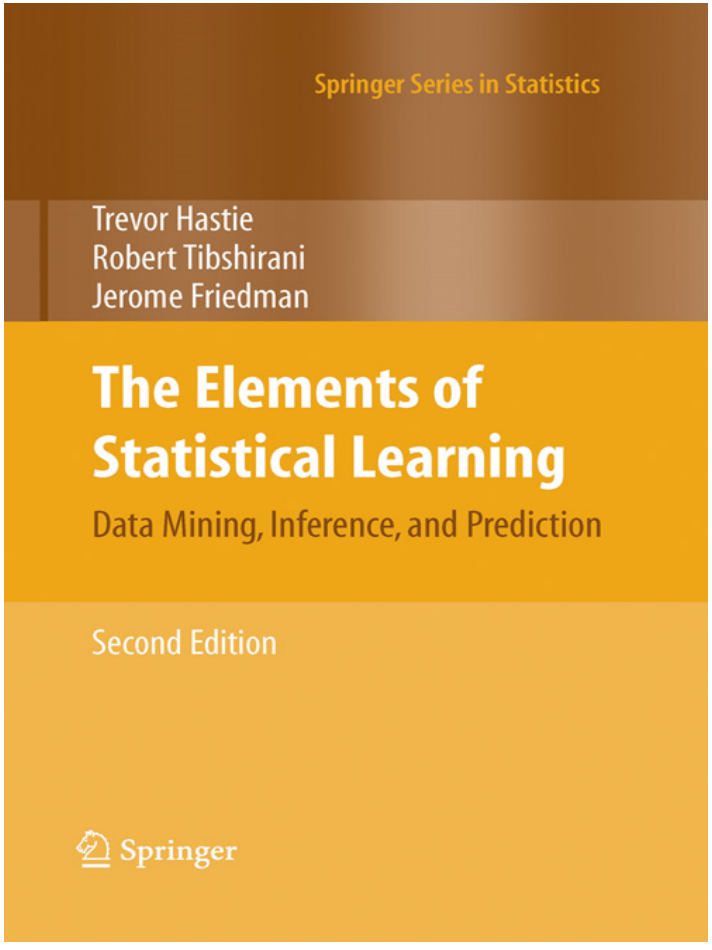
Apprentissage statistique

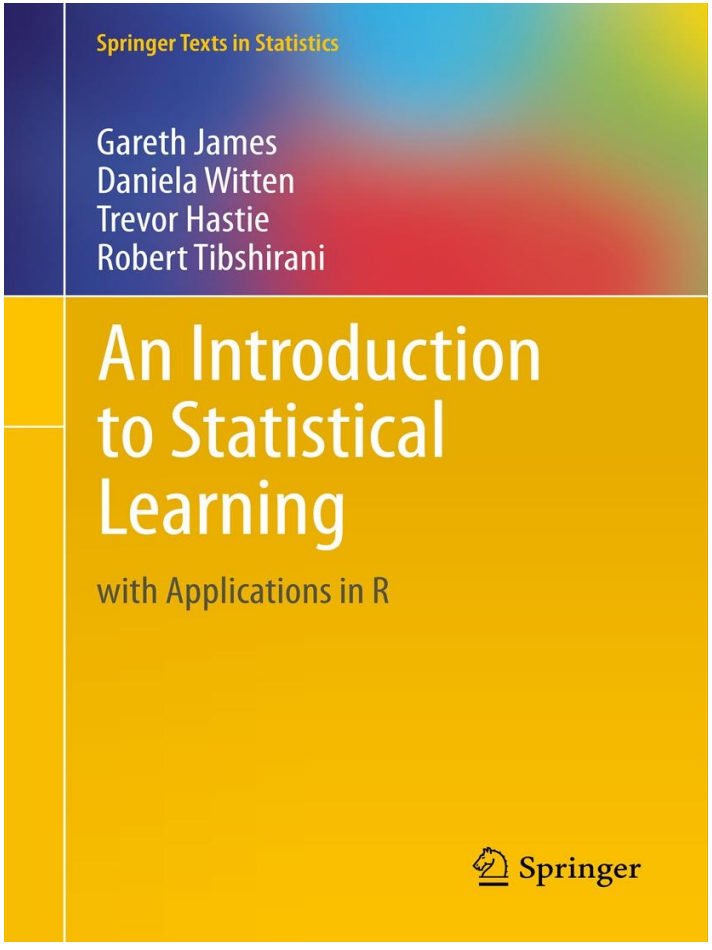
masedki.github.io

Univesité Paris-Saclay

Année 2021/2022

Références


Springer Series in Statistics
Trevor Hastie
Robert Tibshirani
Jerome Friedman
The Elements of Statistical Learning
Data Mining, Inference, and Prediction
Second Edition
Springer


Springer Texts in Statistics
Gareth James
Daniela Witten
Trevor Hastie
Robert Tibshirani
An Introduction to Statistical Learning
with Applications in R
Springer

Problèmes d'apprentissage statistique

- Identifier les facteurs de risque du cancer de la prostate
- Classifier des phonèmes à partir de périodogrammes
- Prédire si une personne est sujette aux crises cardiaques, à partir de mesures cliniques, son régime et des données démographiques
- Personnaliser un système de détection de spam email
- Lecture de codes postaux écrits à la main
- Classification d'échantillons de tissus dans différents types de cancer, en fonction de données d'expression de gènes
- Établir une relation entre salaires et variables démographiques
- Classifier les pixels d'une image satellite

Question

- Sur 4601 mails, on a pu identifier 1813 spams.
- On a également mesuré sur chacun de ces mails la présence ou absence de 57 mots.

Peut-on construire à partir de ces données une méthode de détection automatique de spam ?

Représentation du problème

La plupart de ces problèmes peuvent être appréhendés dans un contexte de régression : on cherche à expliquer une variable Y par d'autres variables dites explicatives $X_1, \ldots, X_p$:

Y	X
Chiffre	image
Mot	courbe
Spam ou pas	présence/absence d'un ensemble mots
Type de leucémie	expressions de gênes

- Lorsque la variable à expliquer est quantitative, on parle de régression.
- Lorsqu'elle est qualitative, on parle de discrimination ou classification supervisée.

Régression

- Un échantillon i.i.d d'apprentissage $(X_1, Y_1), \ldots, (X_n, Y_n)$ d'une loi conjointe $\mathbb{P}$ *inconnue* sur $\mathbb{R}^p \times \mathbb{R}$.
- Objectif : Prédire ou expliquer la variable Y à partir d'une nouvelle observation X.
- Méthode : construire une règle de prédiction (ou régression)

$$m : \mathbb{R}^p \mapsto \mathbb{R}.$$

- Soit $\ell : \mathbb{R} \times \mathbb{R} \mapsto \mathbb{R}_+$ une fonction de perte (i.e, $\ell(y, y') = 0$ et $\ell(y, y') > 0$ pour $y \neq y'$), par exemple

$$\ell(y, y') = |y - y'|^q$$

(perte absolue si $q = 1$ et perte quadratique $q = 2$).

Risque ou erreur de généralisation

- Le risque ou erreur de généralisation d'une règle de décision (ou prédiction) m est défini par

$$R_{\mathbb{P}}(m) = \mathbb{E}_{(X,Y)}\Big[\ell\big(Y, m(X)\big)\Big].$$

La fonction de régression

- Un champion
$$m^*(x) = \mathbb{E}\big[Y|X = x\big]$$
appelé fonction de régression.
- Pour toute autre fonction m, on a
$$\mathbb{E}\left[\big(Y - m^*(X)\big)^2\right] \leq \mathbb{E}\left[\big(Y - m(X)\big)^2\right]$$
.

La fonction de régression

Nous avons

$$\mathbb{E}_{X,Y}\Big[\Big(Y - m(X)\Big)^2\Big] = \mathbb{E}_X \mathbb{E}_{Y|X}\Big[\Big(Y - m(X)\Big)^2 \mid X\Big]$$

Donc il suffit de minimiser cette erreur ponctuellement en X

$$m(x) = \text{argmin}_c \mathbb{E}_{Y|X}\Big[\Big(Y - c\Big)^2 \mid X = x\Big].$$

La solution est donnée par

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

La classification binaire

- Un échantillon i.i.d d'apprentissage $(X_1, Y_1), \ldots, (X_n, Y_n)$ d'une loi conjointe $\mathbb{P}$ inconnue sur $\mathbb{R}^p \times \{0, 1\}$.
- Objectif : Prédire ou expliquer la variable Y à partir d'une nouvelle observation X.
- Méthode : construire une règle classification (ou décision)

$$g : \mathbb{R}^p \mapsto \{0, 1\}.$$

- La fonction de perte binaire $\ell(y, y') = 1_{y \neq y}$.
- Risque associé à g : taux de mauvais classement

$$R_{\mathbb{P}}(g) = \mathbb{E}\Big[\ell\big(g(X), Y\big)\Big] = \mathbb{P}\big(g(X) \neq Y\big).$$

La règle de Bayes

- Un champion appelé règle de Bayes

$$g^*(x) = \begin{cases} 1 & \text{si} \quad \eta(x) \geq \frac{1}{2} \\ 0 & \text{sinon,} \end{cases}$$

où $\eta(x) = \mathbb{P}(Y = 1 | X = x)$.

- Quelque soit la règle de décision g, nous avons

$$R_{\mathbb{P}}(g^*) = \mathbb{P}\big(g^*(X) \neq Y\big) \leq \mathbb{P}\big(g(X) \neq Y\big) = R_{\mathbb{P}}(g).$$

Règle de Bayes : un théorème

- Pour toute règle de classification $g : \mathcal{X} \mapsto \mathcal{Y}$, pour la fonction de perte binaire, nous avons

$$R(g) - R(g^*) = \mathbb{E}_X \left[1\Big\{ g(X) \neq g^*(X) \Big\} \Big| 2\eta(X) - 1 \Big| \right].$$

- Interpréter ce résultat lorsque

$$\eta(x) = \frac{1}{2}, \forall x \in \Big\{ x \in \mathcal{X} : g(x) \neq g^*(x) \Big\}$$

Début de preuve

On remarque que :

$$\begin{aligned}\mathbb{E}_{Y|X=x}\Big[1\big\{Y = g^*(x)\big\}\Big] &= \mathbb{P}_{Y|X}\big[Y = g^*(x)\big] \\ &= \begin{cases}\eta(x) & \text{si} \quad \eta(x) \geq \frac{1}{2} \\ 1-\eta(x) & \text{sinon}\end{cases} \\ &= \frac{1}{2} + \big|\eta(x) - \frac{1}{2}\big|.\end{aligned}$$

Rappel :

$$\mathbb{E}_{X,Y} h(X,Y) = \mathbb{E}_X \mathbb{E}_{Y|X} h(X,Y).$$

suite de la preuve : voir notes

Proposition

$$R^* = R(g^*) = \mathbb{E}_X \left[\min \left\{ \eta(X), 1 - \eta(X) \right\} \right]$$

Preuve : voir notes

Problème majeur !!

- Problème: m^* est inconnu en pratique. Il faut construire un régresseur $\widehat{m}_n$ à partir des données $(X_1, Y_1), \ldots, (X_n, Y_n)$, tel que

$$\widehat{m}_n(x) \approx m^*(x).$$

- Problème: g^* est inconnue en pratique. Il faut construire une règle $\widehat{g}_n$ à partir des données $(X_1, Y_1), \ldots, (X_n, Y_n)$, tel que

$$\widehat{g}_n(x) \approx g^*(x).$$

Un candidat naturel

À partir des expressions de m^* et g^*, proposer deux estimateurs intuitifs.

Décomposition de l'erreur

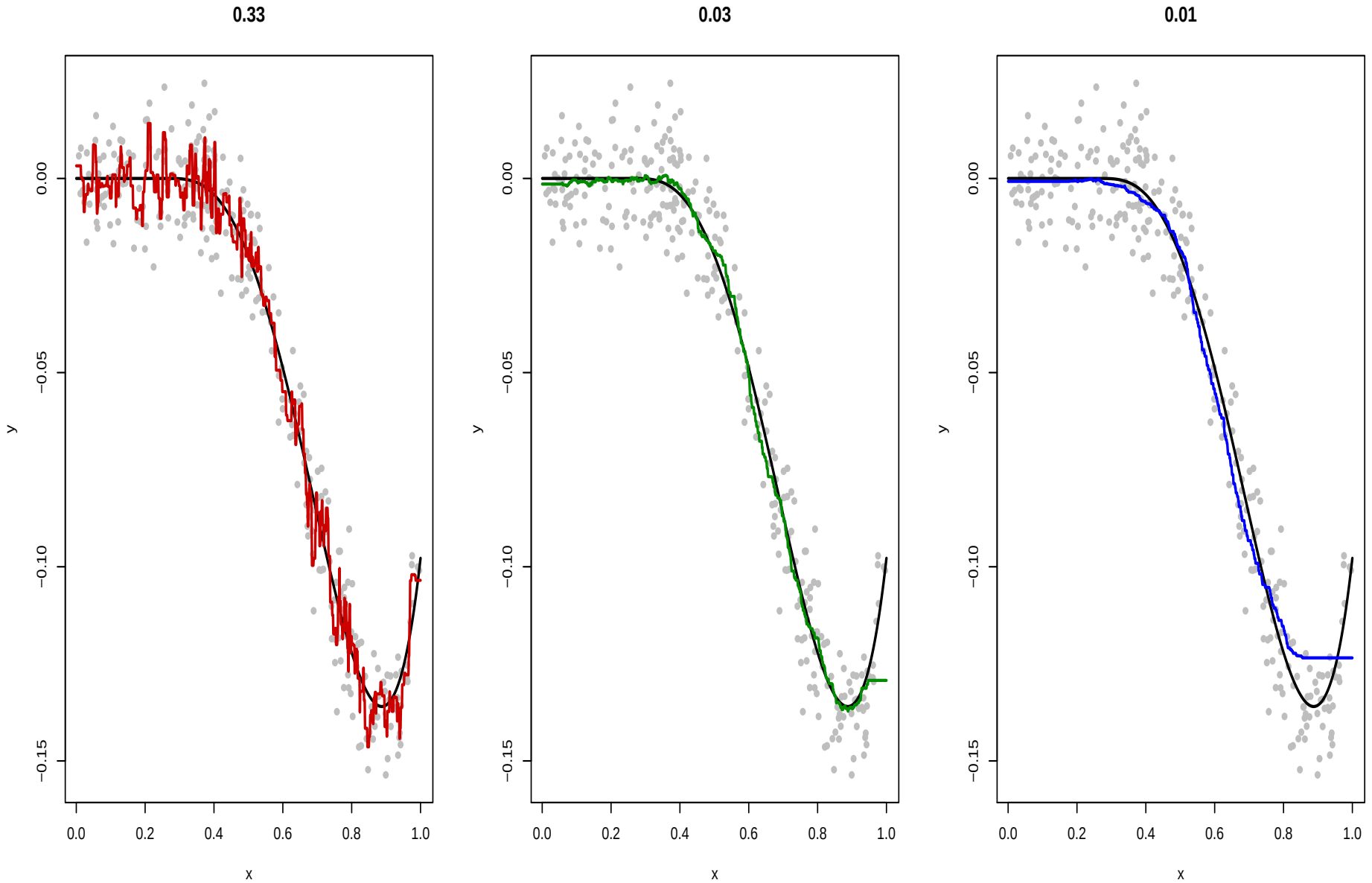
Pour tout estimateur $\widehat{m}_n(x)$ de $m^*(x)$ à x fixé, nous avons

$$\begin{aligned}
\mathbb{E}\Big[\Big(m^*(x) - \widehat{m}_n(x)\Big)^2\Big] &= \big[m^*(x)\big]^2 - 2m^*(x)\mathbb{E}\big[\widehat{m}_n(x)\big] \\
&\quad + \mathbb{E}\Big[\big(\widehat{m}_n(x)\big)^2\Big] \\
&= \Big[m^*(x) - \mathbb{E}\big(\widehat{m}_n(x)\big)\Big]^2 \\
&\quad + \mathbb{E}\Big[\big(\widehat{m}_n(x)\big)^2\Big] - \Big[\mathbb{E}\big(\widehat{m}_n(x)\big)\Big]^2 \\
&= \Big(\text{biais}\Big)^2 + \text{Var}\big[\widehat{m}_n(x)\big]
\end{aligned}$$

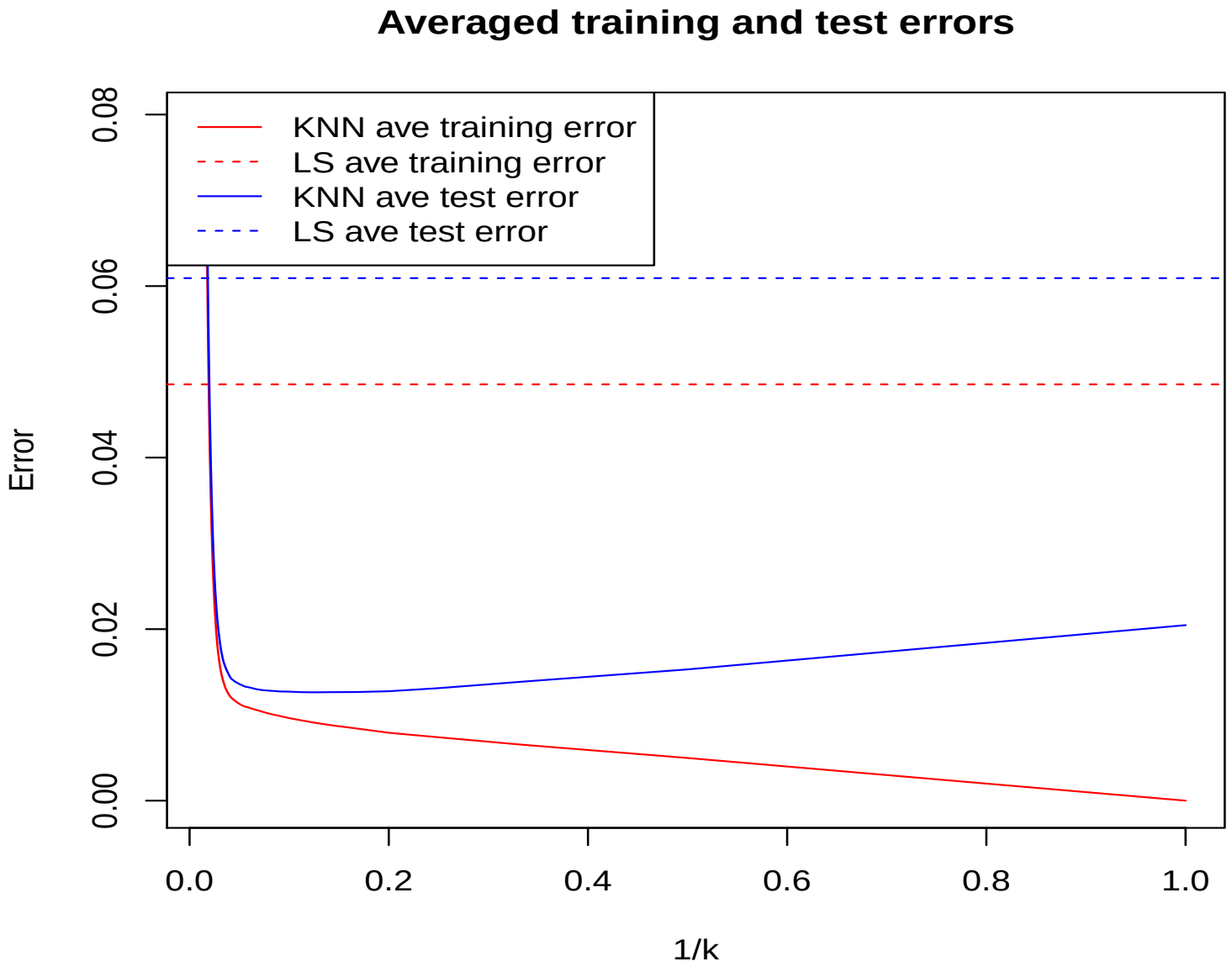
Notations

- On s'intéresse au cas où on cherche à expliquer une variable qualitative Y par p variables explicatives $X_1, \ldots, X_p$.
- Y est à valeurs dans un ensemble discret fini de modalités qui peuvent être numérotées par des les indices $\{1, 2, \ldots, K\}$ et les variables $X_1, \ldots, X_p$ peuvent être qualitatives et/ou quantitatives.
- Néanmoins, pour présenter les méthodes, on se restreint au cas où Y est à 2 modalités (0 et 1).

Complexité d'un modèle (compromis biais variance)



Évaluation de la précision : phénomène de sur-apprentissage



Évaluer la précision : premier pas

Supposons que l'on ajuste un modèle $\widehat{f}(x)$ sur des données d'apprentissage $\text{Tr} = \{(x_1, y_1), \ldots, (x_N, y_N)\}$.

Performance de $\widehat{f}$?

Première idée : erreur moyenne de prédiction sur Tr :

$$\text{MSE}_{\mathbf{Tr}} = \text{Moyenne}_{i \in \mathbf{Tr}} \left(y_i - \widehat{f}(x_i) \right)^2$$

OPTIMISTE

(sur-apprentissage)

Meilleure idée : sur un jeu de données de *test*,
$\text{Te} = \{(x_{N+1}, y_{N+1}), \ldots, \}$,
indépendant de Tr :

$$\text{MSE}_{\mathbf{Te}} = \text{Moyenne}_{i \in \text{Te}} \left(y_i - \widehat{f}(x_i) \right)^2$$

Les Knn sont victimes du fléau de la dimension

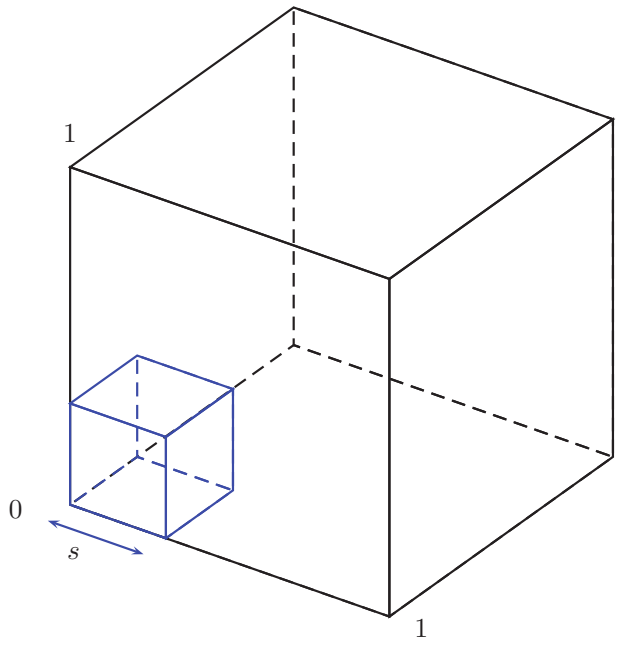
- Ces méthodes, basées sur des moyennes autour des voisins sont plutôt bonnes si
  - petite dimension $p \leq 4$
  - grand échantillon $n \gg p$
- des versions lissées, obtenues par
  - méthodes à noyaux
  - lissage par splines,
  - ...

Raison. le *fléau de la dimension*. Les voisins les plus proches peuvent être éloignés en grande dimension

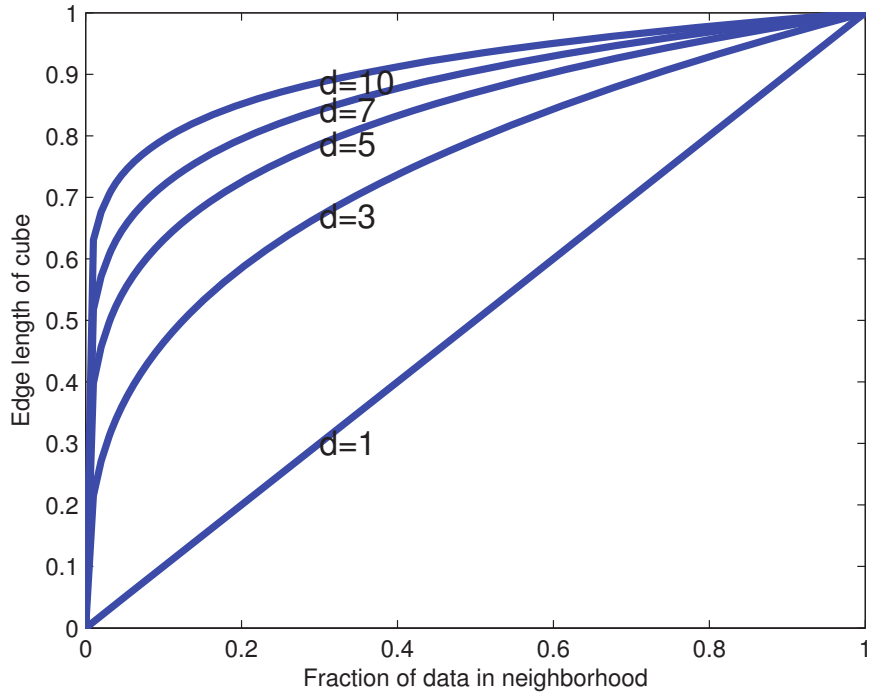
- Il faut une quantité raisonnable de valeurs de y_i à moyenner pour que $\widehat{f}(x)$ ait une faible variance
- En grande dimension, pour obtenir cette quantité d'observation, il faut s'éloigner beaucoup de x.

On perd l'idée de moyenne **locale** autre de $X = x$.

Le fléau de la dimension



(a)



(b)

Validation croisée

But

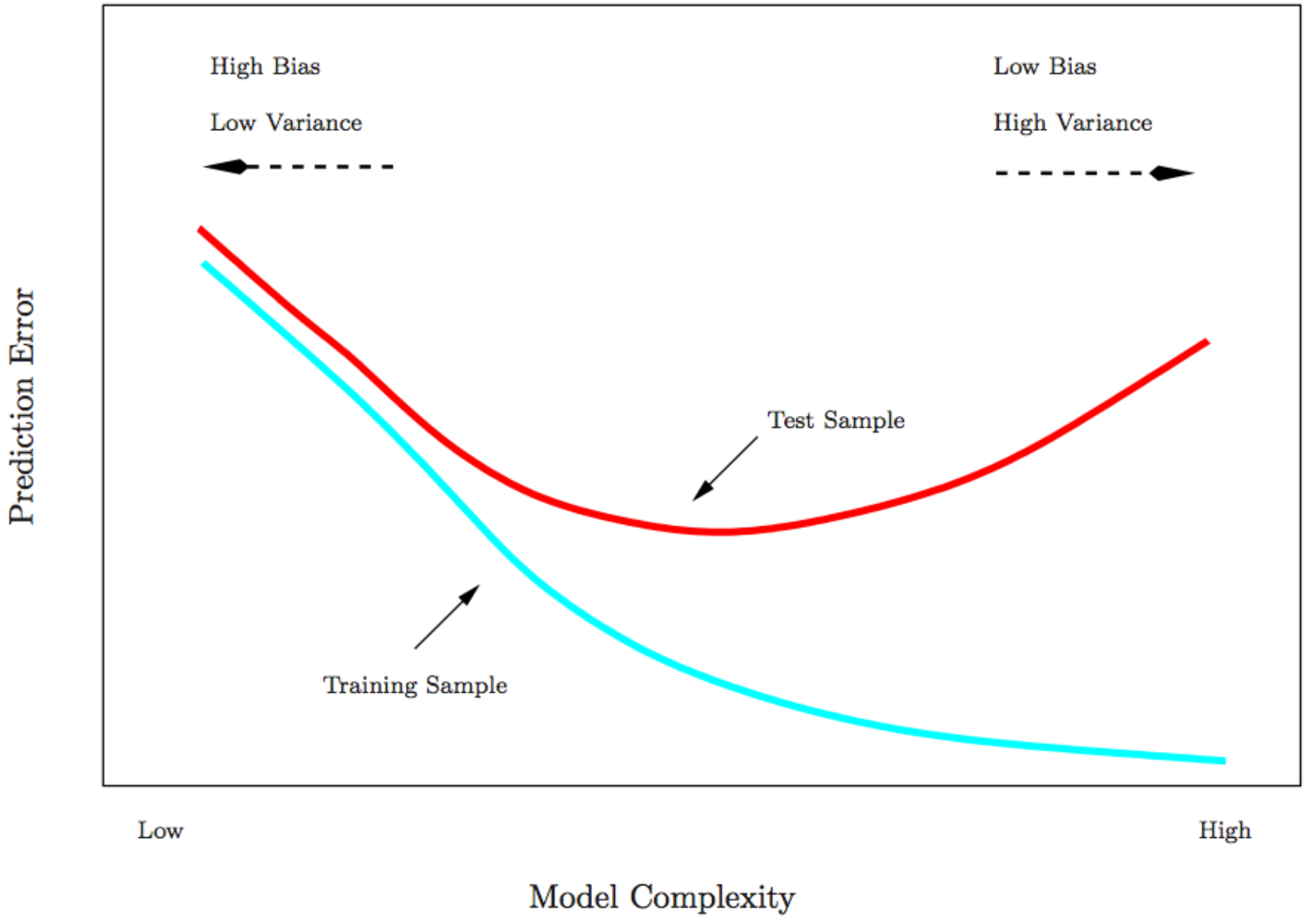
- Dans cette partie, nous allons discuter de deux méthodes de *ré-échantillonnage* : la validation croisée et le bootstrap
- Ces méthodes ré-ajustent le modèle que l'on souhaite sur des échantillons issus de l'échantillon d'apprentissage, dans le but d'obtenir des informations supplémentaires sur ce modèle
- Par exemples, ces méthodes fournissent des estimations de l'erreur sur des ensembles de test, le biais et la variance des estimations de paramètres...

Erreur d'entraînement et erreur de test

On rappelle la différence entre *erreur de test* et *erreur d'entraînement* :

- L'*erreur de test* est l'erreur moyenne commise par une méthode d'apprentissage statistique pour prédire une réponse sur une nouvelle observation, qui n'a pas été utilisée pour ajuster le modèle.
- En revanche, l'*erreur d'entraînement* peut être facilement calculée en appliquant la méthode d'apprentissage sur les données d'entraînement.
- Mais l'erreur d'entraînement est souvent bien différente de l'erreur de test, et en particulier, l'erreur d'entraînement sous-estime parfois grandement l'erreur de test — on parle d'erreur trop *optimiste*.

Erreur d'entraînement et erreur de test



Estimations de l'erreur de prédiction

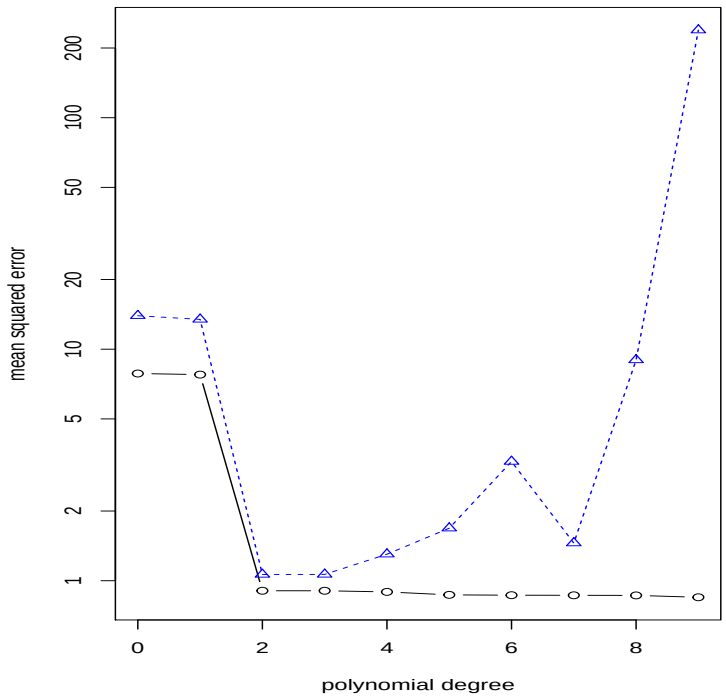
- La meilleure solution : un grand ensemble de test clairement désigné. Bien souvent, ce n'est pas disponible.
- Certaines méthodes permettent de corriger l'erreur d'entrainement pour estimer l'erreur de test, avec des arguments fondés mathématiquement.
  Cela inclut les C_p *de Mallows*, les critères AIC et BIC. Ils seront discutés plus tard.
- Ici, nous nous intéressons à une classe de méthodes qui estime l'erreur de test en mettant de côté un sous-ensemble des données d'entraînement disponibles pour ajuster les modèles, et en appliquant la méthodes ajustée sur ces données mises de côté.

Approche par ensemble de validation

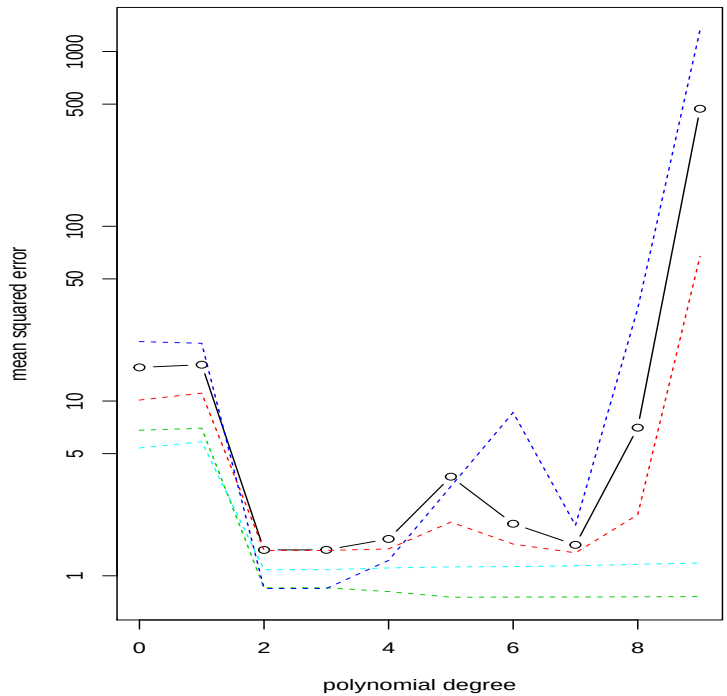
- Cette méthode propose de diviser l'échantillon d'apprentissage en deux : un ensemble d'entraînement et un ensemble de validation
- Le modèle est ajusté sur l'ensemble d'entraînement, et on l'utilise ensuite pour prédire les réponses sur l'échantillon de validation.
- L'erreur obtenue en comparant prédiction et observation sur cet échantillon de validation approche l'erreur de test. On utilise typiquement des moindres carrés (MSE) en régression et des taux de mauvaises classification si la réponse est qualitative (ou une fonction de coût d'erreur)

Exemple sur les données simulées (degré 2)

- On veut comparer la régression linéaires à des régressions polynomiales de différents degrés
- On divise en deux les 200 observations : 100 pour l'entrainement, 100 pour le test.



Sur une partition aléatoire



Variabilité d'une partition à l'autre

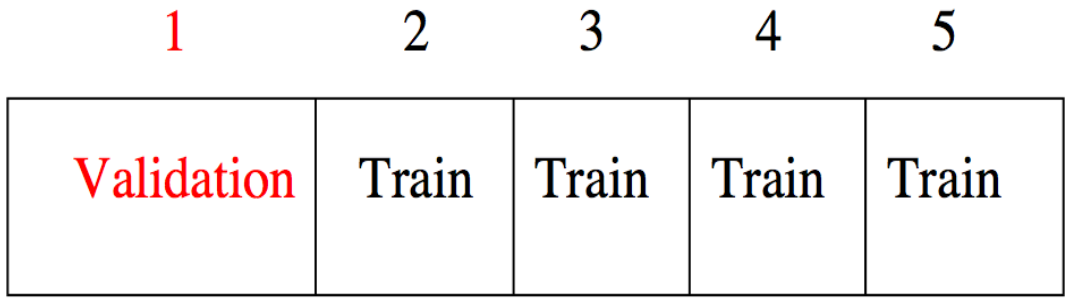
Inconvénients de l'approche par ensemble de validation

- L'estimation obtenue par cette méthode peut être très variable, et dépend de la chance ou malchance dans la construction du sous-échantillon de validation
- Dans cette approche, seule une moitié des observations est utilisée pour ajuster les modèles — celles qui sont dans l'ensemble d'entrainement.
- Cela suggère que l'erreur calculée peut surestimer l'erreur de test d'un modèle ajusté sur l'ensemble des données (moins de variabilité d'échantillonnage dans l'inférence des paramètres du modèle)

Déjà mieux : échanger les rôles entrainement-validation et faire la moyenne des deux erreurs obtenues. On *croise* les rôles.

Validation croisée à K groupes

- C'est la méthode la plus couramment utilisée pour estimer l'erreur de test
- L'estimation peut être utilisée pour choisir le meilleur modèle (la meilleure méthode d'apprentissage), ou approcher l'erreur de prédiction du modèle finalement choisi.
- L'idée est de diviser les données en K groupes de même taille. On laisse le k-ème bloc de côté, on ajuste le modèle, et on l'évalue sur le bloc laissé de côté.
- On répète l'opération en laissant de côté le bloc $k = 1$, puis $k = 2, \ldots$ jusqu'à $k = K$. Et on combine les résultats



Détails

- Pour chacune des observations, on obtient une prédiction $\widehat{y_i} = \widehat{f}(x_i)$ ou $\widehat{g}(x_i)$ au moment où i est dans le groupe mis de côté, et une seule prédiction.
- On compare alors ces prédictions aux observations comme pour l'erreur de test

$$MSE_{(K)} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \widehat{f}(x_i))^2$$

ou

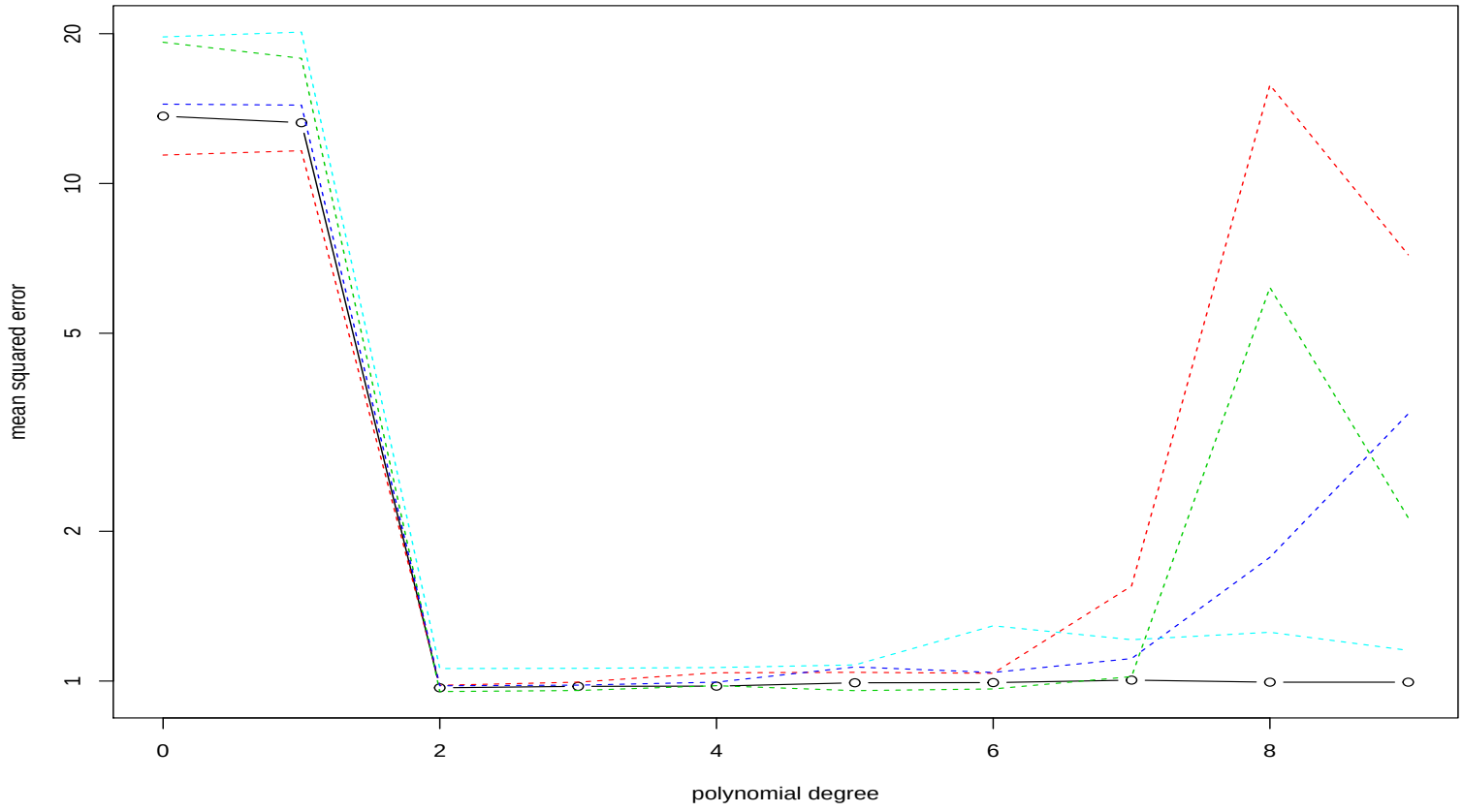
$$\tau_{(K)} = \frac{1}{n}\sum_{i=1}^{n} 1\{y_i \neq \widehat{g}(x_i)\}$$

- Lorsque $K = n$, on parle de *leave-one out cross-validation*(LOOCV)

Danger avec le leave-one out !

- On dit que LOOCV ne secoue pas assez les données. En effet, les classifiers $\widehat{C}$ ou les fonctions de régression inférées $\widehat{f}$ avec $(n-1)$ données sont très corrélés les uns aux autres.
- On ne voit plus l'erreur d'échantillonnage, autrement dit la variabilité de l'estimation de la fonction. C'était pourtant tout l'intérêt de la validation croisée. On choisit généralement $K = 5$ ou $K = 10$ blocs.

Retour au jeu de données simulé



En cas d'égalité, choisir le modèle le plus *parcimonieux* car il aura naturellement moins de variance d'estimation dans les coefficients du modèle.

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

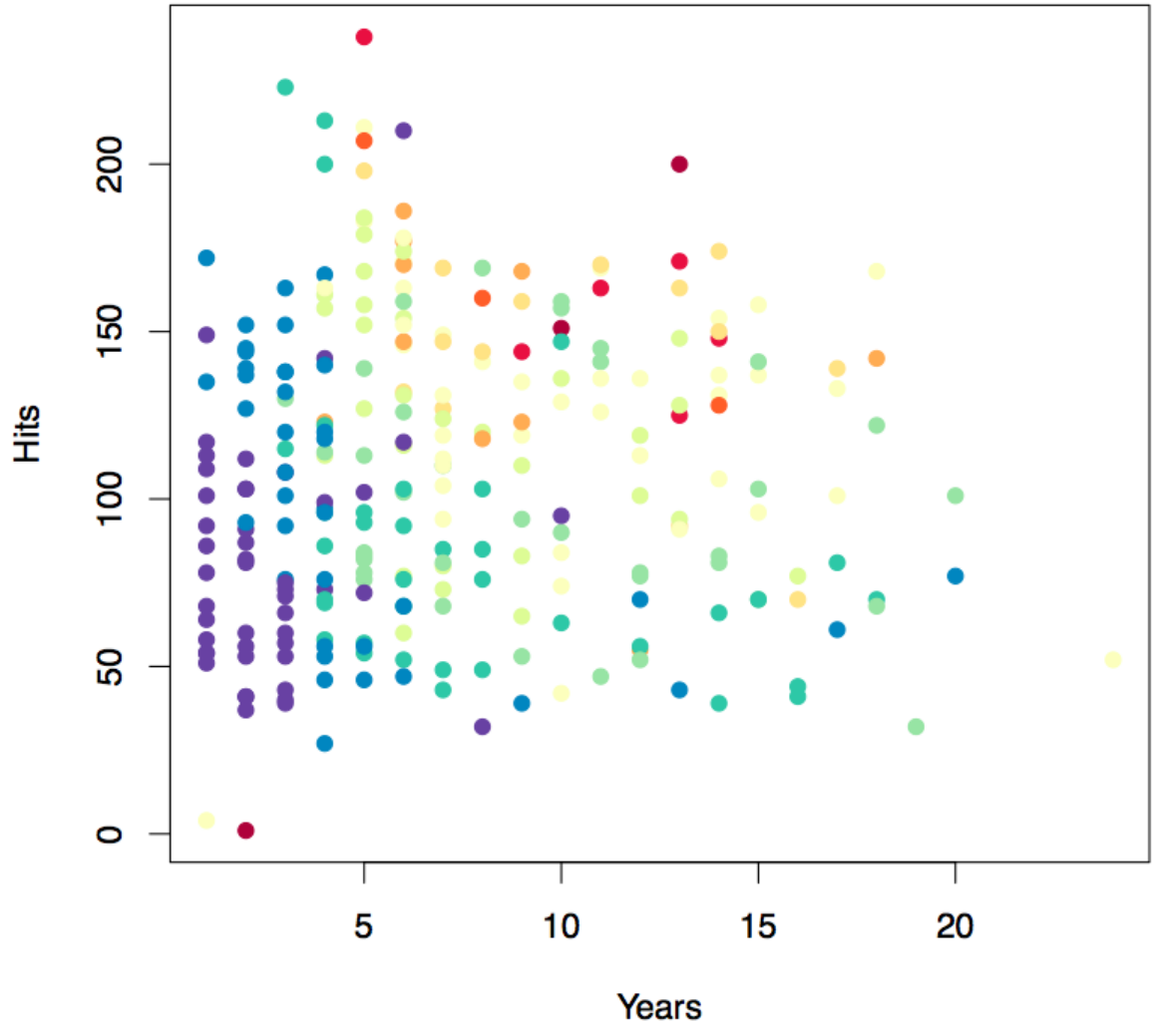
- Les méthodes basées sur des arbres sont simples et utiles pour l'interprétation.
- Cependant, elles ne sont pas capables de rivaliser avec les meilleures approches d'apprentissage supervisé en terme de qualité de prédiction.
- Nous discuterons donc aussi de *bagging*, *forêts aléatoires (random forests)*, 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

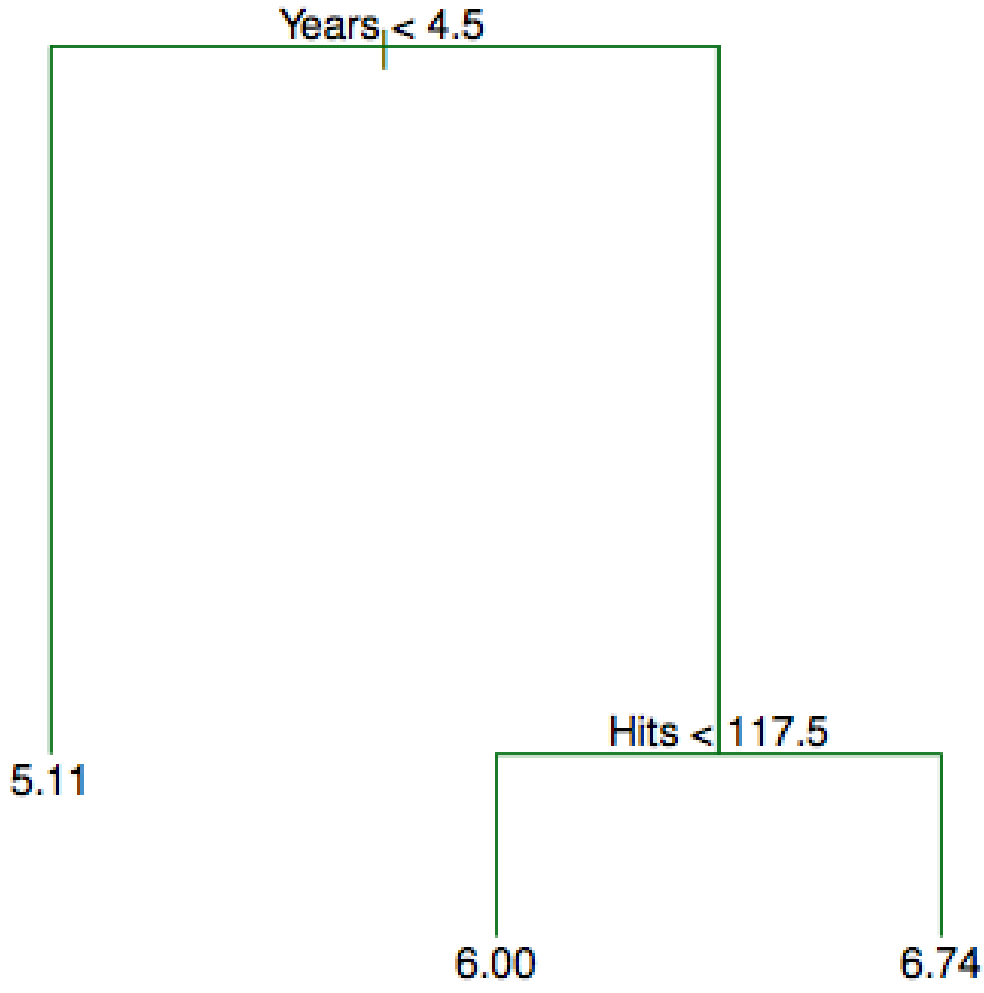
- Les arbres de décision sont utiles aussi bien pour des problèmes de régression que de classification.
- Nous commençons par présenter des problèmes de régression et nous viendrons ensuite à la classification.

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

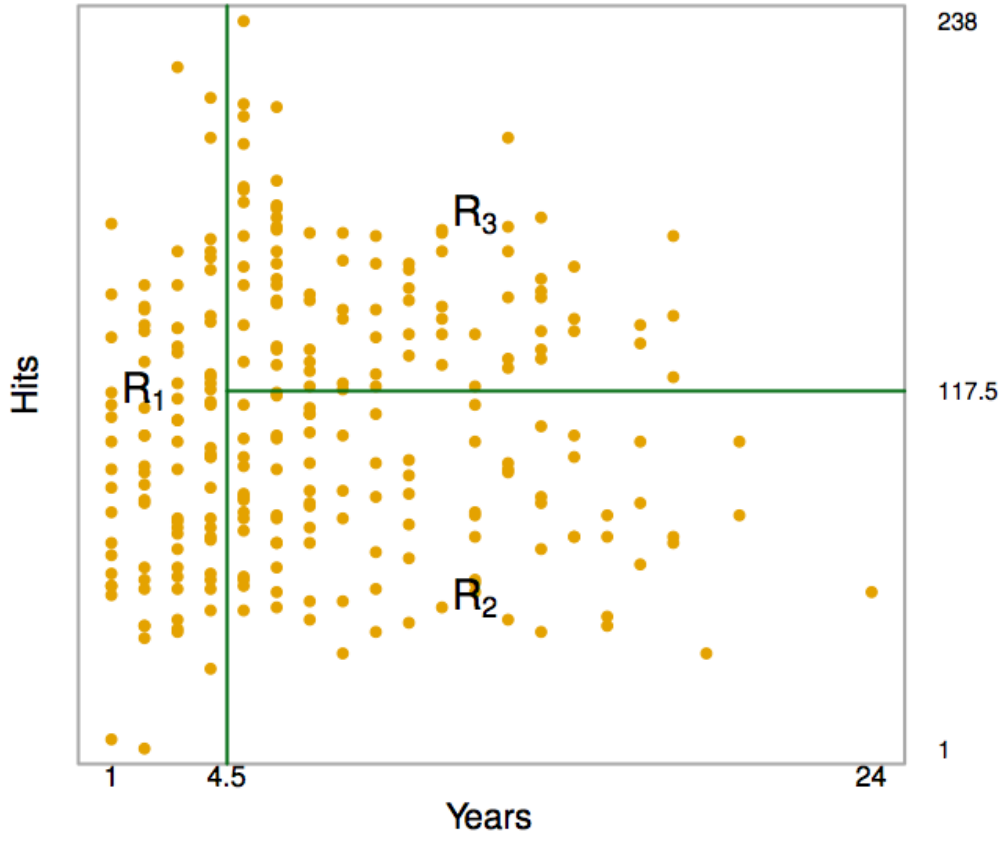


Détails de la précédente figure

- 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'entraînement 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:

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

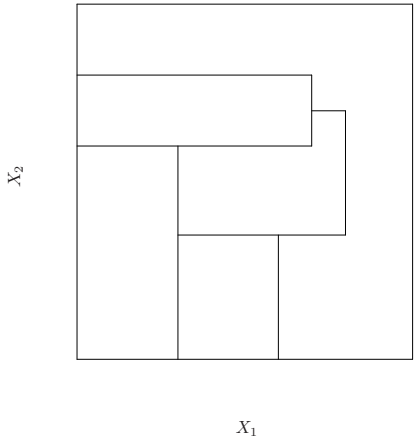
en cherchant j et s qui minimisent

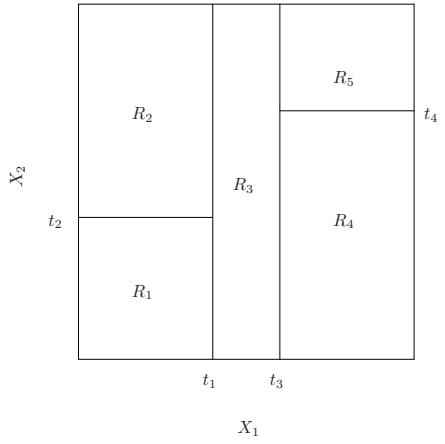
$$\text{RSS}_{\text{new}} = \sum_{i:x_i \in \mathcal{R}_1(j,s)} \left(y_i - \hat{y}_1\right)^2 + \sum_{i:x_i \in \mathcal{R}_2(j,s)} \left(y_i - \hat{y}_2\right)^2 \tag{1}$$

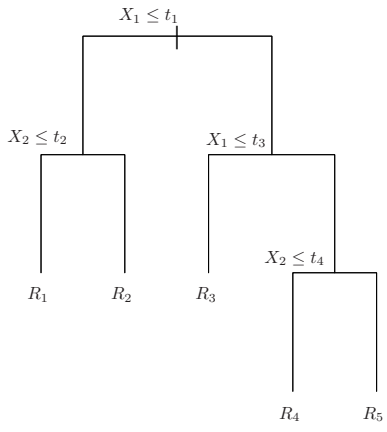
où $\hat{y}_m = \text{ave}\left(y_i \middle| x_i \in \mathcal{R}_m(j,s)\right)$ est la réponse moyenne des données d'apprentissage qui tombent dans la région $R_m(j,s)$ pour $m = 1$ ou $m = 2$.

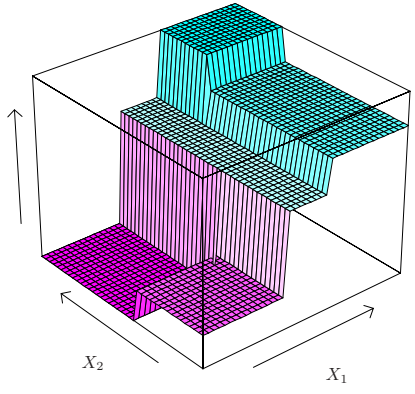
Trouver le couple (j,s) optimal est un problème relativement facile lorsque le nombre de variables p n'est pas trop grand.

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

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

où

- $|T|$ est nombre de feuilles de T
- $N_m(T) = \text{Card}\left\{x_i \in \mathcal{R}_m(T)\right\}$ et
  $Q_m(T) = \frac{1}{N_m(T)} \sum_{x_i \in \mathcal{R}_m(T)} \left(y_i - \hat{y}_m\right)^2$
- $\hat{y}_m = \text{ave}\left(y_i \middle| x_i \in \mathcal{R}_m(T)\right)$
- 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

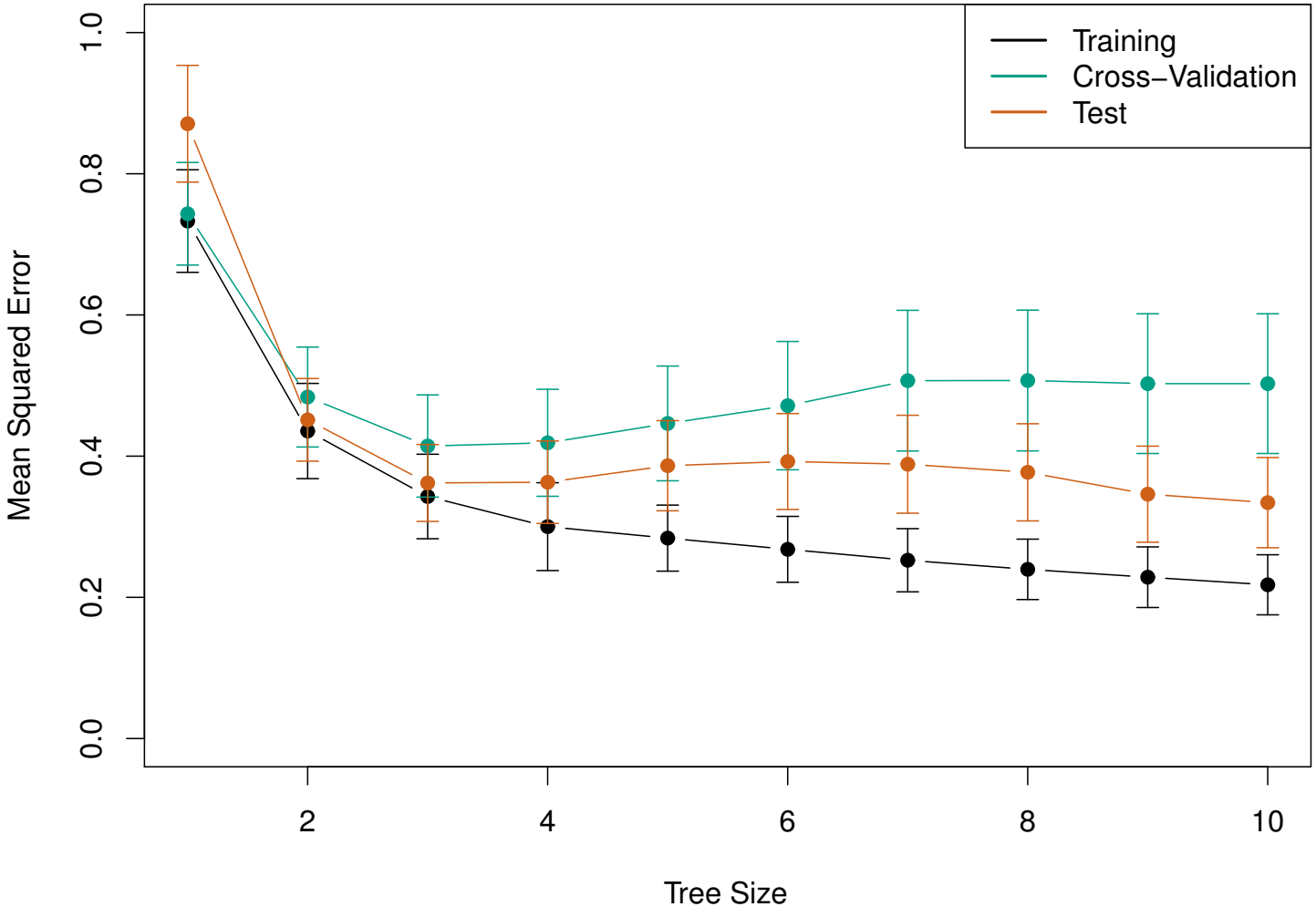
$$\sum_m N_m(T) Q_m(T)$$

   - 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 charges 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'entraînement

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

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

  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 $\hat{c}_m = \text{argmax}_k \, \hat{p}_{mk}$ (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** :

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

- **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

Pima dataset

```
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 ajuster un arbre de décision profond visant à prédire le diabète en fonction des autres variables présentes dans le jeu de données. Calculer l'erreur de test;
- Déduire l'arbre élagué. Calculer son erreur de test.

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.

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'entraînement (pas assez de données observées)

Bagging pour la régression

- Au lieu de cela, on peut bootstrapper en ré-échantillonnant plusieurs fois les données d'entraînement.
- Alors, à partir de B échantillons bootstrap, on entraîne 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 règles de classification 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'entraînement, 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'entraînement 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 entraînement
  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 $\hat{f}^*(x_i)$.

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

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

- 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'agrégation des fonctions de régression $\hat{m}_1, \ldots, \hat{m}_B$. Remarquons que si on suppose que les fonctions de régression $\hat{m}_1, \ldots, \hat{m}_B$ 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 $\theta_b = \theta_b\big(\mathcal{D}_n\big)$ l'échantillon bootstrap de l'étape b et $\hat{m}(\cdot, \theta_b)$ 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

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

- 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
  a. $\theta_b(\mathcal{D}_n)$ est obtenu en tirant n observations avec remise dans $\mathcal{D}_n$, chaque observation ayant la même probabilité d'être tirée $\frac{1}{n}$.
  b. $\theta_b(\mathcal{D}_n)$ 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

- $\sigma^2(x) = \mathrm{Var}\big(\hat{m}(x, \theta_b)\big)$
- $\rho(x) = \mathrm{corr}\Big(\hat{m}(x, \theta_1), \hat{m}(x, \theta_2)\Big)$, 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

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

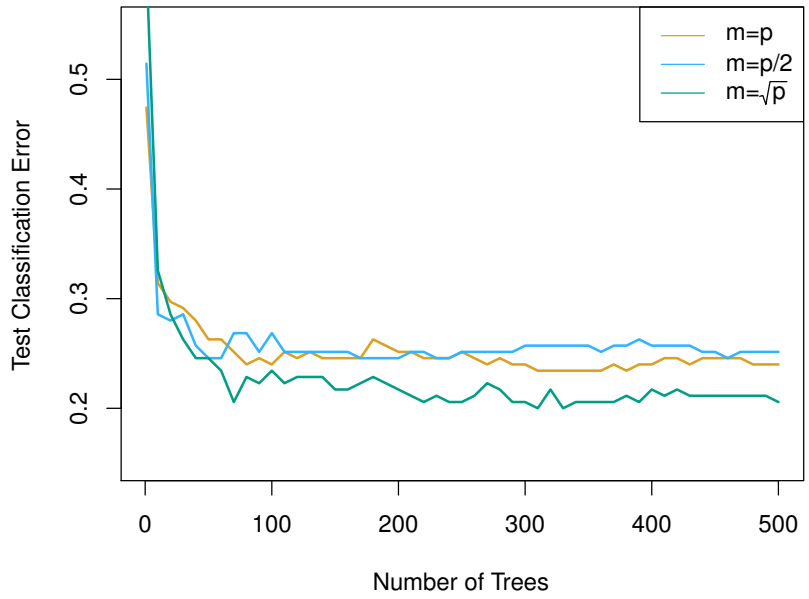
Forêts aléatoires très proche du bagging

- 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%)

Agrégation séquentielle : boosting

- De quoi s'agît-il ?
- Un peu d'histoire
- Gradient boosting pour la régression

De quoi s'agît-il ?

Gradient Boosting = Gradient Descent + Boosting

De quoi s'agît-il ?

- Premier algorithme de "boosting" [Freund and Schapire, 1997].
- Contruire une famille de règles qui sont ensuite agrégées.
- Processus récursif : la règle construite à l'étape k dépend de celle construite à l'étape $k-1$

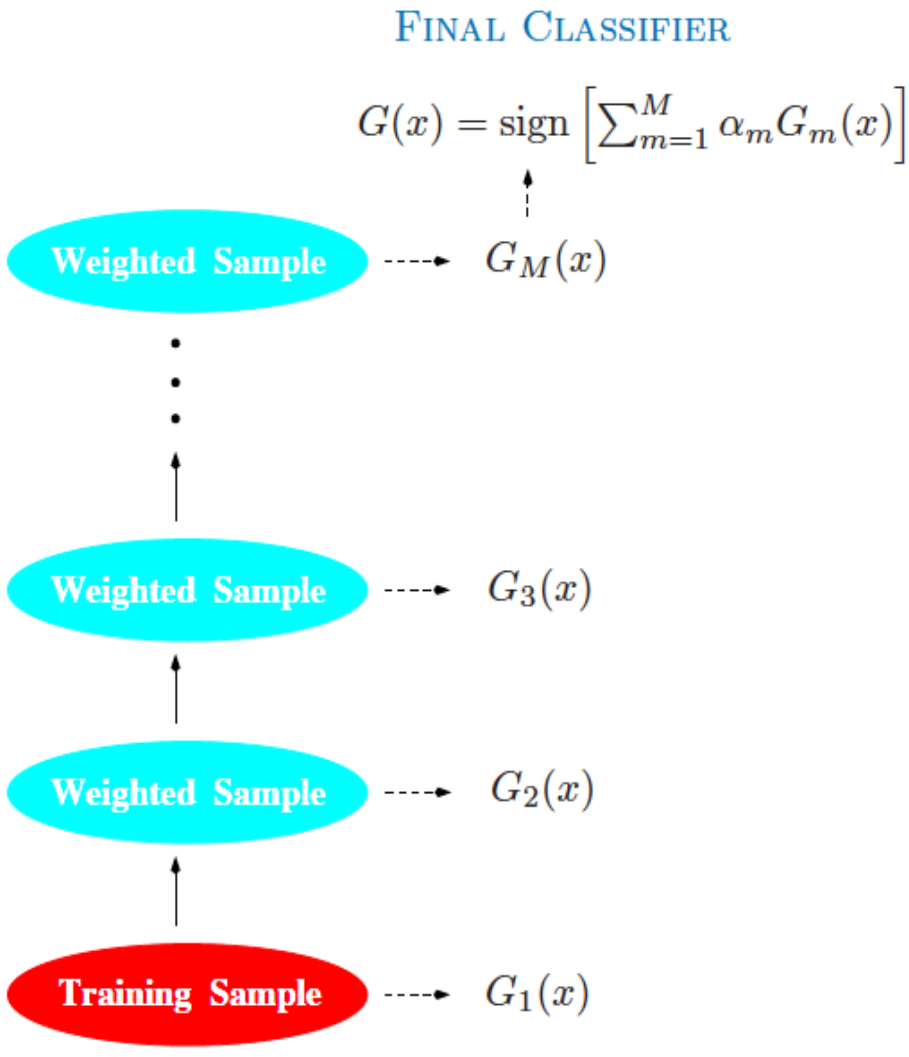
Un peu d'histoire

- Invention Adaboost, premier algorithme de boosting [Freund et al., 1996, Freund and Schapire, 1997]
- Formulation de l'algorithme Adaboost comme une descente du gradient avec une fonction de perte particulière [Breiman et al., 1998, Breiman, 1999]
- Généralisation de l'algorithme Adaboost au Gradient Boosting pour l'adapter à différentes fonctions de perte [Friedman et al., 2000, Friedman, 2001]

Principe

- Le bagging propose d'agréger des modèles à forte variances.
- Le boosting est proposé à l'origine pour des problèmes de classification ensuite adapté à la régression.
- Le boosting combine séquentiellement des règles de classification dites faibles pour produire une règle de classification précise.
- Nous allons introduire l'algorithme de boosting le plus connu appelé AdaBoost.M1 introduit par [Freund and Schapire, 1997].
- On s'intéresse au problème de classification binaire où $Y \in \{-1, 1\}$. Pour un vecteur de variables explicatives, $g(X)$ est une règle de classification qui prédit une des modalités $\{-1, 1\}$.

Schéma (Hastie et al. 2009)



Notion de règle faible

- Le terme boosting s'applique à des méthodes générales permettant de produire des décisions précises à partir de règles faibles.

Définition : *On appelle règle de classification faible une règle légèrement meilleure que le hasard:*

$$g \quad \textit{faible si} \quad \exists \gamma > 0 \quad \textit{tel que} \quad \mathbb{P}\big(g(X) \neq Y\big) = \frac{1}{2} - \gamma.$$

- **Exemple :** arbre à 2 feuilles.

Schéma ou idée

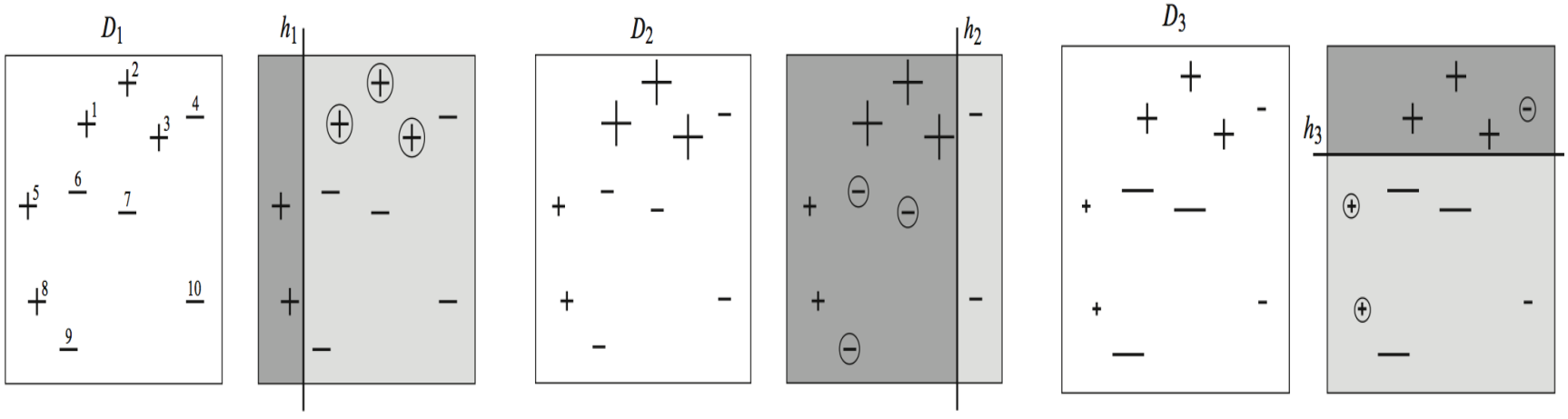


Figure: AdaBoost. Source: Figure 1.1 of [Schapire and Freund, 2012]

Algorithme dit Adaboost.M1

Input : - Une observation x à prédire et l'échantillon $d_n = (x_1, y_1), \ldots, (x_n, y_n)$ - Une règle de classification faible et M le nombre d'itérations

Algorithm of [Freund and Schapire 1997]:

1. Initialisser les poids $w_i = \frac{1}{n}, i = 1, \ldots, n$
2. **Pour** $m = 1$ à M:
   a. Ajuster la règle faible sur l'échantillon d_n pondéré par les poids $w_1, \ldots, w_n$, on note $g_m(x)$ l'estimateur issu de cet ajustement
   b. Calcul du taux d'erreur :

$$err_m = \frac{\sum_{i=1}^{n} w_i 1_{y_i \neq g_m(x_i)}}{\sum_{i=1}^{n} w_i}.$$

   c. Calcul de : $\alpha_m = \log\left(\frac{1-err_m}{err_m}\right)$
   d. Réajuster les poids + normalisation

$$w_i = w_i \exp\left(\alpha_m 1_{y_i \neq g_m(x_i)}\right), \quad i = 1, \ldots, n.$$

Output:

$$\widehat{g}_M(x) = \sum_{m=1}^{M} \alpha_m g_m(x).$$

Schéma ou idée

$$\widehat{H}_3(x) = \sum_{m=1}^{3} \alpha_m h_m(x)$$

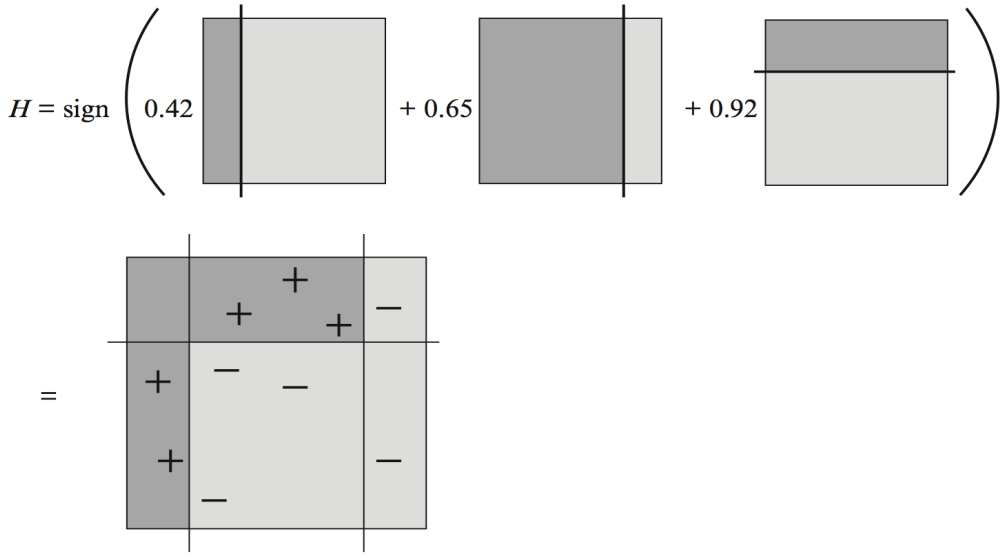


Figure: AdaBoost. Source: Figure 1.2 of [Schapire and Freund, 2012]

Cet algorithme a été introduit en 1996 par Yoav Freund and Rob Shapire (prix Gödel 2003)

Commentaires

- L'étape 1. nécessite que la règle faible puisse prendre en compte des poids. Lorsque ce n'est pas le cas, la règle peut être ajustée sur un sous-échantillon de d_n dans lequel les observations sont tirées avec remise selon les poids $w_1, \ldots, w_n$.
- Les poids $w_1, \ldots, w_n$ sont mis à jour à chaque itération : si le $i^{\text{ième}}$ individu est bien classé son poids est inchangé, sinon il est augmenté.
- Le poids α_m de la règle g_m augmente avec la performance de g_m mesurée sur d_n : α_m augmente lorsque e_m diminue (il faut néanmoins que g_m ne soit pas trop faible : si $e_m > 0.5$ alors $\alpha_m < 0$!!!).

Erreur empirique d'apprentissage

- err_m désigne le taux d'erreur calculé sur l'échantillon de la règle g_m:

$$err_m = \frac{\sum_{i=1}^{n} w_i 1_{y_i \neq g_m(x_i)}}{\sum_{i=1}^{n} w_i}.$$

- γ_m désigne le gain de la règle g_m par rapport à une règle **pûrement aléatoire**

$$err_m = \frac{1}{2} - \gamma_m.$$

Propriété: [Freund and Schapire, 1999]

$$L_n\left(\widehat{g}_M\right) \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right).$$

Conséquence :
L'erreur empirique (calculée sur les données) tend vers 0 lorsque le nombre d'itérations augmente.

Erreur empirique d'apprentissage (suite)

Ils ont montré que

$$L_n\left(\widehat{g}_M\right) = \frac{1}{n}\sum_{i=1}^{n} 1_{y_i \neq \widehat{g}_M(x_i)} \leq \exp\left(-2\sum_{m=1}^{M}\gamma_m^2\right) \leq \exp\left(-2M\gamma^2\right)$$

Erreur de généralisation

Définition : C'est l'erreur moyenne attendue sur un échantillon test

$$L\left(\widehat{g}_M\right) = \mathbb{P}\left[Y \neq \widehat{g}_M(X)\right]$$

Borne obtenue par Freund & Schapire

$$L\left(\widehat{g}_M\right) \leq L_n\left(\widehat{g}_M\right) + \mathcal{O}\left(\sqrt{\frac{MV}{n}}\right)$$

où V est la dimension de Vapnik-Chervonenkis de la famille de règles de classification faibles (3 dans l'exemple simple).

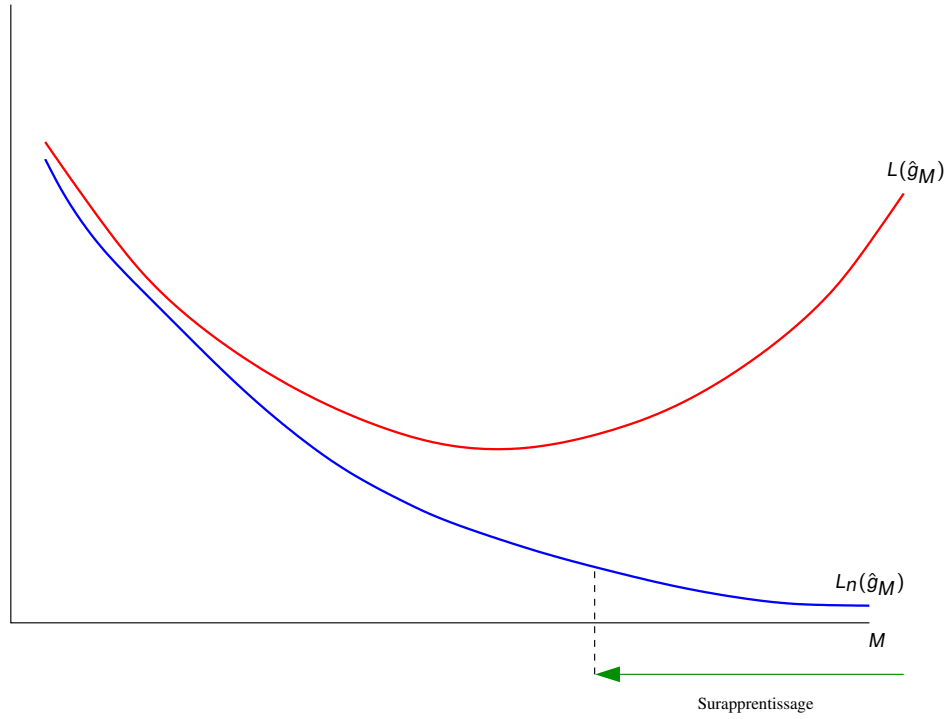
Erreur de généralisation (suite)

Interprétation Il peut y avoir du sur-ajustement

- Le compromis biais/variance ou erreur approximation/estimation est régulé par le nombre d'itérations M :
  1. M petit $\rightarrow$ premier terme (approximation) domine
  2. M grand $\rightarrow$ second terme (estimation) domine
- Lorsque M est (trop) grand, Adaboost aura tendance à sur-ajuster l'échantillon d'apprentissage (sur-ajustement ou overfitting).

Sur-apprentissage: Qu'est-ce que c'est ?

C'est ce qui se passe quand en complexifiant le modèle l'erreur d'apprentissage baisse, alors que l'erreur de généralisation se remet à augmenter.

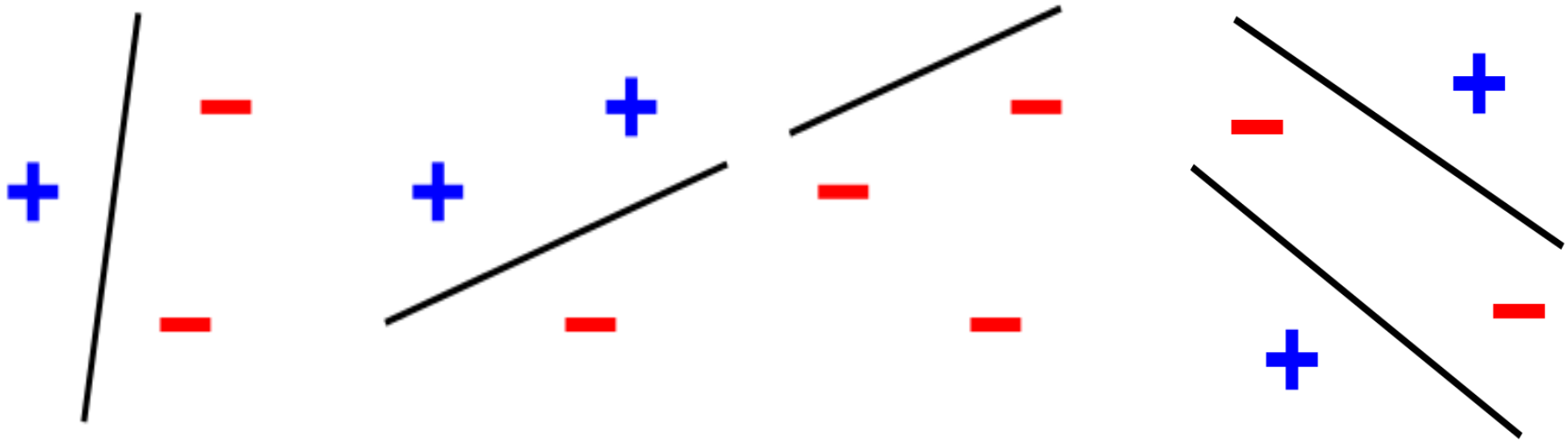


Dimension de Vapnik-Chervonenkis : Qu'est-ce que c'est ?

C'est une mesure de la capacité d'un algorithme de classification statistique

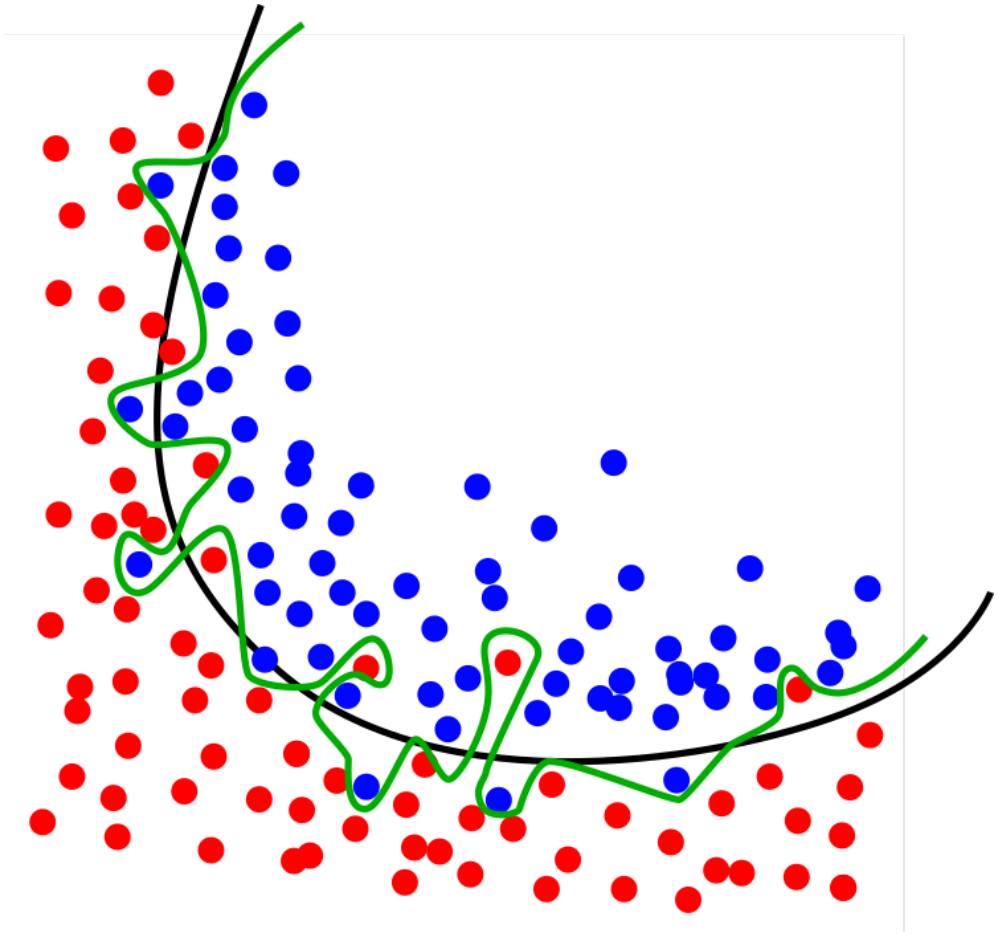
- cardinal du plus grand ensemble de points que l'algorithme peut **pulvériser**
- **Pulvériser:** un modèle de classification g_θ pulvérise un ensemble de données $E = (x_1, x_2, \ldots, x_n)$ si, pour tout étiquetage E, il existe θ tel que g_θ ne fasse aucune erreur dans l'évaluation de cet ensemble de données.
- Une droite en dimension 2 : on peut pulvériser 3 points mais pas 4 points!

Dimension de Vapnik-Chervonenkis



Dimension de Vapnik-Chervonenkis (suite)

Un modèle de dimension VC trop haute risque le sur-apprentissage par un modèle complexe trop adapté aux données d'apprentissage



Gradient boosting pour la régression (intuitif)

- Nous disposons de $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, et une fonction $\hat{f}$ qui minimise l'erreur quadratique moyenne.
- On fait une petit vérification et constate quelques écarts à la vérité : $\hat{f}(x_1) = 0.8$ alors que $y_1 = 0.9$, et $\hat{f}(x_2) = 1.4$ et $y_2 = 1.3, \ldots$. Comment améliorer $\hat{f}$?
- On ne peut pas modifier $\hat{f}$.
- On peut ajouter un modèle (arbre de régression) h à $\hat{f}$ et la prédiction sera donnée par $\hat{f}(x) + h(x)$.

Gradient boosting pour la régression

Solution simple

$$\hat{f}(x_1) + h(x_1) = y_1$$
$$\hat{f}(x_2) + h(x_2) = y_2$$
$$\hat{f}(x_3) + h(x_3) = y_3$$
$$\dots$$
$$\hat{f}(x_n) + h(x_n) = y_n$$

Gradient boosting pour la régression

Peut-on obtenir un arbre h tel que

$$
\begin{aligned}
h(x_1) &= y_1 - \hat{f}(x_1) \\
h(x_2) &= y_2 - \hat{f}(x_2) \\
h(x_3) &= y_3 - \hat{f}(x_3) \\
&\dots \\
h(x_n) &= y_n - \hat{f}(x_n)
\end{aligned}
$$

Oui mais une approximation !

Gradient boosting pour la régression

Peut-on obtenir un arbre h tel que

$$
\begin{aligned}
h(x_1) &= y_1 - \hat{f}(x_1) \\
h(x_2) &= y_2 - \hat{f}(x_2) \\
h(x_3) &= y_3 - \hat{f}(x_3) \\
&\dots \\
h(x_n) &= y_n - \hat{f}(x_n)
\end{aligned}
$$

Oui mais une approximation !

$$
\big(x_1, y_1 - \hat{f}(x_1)\big), \big(x_2, y_2 - \hat{f}(x_2)\big), \dots, \big(x_n, y_n - \hat{f}(x_n)\big)
$$

Gradient boosting pour la régression

Une solution simple

- $y_i - \hat{f}(x_i)$ sont les résidus. La partie qui échappe à $\hat{f}$.
- Le rôle de h est de compenser les lacunes de $\hat{f}$.
- Si la nouvelle fonction de régression estimée $\hat{f} + h$ demeure insatisfaisante, on peut ajouter d'autres arbres de régression.

Modélisation additive linéaire

Modélisation additive linéaire

Contexte: Classification ou régression (presque le même que pour AdaBoost)

- On a toujours une variable $y \in \{-1, 1\}$ ou $y \in \mathbb{R}$ à inférer à partir de règles faibles.
- Cette fois-ci, on se donne un fonction de coût (ou déviance) $L(y, g)$ que l'on cherche à minimiser.

Approche: On modélise à chaque fois le résidu produit par la solution précédente, on a donc

$$\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x) = \widehat{g}_{M-1}(x) + \beta_M g_M(x)$$

Algorithme Forward staging additive modeling

Entrée: Les éléments nécessaires sont

- un jeu de données $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$
- un ensemble de règles faibles
- le nombre M d'itérations

Initialisation: $\widehat{g}_0(x)$

Itération: pour $m = 1$ à M

1. ajuster la règle faible g_m et calculer un coefficient β_m qui minimise
$$\sum_{i=1}^{n} L\big(y_i, \widehat{g}_{m-1}(x_i) + \beta_m g_m(x_i)\big)$$
2. $$\widehat{g}_m(x) = \widehat{g}_{m-1}(x) + \beta_m g_m(x)\big)$$

Sortie: La prédiction est $\operatorname{sign}\widehat{g}_M(x)$ (en classification)

Justification du boosting : minimisation de risque empirique

Pertes théorique et empirique

- (X, Y) couple aléatoire à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$. Étant donnée $\mathcal{G}$ une famille de règles, on se pose la question de trouver la meilleure règle dans $\mathcal{G}$.
- Choisir la règle qui minimise une fonction de perte, par exemple

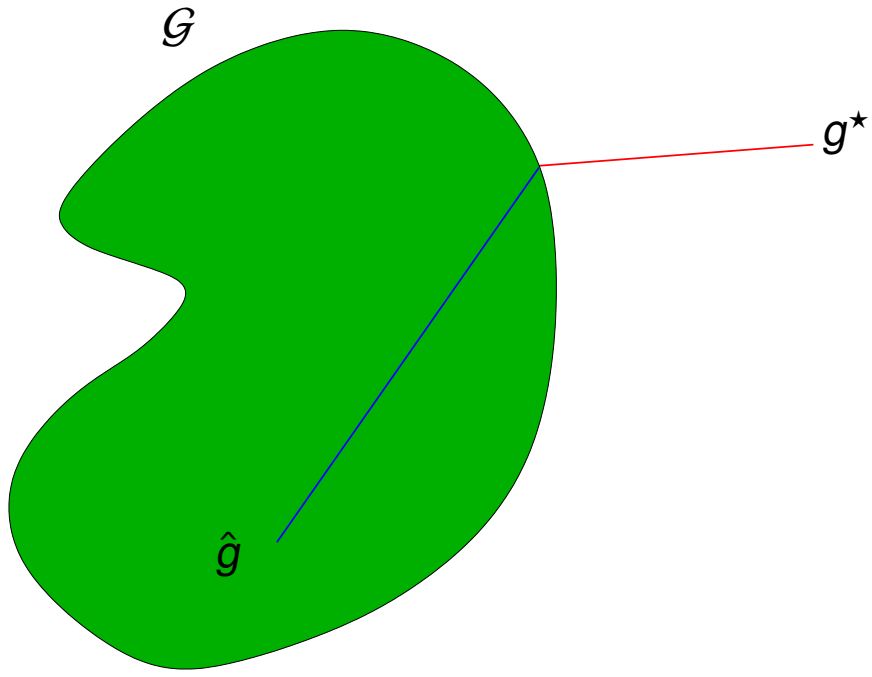
$$L(g) = \mathbb{P}\big(Y \neq g(X)\big).$$

Problème : la fonction de perte n'est pas calculable

- Idée : choisir la règle qui minimise la version empirique de la fonction de perte :

$$L_n(g) = \frac{1}{n} \sum_{i=1}^{n} 1_{g(X_i) \neq Y_i}.$$

Erreurs d'estimation et d'approximation



$$L(\hat{g}) - L^* = L(\hat{g}) - \inf_{g \in \mathcal{G}} L(g) + \inf_{g \in \mathcal{G}} L(g) - L^*.$$

Risque convexifié

Problème : la fonction

$$\begin{aligned} \mathcal{G} &\to \mathbb{R} \\ g &\mapsto \frac{1}{n}\sum_{i=1}^{n} 1_{g(X_i)\neq Y_i} \end{aligned}$$

est généralement difficile à minimiser.
Idée : trouver une autre fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ telle que

$$\begin{aligned} \mathcal{G} &\to \mathbb{R} \\ g &\mapsto \frac{1}{n}\sum_{i=1}^{n} \ell\big(Y_i, g(X_i)\big) \end{aligned}$$

soit "facile" à minimiser (si la fonction fonction $v \mapsto \ell(u, v)$ est convexe par exemple).

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.
- Elle doit donc prendre des valeurs
  - élevées lorsque $yg(x) < 0$
  - faibles lorsque $yg(x) > 0$

Fonction de perte

- La fonction de perte $\ell(y, g(x))$ mesure l'écart entre la quantité à prévoir $y \in \{-1, 1\}$ et $g(x)$.
- Elle doit donc prendre des valeurs
  - élevées lorsque $yg(x) < 0$
  - faibles lorsque $yg(x) > 0$
- Exemple:
  1. $\ell(y, g(x)) = 1_{yg(x)<0}$
  2. $\ell(y, g(x)) = \exp\big(-yg(x)\big)$ (présente l'avantage d'être convexe en le second argument).

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \operatorname{argmin} \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \operatorname{argmin} \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $\big(X_1, Y_1\big), \ldots, \big(X_n, Y_n\big)$ de même loi que $\big(X, Y\big)$, on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

$$\frac{1}{n} \sum_{i=1}^{n} \ell\big(Y_i, g(X_i)\big).$$

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \text{argmin}\, \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $\big(X_1, Y_1\big), \ldots, \big(X_n, Y_n\big)$ de même loi que $\big(X, Y\big)$, on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

$$\frac{1}{n}\sum_{i=1}^{n} \ell\big(Y_i, g(X_i)\big).$$

- Approche récursive : approcher g^* par $\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x)$ où g_m et β_m sont construits de façon récursive.

Récapitulatif

- (X, Y) à valeurs dans $\mathbb{R}^p \times \{-1, 1\}$, une fonction de perte $\ell : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ et on cherche à approcher

$$g^* = \text{argmin}\, \mathbb{E}\left[\ell\big(Y, g(X)\big)\right].$$

- Stratégie : étant donnée un n échantillon i.i.d $\big(X_1, Y_1\big), \ldots, \big(X_n, Y_n\big)$ de même loi que $\big(X, Y\big)$, on cherche à minimiser la version empirique de $\mathbb{E}\left[\ell\big(Y, g(X)\big)\right]$:

$$\frac{1}{n}\sum_{i=1}^{n} \ell\big(Y_i, g(X_i)\big).$$

- Approche récursive : approcher g^* par $\widehat{g}_M(x) = \sum_{m=1}^{M} \beta_m g_m(x)$ où g_m et β_m sont construits de façon récursive.
- Méthode : utiliser une approche numérique (descente de gradients, Newton-Raphson).

Fonctions de coût pour la classification

Exponentielle

$$L(y, g) = \exp(-yg)$$

- On peut prouver qu'on retrouve AdaBoost !!
- Pourtant l'idée est très différente

Logistique $L(y, g) = \log\left(1 + \exp(-2yg)\right)$

- Similaire à AdaBoost a priori
- Moins sensible aux observations mal classées

Fonctions de coût pour la régression

Quadratique

$$\ell(y, g) = \frac{1}{2}(y - g)^2$$

- sensible aux valeurs aberrantes ou extrêmes

Absolue

$$\ell(y, g) = |y - g|$$

- Plus robuste, mais moins précis pour les petites erreurs

Huber

$$\ell(y, g) = (y - g)^2 1_{|y-g| \leq \delta} + \left(2\delta|y - g| - \delta^2\right) 1_{|y-g| > \delta}$$

- combine les bonnes propriétés des deux fonctions précédentes

Un petit rappel

Nous faisons ici un bref rappel sur la méthode de Newton-raphson dans le cas simple de la minimisation d'une fonction strictement convexe $J : \mathbb{R} \to \mathbb{R}$. Si on désigne par $\tilde{x}$ la solution du problème de minimisation, la méthode consiste à construire une suite (x_k) qui converge vers $\tilde{x}$. La suite est tout d'abord initialisée en choisissant une valeur x_0. On cherche alors $x_1 = x_0 + h$ tel que $J'(x_1) \approx 0$. Par un développement limité, on obtient l'approximation

$$J'(x_0 + h) \approx J'(x_0) + hJ''(x_0).$$

Comme $J'(x_0 + h) \approx 0$, il vient $h = -\big(J''(x_0)\big)^{-1} J'(x_0)$. Si on pose $\lambda = \big(J''(x_0)\big)^{-1}$, alors $x_1 = x_0 - \lambda J'(x_0)$ et on déduit la formule de récurrence

$$x_k = x_{k-1} - \lambda J'(x_{k-1}).$$

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n} \sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

- La formule de récurrence de l'algorithme de Newton-Raphson est donnée par

$$\boldsymbol{g_m} = \boldsymbol{g_{m-1}} - \lambda \nabla J\big(\boldsymbol{g_{m-1}}\big),$$

où $\lambda > 0$ désigne le pas de descente de gradient.

Newton Raphson

- On note $\boldsymbol{g_m} = \big(g_m(x_1), \ldots, g_m(x_n)\big)$, et

$$J\big(\boldsymbol{g_m}\big) = \frac{1}{n}\sum_{i=1}^{n} \ell\big(y_i, g_m(x_i)\big).$$

- La formule de récurrence de l'algorithme de Newton-Raphson est donnée par

$$\boldsymbol{g_m} = \boldsymbol{g_{m-1}} - \lambda \nabla J\big(\boldsymbol{g_{m-1}}\big),$$

où $\lambda > 0$ désigne le pas de descente de gradient.

- **Inconvénients**
  a. Cet algorithme permet de calculer l'estimateur uniquement en les points du design $x_1, \ldots, x_n$.
  b. Ne prend pas en compte une éventuelle régularité de la fonction à estimer (si x_i est proche de x_j alors $g^*(x_i)$ est proche de $g^*(x_j)$).

Boosting par descente du gradient

Entrées :

- $(x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $\lambda > 0$ et M le nombre d'itérations.

a. Initialisation : $\widehat{g}_0(\cdot) = \text{argmin}_c \sum_{i=1}^{n} \ell(y_i, c)$

b. **Pour** $m = 1$ à M :

1..1 Calculer l'opposé du gradient et l'évaluer aux points d'observation

$$r_{im} = -\frac{\partial}{\partial g(x_i)} \ell\big(y_i, g_m(x_i)\big)\Big|_{y=y_i, g(x_i)=\widehat{g}_{m-\mathbf{1}}(x_i)}, \quad i = 1, \ldots, n.$$

2..2 ajuster une règle faible g_m sur l'échantillon $\big(x_1, r_{1m}\big), \ldots, \big(x_n, r_{nm}\big)$

3..3 Mise à jour : $\widehat{g}_m(x) = \widehat{g}_{m-1}(x) + \lambda g_m(x)$.

c. **Sortie** : La règle $\widehat{g}_M(x)$ pour la régression et $\text{sign}\widehat{g}_M(x)$ pour la classification.

Boosting par descente du gradient avec des arbres

- Notation formelle d'un arbre

$$T(x, \Theta) = \sum_{j=1}^{J} \gamma_j 1(x \in R_j)$$

où $\Theta = \{R_j, \gamma_j\}_1^J$

- Un arbre boosté donnera

$$\widehat{f}_M(x) = \sum_{m=1}^{M} T(x, \Theta_m)$$

Boosting par descente du gradient avec des arbres

Entrées :

- $(x_1, y_1), \ldots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $\lambda > 0$ et M le nombre d'itérations.

a. Initialisation : $f_0(x) = \text{argmin}_\gamma \sum_{i=1}^{n} \ell(y_i, \gamma)$

b. **Pour** $m = 1$ à M :

1..1 Calculer l'opposé du gradient et l'évaluer aux points d'observation

$$r_{im} = -\frac{\partial}{\partial g(x_i)} \ell\big(y_i, f_m(x_i)\big)\Big|_{y=y_i, f(x_i)=f_{m-1}(x_i)}, \quad i = 1, \ldots, n.$$

2..2 ajuster un arbre sur l'échantillon $\big(x_1, r_{1m}\big), \ldots, \big(x_n, r_{nm}\big)$ qui donne les feuilles $R_{jm}, j = 1, \ldots J_m$.

3..3 Pour $j = 1, \ldots, J_m$ calculer

$$\gamma_{jm} = \text{argmin}_\gamma \sum_{x_i \in R_{jm}} \ell\big(y_i, f_{m-1}(x_i) + \gamma\big)$$

4..4 Mise à jour : $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} 1(x \in R_{jm})$.

c. $\widehat{f}(x) = f_M(x)$