_m$ à la classe $\hat{c}_m = \operatorname{argmax}_k \hat{p}_{mk}$ (vote à la majorité simple)

Mesures *d'impureté*

En classification, les différentes mesures d'impureté $Q_m(T)$ d'une feuille m sont

- **Taux de mauvais classement** :

$$\frac{1}{N_m} \sum_{x_i \in \mathcal{R}_m} \mathbf{1}\{y_i \neq \hat{c}_m\} = 1 - \hat{p}_{m\hat{c}_m}$$

- **Indice de Gini** :

$$\sum_{k \neq k'} \hat{p}_{mk} \hat{p}_{mk'} = \sum_k \hat{p}_{mk}(1 - \hat{p}_{mk})$$

- **Entropie** :

$$-\sum_k \hat{p}_{mk} \ln \hat{p}_{mk}$$

Mesures *d'impureté*

- Si $\mathcal{R}_m$ est presque *pure*, la plupart des observations proviennent d'une seule classe, alors l'indice de Gini et l'entropie prendraient des valeurs plus petites que le taux de mauvais classement
- L'indice de Gini et l'enropie sont plus sensibles à la pureté des nœuds
- Pour évaluer la qualité d'une partition, l'indice de Gini et l'entropie sont souvent utilisés comme mesure d'erreur (plus que le taux de mauvais classement)
- Chacune de ces trois mesures peut être utilisée lors de l'élagage d'un arbre
- Le taux de mauvais classement est préférable si on vise une meilleure précision de prédiction de l'arbre élagué final

Jeu de données Pima

```
rm(list=ls())
require(rpart)
require(rpart.plot)
require(MASS)
data(”Pima.tr”)
data(”Pima.te”)
```

- Reprendre les étapes de l’exemple de régression pour ajus ter un arbre de décision visant à prédire `type` (diabète : Yes or No) en fonction des autres variables présentes dans le jeu de données..

Avantages et inconvénients des arbres

- ▲ Les arbres sont faciles à expliquer à n'importe qui. Ils sont plus faciles à expliquer que les modèles linéaires
- ▲ Les arbres peuvent être représentés graphiquement, et sont interprétables même par des non-experts
- ▲ Ils peuvent gérer des variables explicatives catégorielles sans introduire des variables binaires
- ▼ Malheureusement, ils n'ont pas la même qualité prédictives que les autres approches d'apprentissage.

Cependant, en agrégeant plusieurs arbres de décision, les performances prédictives s'améliorent substantiellement.

Outline

Agrégation par Bagging

- L'agrégation bootstrap ou bagging est méthode de réduction de la variance en apprentissage statistique. Elle est particulièrement utile sur les arbres de décision.
- Rappelons que, sur un ensemble de n observations indépendantes $Z_1, \ldots, Z_n$, chacune de variance σ^2, la variance de la moyenne $\bar{Z}$ est σ^2/n.
- En pratique, il n'est pas possible de moyenner des arbres de décision construits sur de multiples ensembles d'entrainement (pas assez de données observées)

Bagging pour la régression

- Au lieu de cela, on peut bootstrapper en ré-échantillonnant plusieurs fois les données d'entrainement.
- Alors, à partir de B échantillons bootstrap, on entraine une méthode d'apprentissage pour ajuster B fonctions de régressions, notées $\hat{f}^{*b}(x)$, $b = 1, \ldots, B$
- La fonction de régression *bagguée* est alors

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x)$$

Bagging pour la classification

- Sur un problème de classification, $\hat{f}^{*b}(x)$ renvoie une classe possible pour chaque échantillon bootstrap b.
- La décision finale $\hat{f}_{\text{bag}}(x)$ se prend par un vote à la majorité simple parmi les B prédictions des classifieurs bootstrap.

Intuitivement

- Cela fonctionne mieux pour les méthodes d'apprentissage à faible biais et à forte variance
- C'est le cas des arbres de décision, en particulier les arbres profonds.
- Sur des gros jeux de données d'entrainement, faire parfois du sous-échantillonnage bootstrap.

Erreur *Out-Of-Bag* (OOB)

- Il y a une façon simple d'estimer l'erreur de test quand on fait du bagging.
- La clé du bagging est l'entrainement de nombreux $\hat{f}(x)$ sur des échantillons bootstraps. On peut donc utiliser les observations hors du $b^{\text{ième}}$ bootstrap pour évaluer chaque $\hat{f}^{*b}(x)$.
- Ce qui donne l'algorithme ci-dessous.

    1. Pour chaque observation (x_i, y_i), calculer $\hat{y}_i^{\text{oob}}$ la prédiction en n'utilisant que les estimateurs $\hat{f}^{*b}(x)$ qui n'ont pas vu cette observation dans leur entrainement
    2. Évaluer l'erreur entre $\hat{y}_i^{\text{oob}}$ et les y_i (erreur quadratique moyenne ou taux de mauvaise classification)

Erreur *Out-Of-Bag* pour l'estimation de l'erreur de test

- La probabilité qu'une observation i ne fasse pas partie d'un échantillon bootstrap est de $\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e}$.
- Le nombre d'observations qui ne font pas partie d'un tirage bootstrap est $n\left(1 - \frac{1}{n}\right)^n \approx \frac{n}{e}$. Ces observations sont dites *out-of-bag*.
- Sur B tirages bootstrap, il y a environ $\frac{B}{e}$ échantillon qui ne contiennent pas l'observation i.
- Les arbres de décisions ajustés sur ces échantillons servent à prédire la réponse de l'observation i. Il y a environ $\frac{B}{e}$ prédictions.
- On fait la moyenne des ces prédictions pour la régression ou prendre le vote à majorité simple pour la classification pour calculer la prédiction *bagguée* de l'observation i qu'on notera $\hat{f}^*\left(x_i\right)$.

Estimation de l'erreur de test par OOB

- L'erreur quadratique moyenne ($\propto$ moindres carrés) OOB pour la régression

$$\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \hat{f}^*(x_i)\right)^2.$$

- L'erreur de classification OOB

$$\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\{y_i \neq \hat{f}^*(x_i)\}.$$

- L'erreur OOB est l'équivalente d'une erreur de test.
- Lorsque B est grand, on peut montrer que l'erreur OOB est équivalente à l'erreur calculée par validation-croisée one-leave-one-out.

Mesurer l’importance des variables

- Le bagging améliore la précision d’un modèle au détriment de son interprétation
- On peut obtenir un résumé général de l’importance d’une variable à l’aide des moindres carrés pour le bagging d’arbres de régression et l’indice de Gini pour le bagging d’arbres de classification.
- Pour chaque arbre de régression (ou classification) ajusté sur un échantillon bootstrap, on calcule le nombre de fois où les moindres carrés (ou l’indice de Gini pour la classification) a diminué par une partition d’une variable j. On fait la moyenne de cet indicateur sur les B échantillons bootstraps.
- Une grande valeur de cet indicateur indique une importance de la variable j

Garanties théoriques : un peu de notations

- On note l'échantillon $\mathcal{D}_n = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ et on rappelle la fonction de régression

$$m^*(x) = \mathbb{E}\big[Y|X = x\big].$$

- Pour $x \in \mathbb{R}^p$, on considère l'erreur quadratique moyenne d'un estimateur $\hat{m}$ et sa décomposition biais-variance

$$\mathbb{E}\Big[\big(\hat{m}(x) - m^*(x)\big)^2\Big] = \Big(\mathbb{E}\big(\hat{m}(x)\big) - m^*(x)\Big)^2 + \mathrm{Var}\big(\hat{m}(x)\big).$$

- Soit l'estimateur $\hat{m}_{\mathrm{bag}}(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{m}_b(x)$ obtenue par l'aggrégation des régresseurs $\hat{m}_1, \ldots, \hat{m}_B$. Remarquons que si on suppose les régresseurs $\hat{m}_1, \ldots, \hat{m}_B$ i.i.d, on a

$$\mathbb{E}\big[\hat{m}_{\mathrm{bag}}(x)\big] = \mathbb{E}\big[\hat{m}_1(x)\big] \quad \text{et} \quad \mathrm{Var}\big[\hat{m}_{\mathrm{bag}}(x)\big] = \frac{1}{B}\mathrm{Var}\big[\hat{m}_1(x)\big].$$

même biais mais la variance diminue

Garanties théoriques : boostrap

- Le fait de considérer des échantillons bootstrap introduit un aléa supplémentaire dans l'estimateur. Afin de prendre en compte cette nouvelle source d'aléatoire, on note $\theta_b = \theta_b\left(\mathcal{D}_n\right)$ l'échantillon bootstrap de l'étape b et $\hat{m}\left(\cdot, \theta_b\right)$ l'estimateur construit à l'étape b. On écrira l'estimateur final $\hat{m}_B(x) = \frac{1}{B}\sum_{b=1}^{B} \hat{m}(x, \theta_b)$.
- Conditionnellement à $\mathcal{D}_n$, les $\theta_1, \ldots, \theta_B$ sont i.i.d. Par la loi des grands nombres

$$\lim_{B\to\infty} \hat{m}_B(x) = \lim_{B\to\infty} \frac{1}{B}\sum_{b=1}^{B} \hat{m}(x, \theta_b) = \mathbb{E}_\theta\left[\hat{m}(x, \theta)\middle|\mathcal{D}_n\right] \qquad \text{p.s.}$$

- L'espérance est ici calculée par rapport à la loi de θ. On déduit de ce résultat que, contrairement au boosting, prendre B trop grand ne va pas sur-ajuster l'échantillon. Dit brutalement, prendre la limite en B revient à considérer un estimateur *moyen* calculé sur tous les échantillons bootstrap. Le choix de B n'est donc pas crucial pour la performance de l'estimateur, il est recommandé de le prendre le plus grand possible (en fonction du temps de calcul).

Garanties théoriques : premier résultat

- Deux techniques sont généralement utilisées pour générer les échantillons bootstrap
  1. $\theta_b\left(\mathcal{D}_n\right)$ est obtenu en tirant n observations avec remise dans $\mathcal{D}_n$, chaque observation ayant la même probabilité d'être tirée $\frac{1}{n}$.
  2. $\theta_b\left(\mathcal{D}_n\right)$ est obtenu en tirant ℓ observations (avec ou sans remise) dans D_n avec $\ell < n$.

- ***Théorème de Biau & Devroye (2010)*** *Si $\ell = \ell_n$ tel que $\lim_{n\to\infty} \ell_n = +\infty$ et $\lim_{n\to\infty} \frac{\ell_n}{n} = 0$ alors l'estimateur $\hat{m}(x) = \mathbb{E}_\theta\left[\hat{m}(x,\theta)|\mathcal{D}_n\right]$ est universellement consistant.*

Garanties théoriques : biais et variance

- $\sigma^2(x) = \text{Var}\big(\hat{m}(x, \theta_b)\big)$
- $\rho(x) = \text{corr}\Big(\hat{m}(x, \theta_1), \hat{m}(x, \theta_2)\Big)$, le coefficient de corrélation entre deux estimateurs que l'on agrège (calculés sur deux échantillons bootstrap).
- La variance $\sigma^2(x)$ et la corrélation $\rho(x)$ sont calculées par rapport aux lois de $\mathcal{D}_n$ et de θ. On suppose que les estimateurs $\hat{m}(x, \theta_1), \ldots, \hat{m}(x, \theta_B)$ sont identiquement distribués.
-
- ***Proposition*** *On a :*

$$Var_B\big(\hat{m}_B(x)\big) = \rho(x)\sigma^2(x) + \frac{1 - \rho(x)}{B}\sigma^2(x).$$

Par conséquent

$$Var\big[\hat{m}(x)\big] = \rho(x)\sigma^2(x).$$

Exemple : reprendre les deux exemples Boston et bes

- La fonction **bagging** du package **ipred** permet du faire du bagging

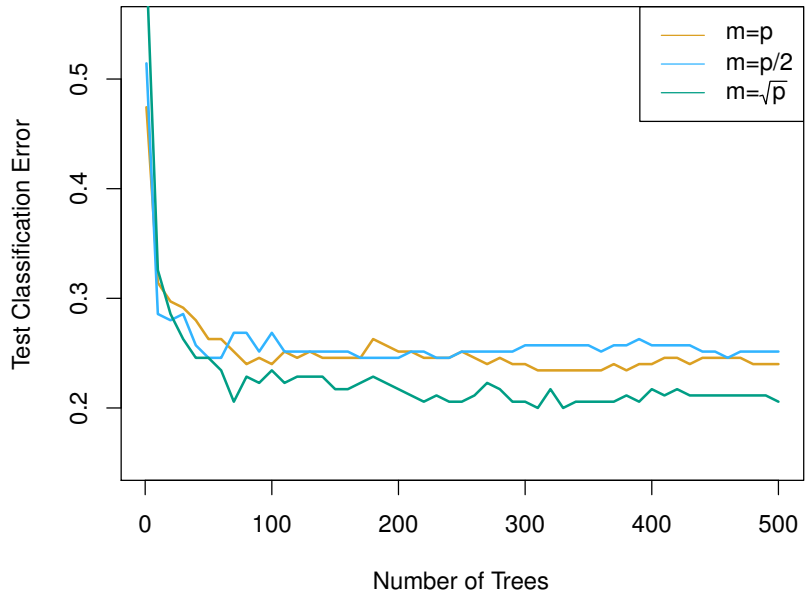
Forêts aléatoires très proche du bagging

- C'est la même idée que le bagging à l'exception …
- À chaque partition, on ne considère que m variables explicatives au hasard parmi les p variables explicatives du problème.
- Souvent $m \approx \sqrt{p}$.

Forêts aléatoires

- À chaque pas, la partition est contrainte sur un petit nombre m de variables explicatives choisies au hasard.
- Permet d'avoir des arbres différents.
- Deux arbres similaires sont hautement corrélés, la moyenne d'arbres hautement corrélés ne peut produire une réduction importante de la variance. Penser au cas extrême où tous les arbres sont les mêmes.
- La moyenne d'arbres non-corrélés ou faiblement corrélés permet une réduction importante de la variance.
- Une forêt aléatoire produit des arbres moins corrélés.
- Une forêt aléatoire est équivalente à un bagging si $m = p$.

Illustration : données d'expression de gênes



- Résultats de forêts aléatoires pour prédire les 15 classes à partir du niveau d'expression de 500 gènes
- L'erreur de test (évaluée par OOB) dépend du nombre d'arbres. Les différentes couleurs correspondent à différentes valeurs de m.
- Les forêts aléatoires améliorent significativement le taux d'erreur de CART (environ 45.7$\%$)

Table of Contents