

Machine learning

L. Rouvière

laurent.rouviere@univ-rennes2.fr

Novembre 2019

Présentation

- **Objectifs** : comprendre les aspects **théoriques** et **pratiques** de quelques algorithmes machine learning.
- **Pré-requis** : théorie des probabilités, modélisation statistique, régression (linéaire et logistique). R, niveau avancé.
- **Enseignant** : Laurent Rouvière laurent.rouviere@univ-rennes2.fr
 - **Recherche** : statistique non paramétrique, apprentissage statistique
 - **Enseignements** : statistique et probabilités (Université, école d'ingénieur et de commerce, formation continue).
 - **Consulting** : énergie, finance, marketing.

- **Matériel** : slides + Notebook R. Disponible à l'url :
https://lrouviere.github.io/ml_lecture/
- **4 parties** :
 1. Contexte mathématique pour l'apprentissage (rappels ?) 2h00.
 2. Support vector machine : 3h.
 3. Agrégation : forêts aléatoires et boosting+arbres (rappels ?) : 5h.
 4. Réseau de neurones et introduction au deep learning : 2h.

Première partie I

Apprentissage : contexte et formalisation

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

Apprentissage statistique ?

Plusieurs "définitions"

1. "... explores way of **estimating functional dependency** from a given collection of data" [Vapnik, 2000].
2. "...vast set of tools for modelling and understanding **complex data**" [James et al., 2015]

Apprentissage statistique ?

Plusieurs "définitions"

1. "... explores way of **estimating functional dependency** from a given collection of data" [Vapnik, 2000].
2. "...vast set of tools for modelling and understanding **complex data**" [James et al., 2015]
3. Comprendre et apprendre un comportement à partir d'**exemples**.

Apprentissage statistique ?

Plusieurs "définitions"

1. "... explores way of **estimating functional dependency** from a given collection of data" [Vapnik, 2000].
2. "...vast set of tools for modelling and understanding **complex data**" [James et al., 2015]
3. Comprendre et apprendre un comportement à partir d'**exemples**.

Constat

- Le **développement des moyens informatiques** fait que l'on est confronté à des données de plus en plus **complexes**.
- Les méthodes **traditionnelles** se révèlent souvent **peu efficaces** face à ce type de données.
- Nécessité de proposer des **algorithmes/modèles statistiques** qui apprennent directement à partir des données.

Un peu d'histoire - voir [Besse and Laurent,]

Période	Mémoire	Ordre de grandeur
1940-70	Octet	$n = 30, p \leq 10$
1970	kO	$n = 500, p \leq 10$
1980	MO	Machine Learning
1990	GO	Data-Mining
2000	TO	$p > n$, apprentissage statistique
2010	PO	n explose, cloud, cluster...
2013	??	Big data
2017	??	Intelligence artificielle...

Un peu d'histoire - voir [Besse and Laurent,]

Période	Mémoire	Ordre de grandeur
1940-70	Octet	$n = 30, p \leq 10$
1970	kO	$n = 500, p \leq 10$
1980	MO	Machine Learning
1990	GO	Data-Mining
2000	TO	$p > n$, apprentissage statistique
2010	PO	n explose, cloud, cluster...
2013	??	Big data
2017	??	Intelligence artificielle...

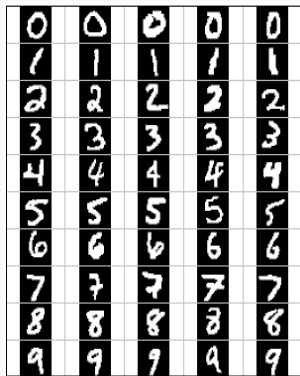
Conclusion

Moyens informatiques \implies Data Mining \implies Apprentissage statistique
 \implies Intelligence artificielle...

Reconnaissance de l'écriture

Apprentissage statistique

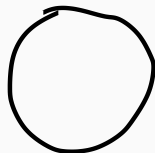
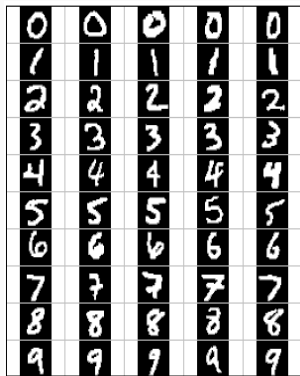
Comprendre et apprendre un comportement à partir d'exemples.



Reconnaissance de l'écriture

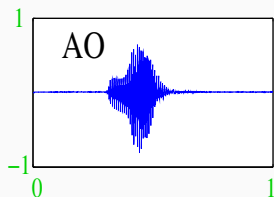
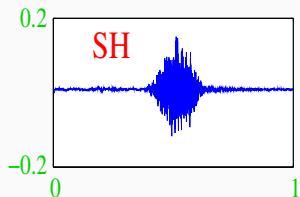
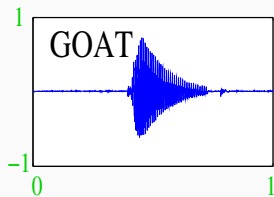
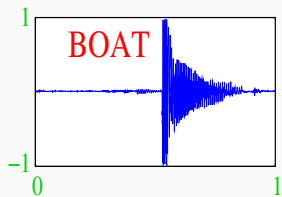
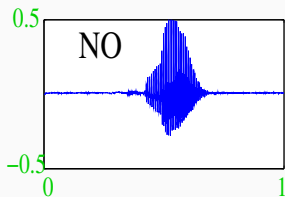
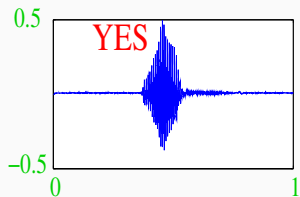
Apprentissage statistique

Comprendre et apprendre un comportement à partir d'exemples.

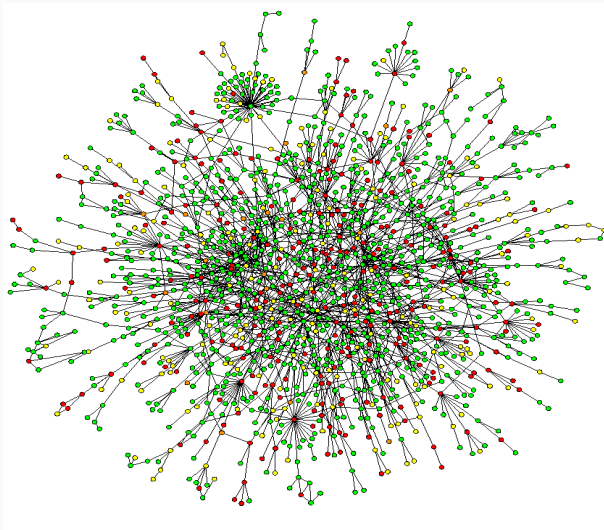


Qu'est-ce qui est écrit ? 0, 1, 2... ?

Reconnaissance de la parole



Apprentissage sur les réseaux



Prévision de pics d'ozone

- On a mesuré pendant 366 jours la **concentration maximale** en ozone (V4) ;
- On dispose également d'autres variables météorologiques (température, nébulosité, vent...).

```
> head(Ozone)
```

##	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
## 1	1	1	4	3	5480	8	20	NA	NA	5000	-15	30.56	200
## 2	1	2	5	3	5660	6	NA	38	NA	NA	-14	NA	300
## 3	1	3	6	3	5710	4	28	40	NA	2693	-25	47.66	250
## 4	1	4	7	5	5700	3	37	45	NA	590	-24	55.04	100
## 5	1	5	1	5	5760	3	51	54	45.32	1450	25	57.02	60
## 6	1	6	2	6	5720	4	69	35	49.64	1568	15	53.78	60

Prévision de pics d'ozone

- On a mesuré pendant 366 jours la **concentration maximale** en ozone (V4) ;
- On dispose également d'autres variables météorologiques (température, nébulosité, vent...).

```
> head(Ozone)
```

##	V1	V2	V3	V4	V5	V6	V7	V8	V9	V10	V11	V12	V13
## 1	1	1	4	3	5480	8	20	NA	NA	5000	-15	30.56	200
## 2	1	2	5	3	5660	6	NA	38	NA	NA	-14	NA	300
## 3	1	3	6	3	5710	4	28	40	NA	2693	-25	47.66	250
## 4	1	4	7	5	5700	3	37	45	NA	590	-24	55.04	100
## 5	1	5	1	5	5760	3	51	54	45.32	1450	25	57.02	60
## 6	1	6	2	6	5720	4	69	35	49.64	1568	15	53.78	60

Question

Peut-on **prédire** la concentration maximale en ozone du **lendemain** à partir des prévisions météorologiques ?

Détection de spam

- Sur 4 601 mails, on a pu identifier **1813 spams**.
- On a également mesuré sur chacun de ces mails la présence ou absence de **57 mots**.

```
> spam %>% select(c(1:8,58)) %>% head()
##   make address  all num3d  our over remove internet type
## 1 0.00    0.64 0.64    0 0.32 0.00   0.00    0.00 spam
## 2 0.21    0.28 0.50    0 0.14 0.28   0.21    0.07 spam
## 3 0.06    0.00 0.71    0 1.23 0.19   0.19    0.12 spam
## 4 0.00    0.00 0.00    0 0.63 0.00   0.31    0.63 spam
## 5 0.00    0.00 0.00    0 0.63 0.00   0.31    0.63 spam
## 6 0.00    0.00 0.00    0 1.85 0.00   0.00    1.85 spam
```

Détection de spam

- Sur 4 601 mails, on a pu identifier **1813 spams**.
- On a également mesuré sur chacun de ces mails la présence ou absence de **57 mots**.

```
> spam %>% select(c(1:8,58)) %>% head()
##   make address  all num3d  our over remove internet type
## 1 0.00    0.64 0.64    0 0.32 0.00   0.00    0.00 spam
## 2 0.21    0.28 0.50    0 0.14 0.28   0.21    0.07 spam
## 3 0.06    0.00 0.71    0 1.23 0.19   0.19    0.12 spam
## 4 0.00    0.00 0.00    0 0.63 0.00   0.31    0.63 spam
## 5 0.00    0.00 0.00    0 0.63 0.00   0.31    0.63 spam
## 6 0.00    0.00 0.00    0 1.85 0.00   0.00    1.85 spam
```

Question

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

Problématiques associées à l'apprentissage

- **Apprentissage supervisé** : expliquer/prédire une sortie $y \in \mathcal{Y}$ à partir d'entrées $x \in \mathcal{X}$;
- **Apprentissage non supervisé** : établir une typologie des observations ;
- **Règles d'association** : mesurer le lien entre différents produits ;
- **Systèmes de recommandation** : identifier les produits susceptibles d'intéresser des consommateurs.

Problématiques associées à l'apprentissage

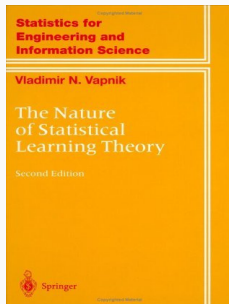
- **Apprentissage supervisé** : expliquer/prédire une sortie $y \in \mathcal{Y}$ à partir d'entrées $x \in \mathcal{X}$;
- **Apprentissage non supervisé** : établir une typologie des observations ;
- **Règles d'association** : mesurer le lien entre différents produits ;
- **Systèmes de recommandation** : identifier les produits susceptibles d'intéresser des consommateurs.

Nombreuses applications

finance, économie, marketing, biologie, médecine...

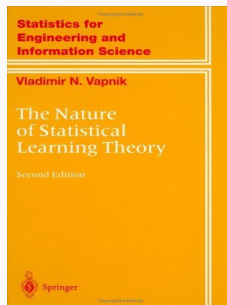
Approche mathématique

- Ouvrage fondateur : [Vapnik, 2000]

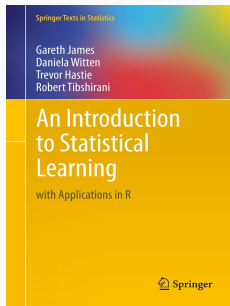
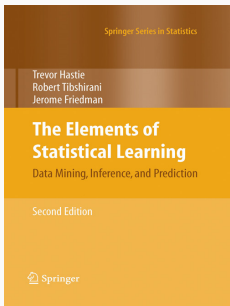


Approche mathématique

- Ouvrage fondateur : [Vapnik, 2000]
- voir aussi [Bousquet et al., 2003].



The Elements of Statistical Learning [[Hastie et al., 2009](#), [James et al., 2015](#)]



- Disponibles (avec jeux de données, codes...) aux url :

<https://web.stanford.edu/~hastie/ElemStatLearn/>

<http://www-bcf.usc.edu/~gareth/ISL/>

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

Régression vs Discrimination

- **Données** de type **entrée-sortie** : $d_n = (x_1, y_1), \dots, (x_n, y_n)$ où $x_i \in \mathcal{X}$ représente l'entrée et $y_i \in \mathcal{Y}$ la sortie.

Objectifs

1. **Expliquer** le(s) mécanisme(s) liant les entrée x_i aux sorties y_i ;
2. **Prédire** « au mieux » la sortie y associée à une nouvelle entrée $x \in \mathcal{X}$.

Régression vs Discrimination

- **Données** de type **entrée-sortie** : $d_n = (x_1, y_1), \dots, (x_n, y_n)$ où $x_i \in \mathcal{X}$ représente l'entrée et $y_i \in \mathcal{Y}$ la sortie.

Objectifs

1. **Expliquer** le(s) mécanisme(s) liant les entrée x_i aux sorties y_i ;
2. **Prédire** « au mieux » la sortie y associée à une nouvelle entrée $x \in \mathcal{X}$.

Vocabulaire

- Lorsque la variable à expliquer est quantitative ($\mathcal{Y} \subseteq \mathbb{R}$), on parle de **régression**.
- Lorsqu'elle est qualitative ($\text{Card}(\mathcal{Y})$ fini), on parle de **discrimination** ou de **classification supervisée**.

Exemples

- La plupart des problèmes présentés précédemment peuvent être appréhendés dans un contexte d'apprentissage supervisé : on cherche à expliquer une sortie y par des entrées x :

y_i	x_i	
Chiffre	image	Discri.
Mot	courbe	Discri.
Spam	présence/absence de mots	Discri.
C. en O_3	données météo.	Régression

Exemples

- La plupart des problèmes présentés précédemment peuvent être appréhendés dans un contexte d'**apprentissage supervisé** : on cherche à expliquer une sortie y par des entrées x :

y_i	x_i	
Chiffre	image	Discri.
Mot	courbe	Discri.
Spam	présence/absence de mots	Discri.
C. en O_3	données météo.	Régression

Remarque

- La nature des variables associées aux **entrées** x_i est **variée** (quantitative, qualitative, fonctionnelle...).

Un début de formalisation mathématique

- Etant données des observations $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ on cherche à **expliquer/prédire** les sortie $y_i \in \mathcal{Y}$ à partir des entrées $x_i \in \mathcal{X}$.

Un début de formalisation mathématique

- Etant données des observations $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ on cherche à **expliquer/prédire** les sortie $y_i \in \mathcal{Y}$ à partir des entrées $x_i \in \mathcal{X}$.
- Il s'agit donc de trouver **une fonction de prévision** $f : \mathcal{X} \rightarrow \mathcal{Y}$ telle que

$$f(x_i) \approx y_i, i = 1, \dots, n.$$

Un début de formalisation mathématique

- Etant données des observations $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ on cherche à **expliquer/prédire** les sortie $y_i \in \mathcal{Y}$ à partir des entrées $x_i \in \mathcal{X}$.
- Il s'agit donc de trouver **une fonction de prévision** $f : \mathcal{X} \rightarrow \mathcal{Y}$ telle que

$$f(x_i) \approx y_i, i = 1, \dots, n.$$

- Nécessité de se donner un **critère** qui permette de mesurer la qualité des fonctions de prévision f .

Un début de formalisation mathématique

- Etant données des observations $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ on cherche à **expliquer/prédire** les sortie $y_i \in \mathcal{Y}$ à partir des entrées $x_i \in \mathcal{X}$.
- Il s'agit donc de trouver **une fonction de prévision** $f : \mathcal{X} \rightarrow \mathcal{Y}$ telle que

$$f(x_i) \approx y_i, i = 1, \dots, n.$$

- Nécessité de se donner un **critère** qui permette de mesurer la qualité des fonctions de prévision f .
- Le plus souvent, on utilise une **fonction de perte** $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ telle que

$$\begin{cases} \ell(y, y') = 0 & \text{si } y = y' \\ \ell(y, y') > 0 & \text{si } y \neq y'. \end{cases}$$

- On suppose que les données $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ sont des réalisations d'un n -échantillon $\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ de loi inconnue.
- Les X_i sont des variables aléatoires à valeurs dans \mathcal{X} , les Y_i dans \mathcal{Y} .
- Le plus souvent on supposera que les couples $(X_i, Y_i), i = 1, \dots, n$ sont i.i.d de loi P .

Approche statistique

- On suppose que les données $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ sont des réalisations d'un n -échantillon $\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ de loi inconnue.
- Les X_i sont des variables aléatoires à valeurs dans \mathcal{X} , les Y_i dans \mathcal{Y} .
- Le plus souvent on supposera que les couples $(X_i, Y_i), i = 1, \dots, n$ sont i.i.d de loi P .

Performance d'une fonction de prévision

- Etant donné une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$, la performance d'une fonction (mesurable) de prévision $f : \mathcal{X} \rightarrow \mathcal{Y}$ est mesurée par

$$\mathcal{R}(f) = \mathbf{E}[\ell(Y, f(X))]$$

où (X, Y) est indépendant des (X_i, Y_i) et de même loi P .

Approche statistique

- On suppose que les données $d_n = \{(x_1, y_1), \dots, (x_n, y_n)\}$ sont des réalisations d'un n -échantillon $\mathcal{D}_n = \{(X_1, Y_1), \dots, (X_n, Y_n)\}$ de loi inconnue.
- Les X_i sont des variables aléatoires à valeurs dans \mathcal{X} , les Y_i dans \mathcal{Y} .
- Le plus souvent on supposera que les couples $(X_i, Y_i), i = 1, \dots, n$ sont i.i.d de loi P .

Performance d'une fonction de prévision

- Etant donné une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$, la performance d'une fonction (mesurable) de prévision $f : \mathcal{X} \rightarrow \mathcal{Y}$ est mesurée par

$$\mathcal{R}(f) = \mathbf{E}[\ell(Y, f(X))]$$

où (X, Y) est indépendant des (X_i, Y_i) et de même loi P .

- $\mathcal{R}(f)$ est appelé risque ou erreur de généralisation de f .

Aspect théorique

- Pour une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$ donnée, le problème **théorique** consiste à trouver

$$f^* \in \operatorname{argmin}_f \mathcal{R}(f).$$

- Une telle fonction f^* (si elle existe) est appelée **fonction de prévision optimale** pour la perte ℓ .

Aspect pratique

- La fonction de prévision optimale f^* dépend le plus souvent de la loi P des (X, Y) qui est en pratique **inconnue**.

Aspect pratique

- La fonction de prévision optimale f^* dépend le plus souvent de la loi P des (X, Y) qui est en pratique **inconnue**.
- Le job du statisticien est de trouver un **estimateur** $f_n = f_n(., \mathcal{D}_n)$ tel que $\mathcal{R}(f_n) \approx \mathcal{R}(f^*)$.

Aspect pratique

- La fonction de prévision optimale f^* dépend le plus souvent de la loi P des (X, Y) qui est en pratique **inconnue**.
- Le job du statisticien est de trouver un **estimateur** $f_n = f_n(., \mathcal{D}_n)$ tel que $\mathcal{R}(f_n) \approx \mathcal{R}(f^*)$.

Définition

- *Un algorithme de prévision est représenté par une suite $(f_n)_n$ d'applications (mesurables) telles que pour $n \geq 1$, $f_n : (\mathcal{X} \times (\mathcal{X} \times \mathcal{Y})^n) \rightarrow \mathcal{Y}$.*
- *On dit que la suite $(f_n)_n$ est **universellement consistante** si $\forall P$*

$$\lim_{n \rightarrow \infty} \mathcal{R}(f_n) = \mathcal{R}(f^*).$$

Choix de la fonction de perte

- Le cadre mathématique développé précédemment sous-entend qu'une fonction est performante (voire optimale) vis-à-vis d'un critère (représenté par la fonction de perte ℓ).
- Un algorithme de prévision performant pour un critère ne sera pas forcément performant pour un autre.

Choix de la fonction de perte

- Le cadre mathématique développé précédemment sous-entend qu'une fonction est **performante** (voire **optimale**) vis-à-vis d'un **critère** (représenté par la **fonction de perte** ℓ)).
- Un algorithme de prévision performant pour un critère ne sera **pas forcément performant pour un autre**.

Conséquence pratique

Avant de s'attacher à construire un algorithme de prévision, il est **capital** de savoir **mesurer la performance** d'un algorithme de prévision.

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

- Dans un contexte de régression ($\mathcal{Y} = \mathbb{R}$), la **perte quadratique** est la plus souvent utilisée. Elle est définie par :

$$\begin{aligned}\ell : \mathbb{R} \times \mathbb{R} &\rightarrow \mathbb{R}^+ \\ (y, y') &\mapsto (y - y')^2\end{aligned}$$

- Le **risque** pour une **fonction de prévision** ou **régresseur** $m : \mathcal{X} \rightarrow \mathbb{R}$ est alors donné par

$$\mathcal{R}(m) = \mathbf{E}((Y - m(X))^2).$$

- Dans un contexte de discrimination binaire ($\mathcal{Y} = \{-1, 1\}$), la **perte indicatrice** est la plus souvent utilisée. Elle est définie par :

$$\begin{aligned}\ell : \{-1, 1\} \times \{-1, 1\} &\rightarrow \mathbb{R}^+ \\ (y, y') &\mapsto \mathbf{1}_{y \neq y'}\end{aligned}$$

- Le **risque** pour une **fonction de prévision** ou **règle de prévision** $g : \mathcal{X} \rightarrow \{-1, 1\}$ est alors donné par

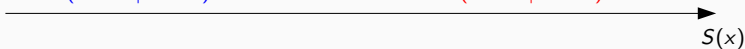
$$\mathcal{R}(g) = \mathbf{E}(\mathbf{1}_{g(X) \neq Y}) = \mathbf{P}(g(X) \neq Y).$$

Fonction de score

- On reste dans un cadre de **classification binaire** ($\mathcal{Y} = \{-1, 1\}$).
- Mais... plutôt que de chercher une règle de prévision $g : \mathcal{X} \rightarrow \{-1, 1\}$, on **cherche une fonction** $S : \mathcal{X} \rightarrow \mathbb{R}$ telle que

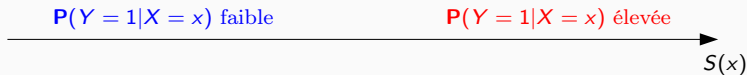
$P(Y = 1|X = x)$ faible

$P(Y = 1|X = x)$ élevée



Fonction de score

- On reste dans un cadre de **classification binaire** ($\mathcal{Y} = \{-1, 1\}$).
- Mais... plutôt que de chercher une règle de prévision $g : \mathcal{X} \rightarrow \{-1, 1\}$, on **cherche une fonction** $S : \mathcal{X} \rightarrow \mathbb{R}$ telle que



- Une telle fonction est appelée **fonction de score** : plutôt que de prédire directement le groupe d'un nouvel individu $x \in \mathcal{X}$, on lui donne une **note** $S(x)$
 - **élevée** si il a des "chances" d'être dans le groupe 1 ;
 - **faible** si il a des "chances" d'être dans le groupe -1 ;

Courbe ROC et AUC

- On utilise souvent la **courbe ROC** pour **visualiser** la performance d'un score :

$$\begin{cases} x(s) = \alpha(s) = 1 - sp(s) = \mathbf{P}(S(X) > s | Y = -1) \\ y(s) = 1 - \beta(s) = se(s) = \mathbf{P}(S(X) \geq s | Y = 1) \end{cases}$$

Courbe ROC et AUC

- On utilise souvent la **courbe ROC** pour **visualiser** la performance d'un score :

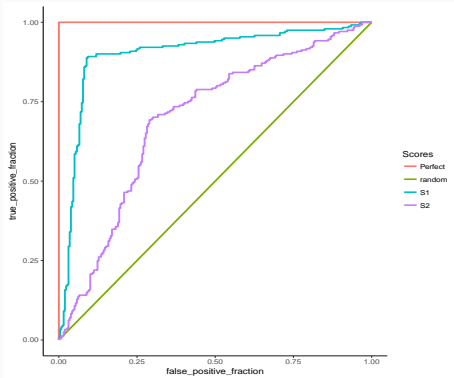
$$\begin{cases} x(s) = \alpha(s) = 1 - sp(s) = \mathbf{P}(S(X) > s | Y = -1) \\ y(s) = 1 - \beta(s) = se(s) = \mathbf{P}(S(X) \geq s | Y = 1) \end{cases}$$

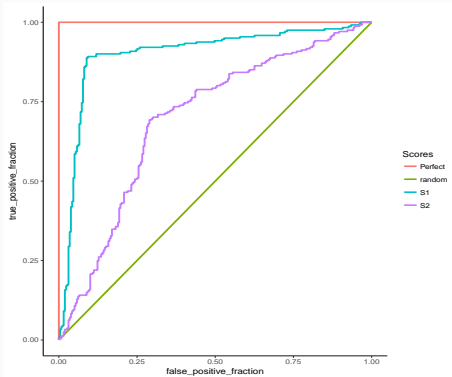
- On déduit de ce critère un **risque** pour les scores en considérant l'**aire sous la courbe ROC (AUC)** :

$$\mathcal{R}(S) = \text{AUC}(S).$$

Propriété

- $0.5 \leq \text{AUC}(S) \leq 1$.
- Plus l'AUC est **grand**, **meilleur** est le score.





```
> library(pROC)
> df1 %>% group_by(Scores) %>% summarize(auc(D,M))
## # A tibble: 4 x 2
##   Scores   'auc(D, M)'
##   <chr>         <dbl>
## 1 Perfect         1
## 2 random         0.5
## 3 S1             0.896
## 4 S2             0.699
```

Résumé

	Perte $\ell(y, f(x))$	Risque $\mathbf{E}[\ell(Y, f(X))]$	Champion f^*
Régression	$(y - f(x))^2$	$\mathbf{E}[Y - f(X)]^2$	$\mathbf{E}[Y X = x]$
Classif. binaire	$\mathbf{1}_{y \neq f(x)}$	$\mathbf{P}(Y \neq f(X))$	Bayes
Scoring		$AUC(S)$	$\mathbf{P}(Y = 1 X = x)$

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

- n observations $(X_1, Y_1), \dots, (X_n, Y_n)$ i.i.d à valeurs dans $\mathcal{X} \times \mathcal{Y}$.

Objectif

Etant donnée une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$, on cherche un **algorithme de prévision** $f_n(x) = f_n(x, \mathcal{D}_n)$ qui soit "proche" de l'oracle f^* défini par

$$f^* \in \underset{f}{\operatorname{argmin}} \mathcal{R}(f)$$

où $\mathcal{R}(f) = \mathbf{E}[\ell(Y, f(X))]$.

Rappels

- n observations $(X_1, Y_1), \dots, (X_n, Y_n)$ i.i.d à valeurs dans $\mathcal{X} \times \mathcal{Y}$.

Objectif

Etant donnée une fonction de perte $\ell : \mathcal{Y} \times \mathcal{Y} \rightarrow \mathbb{R}^+$, on cherche un **algorithme de prévision** $f_n(x) = f_n(x, \mathcal{D}_n)$ qui soit "proche" de l'oracle f^* défini par

$$f^* \in \underset{f}{\operatorname{argmin}} \mathcal{R}(f)$$

où $\mathcal{R}(f) = \mathbf{E}[\ell(Y, f(X))]$.

Question

Etant donné un algorithme f_n , **que vaut son risque** $\mathcal{R}(f_n)$?

Risque empirique

- La loi de (X, Y) étant **inconnue** en pratique, il est **impossible de calculer** $\mathcal{R}(f_n) = \mathbf{E}[\ell(Y, f_n(X))]$.

Risque empirique

- La loi de (X, Y) étant **inconnue** en pratique, il est **impossible de calculer** $\mathcal{R}(f_n) = \mathbf{E}[\ell(Y, f_n(X))]$.
- **Première approche** : $\mathcal{R}(f_n)$ étant une espérance, on peut l'estimer (LGN) par sa **version empirique**

$$\mathcal{R}_n(f_n) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_n(X_i)).$$

Risque empirique

- La loi de (X, Y) étant **inconnue** en pratique, il est **impossible de calculer** $\mathcal{R}(f_n) = \mathbf{E}[\ell(Y, f_n(X))]$.
- **Première approche** : $\mathcal{R}(f_n)$ étant une espérance, on peut l'estimer (LGN) par sa **version empirique**

$$\mathcal{R}_n(f_n) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_n(X_i)).$$

Problème

- L'échantillon \mathcal{D}_n a **déjà été utilisé** pour construire l'algorithme de prévision $f_n \implies$ La LGN ne peut donc s'appliquer !
- **Conséquence** : $\mathcal{R}_n(f_n)$ conduit souvent à une **sous-estimation** de $\mathcal{R}(f_n)$.

Risque empirique

- La loi de (X, Y) étant **inconnue** en pratique, il est **impossible de calculer** $\mathcal{R}(f_n) = \mathbf{E}[\ell(Y, f_n(X))]$.
- **Première approche** : $\mathcal{R}(f_n)$ étant une espérance, on peut l'estimer (LGN) par sa **version empirique**

$$\mathcal{R}_n(f_n) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_n(X_i)).$$

Problème

- L'échantillon \mathcal{D}_n a **déjà été utilisé** pour construire l'algorithme de prévision $f_n \implies$ La LGN ne peut donc s'appliquer !
- **Conséquence** : $\mathcal{R}_n(f_n)$ conduit souvent à une **sous-estimation** de $\mathcal{R}(f_n)$.

Une solution

Utiliser des méthodes de type **validation croisée** ou **bootstrap**.

Apprentissage - Validation ou Validation hold out

- Elle consiste à séparer l'échantillon \mathcal{D}_n en :
 1. un échantillon d'apprentissage $\mathcal{D}_{n,app}$ pour construire f_n ;
 2. un échantillon de validation $\mathcal{D}_{n,test}$ utilisé pour estimer le risque de f_n .

Apprentissage - Validation ou Validation hold out

- Elle consiste à séparer l'échantillon \mathcal{D}_n en :
 1. un **échantillon d'apprentissage** $\mathcal{D}_{n,app}$ pour construire f_n ;
 2. un **échantillon de validation** $\mathcal{D}_{n,test}$ utilisé pour estimer le risque de f_n .

Algorithme

Entrées. \mathcal{D}_n : données, $\{\mathcal{A}, \mathcal{V}\}$: partition de $\{1, \dots, n\}$.

1. Construire l'algorithme de prédiction sur $\mathcal{D}_{n,app} = \{(X_i, Y_i) : i \in \mathcal{A}\}$, on le note $f_{n,app}$;
2. Calculer $\hat{\mathcal{R}}_n(f_n) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \ell(Y_i, f_{n,app}(X_i))$.

Apprentissage - Validation ou Validation hold out

- Elle consiste à séparer l'échantillon \mathcal{D}_n en :
 1. un **échantillon d'apprentissage** $\mathcal{D}_{n,app}$ pour construire f_n ;
 2. un **échantillon de validation** $\mathcal{D}_{n,test}$ utilisé pour estimer le risque de f_n .

Algorithme

Entrées. \mathcal{D}_n : données, $\{\mathcal{A}, \mathcal{V}\}$: partition de $\{1, \dots, n\}$.

1. Construire l'algorithme de prédiction sur $\mathcal{D}_{n,app} = \{(X_i, Y_i) : i \in \mathcal{A}\}$, on le note $f_{n,app}$;
2. Calculer $\hat{\mathcal{R}}_n(f_n) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \ell(Y_i, f_{n,app}(X_i))$.

Commentaires

Nécessite d'avoir un **nombre suffisant d'observations** dans

1. $\mathcal{D}_{n,app}$ pour bien ajuster l'algorithme de prévision ;
2. $\mathcal{D}_{n,test}$ pour bien estimer l'erreur de l'algorithme.

Validation croisée K -blocs

- **Principe** : répéter l'algorithme apprentissage/validation sur **différentes partitions**.

Algorithme - CV

Entrées. \mathcal{D}_n : données, K un entier qui divise n ;

1. Construire une partition $\{\mathcal{I}_1, \dots, \mathcal{I}_K\}$ de $\{1, \dots, n\}$;
2. Pour $k = 1, \dots, K$
 - 2.1 $\mathcal{I}_{app} = \{1, \dots, n\} \setminus \mathcal{I}_k$ et $\mathcal{I}_{test} = \mathcal{I}_k$;
 - 2.2 Construire l'algorithme de prédiction sur $\mathcal{D}_{n,app} = \{(X_i, Y_i) : i \in \mathcal{I}_{app}\}$, on le note $f_{n,k}$;
 - 2.3 En déduire $f_n(X_i) = f_{n,k}(X_i)$ pour $i \in \mathcal{I}_{test}$;
3. Retourner

$$\hat{\mathcal{R}}_n(f_n) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_n(X_i)).$$

- Plus adapté que la technique apprentissage/validation lorsqu'on a **peu d'observations**.
- Le **choix de K** doit être fait par l'utilisateur (souvent $K = 10$).

Leave one out

- Lorsque $K = n$, on parle de validation croisée **leave one out** ;
- Le risque est alors estimé par

$$\hat{\mathcal{R}}_n(f_n) = \frac{1}{n} \sum_{i=1}^n \ell(Y_i, f_n^i(X_i))$$

où f_n^i désigne l'algorithme de prévision construit sur \mathcal{D}_n **amputé de la i -ème observation**.

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

- La plupart des modèles statistiques renvoient des estimateurs qui dépendent de paramètres λ à calibrer.

- La plupart des modèles statistiques renvoient des estimateurs qui dépendent de **paramètres λ à calibrer**.

Exemples

- nombres de variables dans un modèle linéaire ou logistique.
- paramètre de pénalités pour les régressions pénalisées.
- profondeur des arbres.
- nombre de plus proches voisins.
- nombre d'itérations en boosting.
- ...

- La plupart des modèles statistiques renvoient des estimateurs qui dépendent de paramètres λ à calibrer.

Exemples

- nombres de variables dans un modèle linéaire ou logistique.
- paramètre de pénalités pour les régressions pénalisées.
- profondeur des arbres.
- nombre de plus proches voisins.
- nombre d'itérations en boosting.
- ...

Remarque importante

Le choix de ces paramètres est le plus souvent crucial pour la performance de l'estimateur sélectionné.

- Le paramètre λ à sélectionner représente le plus souvent la complexité du modèle :

- Le paramètre λ à sélectionner représente le plus souvent la **complexité du modèle** :

Complexité \Rightarrow compromis biais/variance

- λ petit \Rightarrow modèle peu flexible \Rightarrow mauvaise adéquation sur les données \Rightarrow biais \nearrow , variance \searrow .

- Le paramètre λ à sélectionner représente le plus souvent la **complexité du modèle** :

Complexité \Rightarrow compromis biais/variance

- λ petit \Rightarrow modèle peu flexible \Rightarrow mauvaise adéquation sur les données \Rightarrow biais \nearrow , variance \searrow .
- λ grand \Rightarrow modèle trop flexible \Rightarrow **sur-ajustement** \Rightarrow biais \searrow , variance \nearrow .

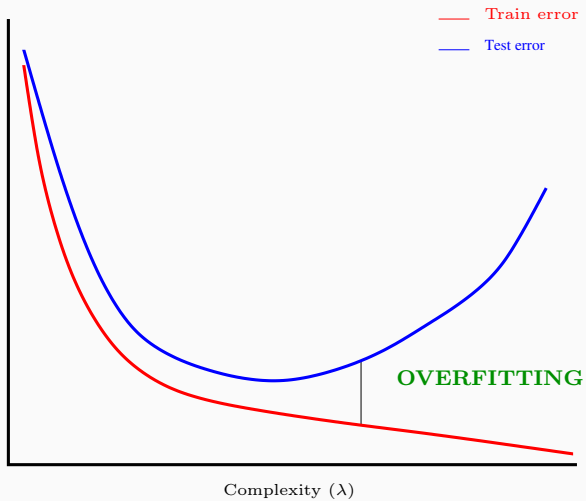
- Le paramètre λ à sélectionner représente le plus souvent la **complexité du modèle** :

Complexité \Rightarrow compromis biais/variance

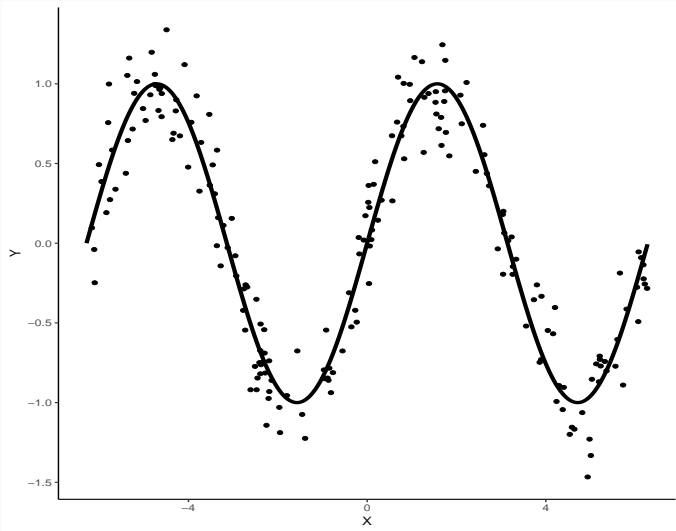
- λ petit \Rightarrow modèle peu flexible \Rightarrow mauvaise adéquation sur les données \Rightarrow biais \nearrow , variance \searrow .
- λ grand \Rightarrow modèle trop flexible \Rightarrow **sur-ajustement** \Rightarrow biais \searrow , variance \nearrow .

Overfitting

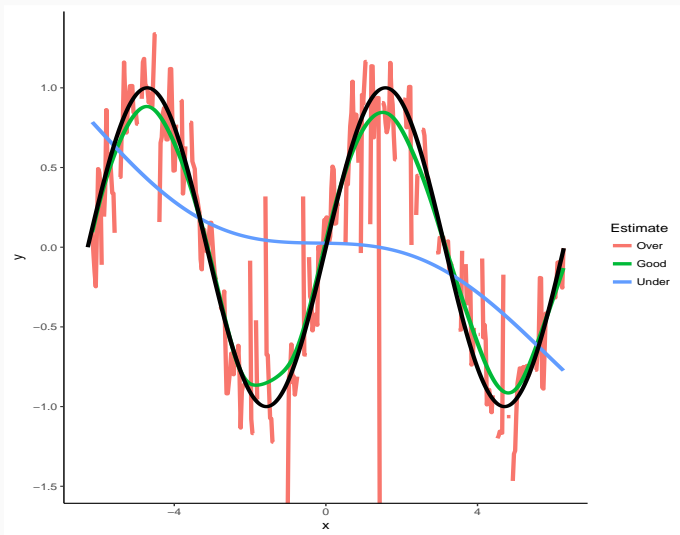
Sur-ajuster signifie que le modèle va (trop) bien ajuster sur les données d'apprentissage, il aura du mal à s'adapter à de nouveaux individus.



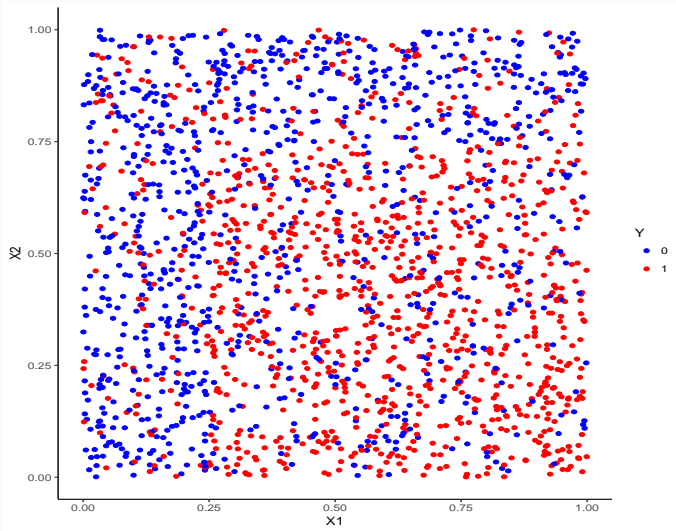
Overfitting en regression



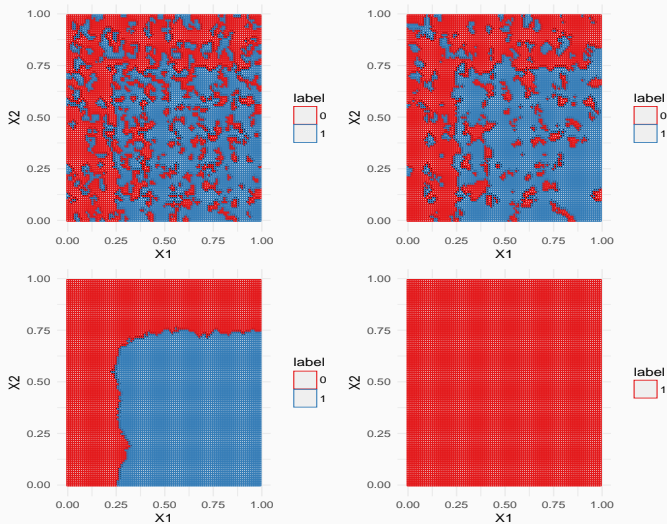
Overfitting en regression



Overfitting en classification supervisée



Overfitting en classification supervisée



Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie

Le package caret

- Il permet d'évaluer la performance de plus de 230 méthodes :
<http://topepo.github.io/caret/index.html>

Le package caret

- Il permet d'évaluer la performance de **plus de 230 méthodes** :
<http://topepo.github.io/caret/index.html>
- Il suffit d'indiquer :
 - la **méthode** (logistique, ppv, arbre, randomForest...)
 - Une grille pour les **paramètres** (nombre de ppv...)
 - Le **critère de performance** (erreur de classification, AUC, risque quadratique...)
 - La méthode d'**estimation du critère** (apprentissage validation, validation croisée, bootstrap...)

Apprentissage-validation

```
> library(caret)
> K_cand <- data.frame(k=seq(1,500,by=20))
> library(caret)
> ctrl1 <- trainControl(method="LGOCV",number=1,index=list(1:1500))
> e1 <- train(Y~.,data=donnees,method="knn",trControl=ctrl1,tuneGrid=K_cand)
> e1

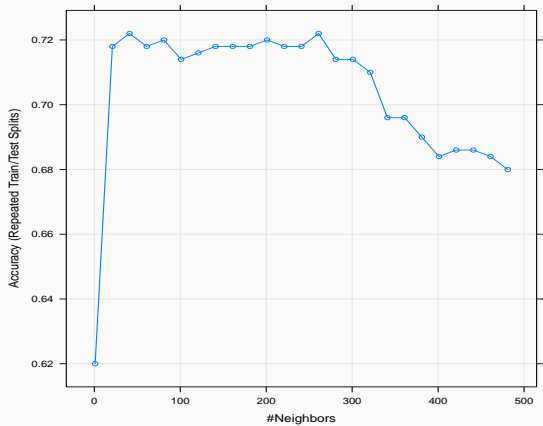
## k-Nearest Neighbors
##
## 2000 samples
##    2 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 75%)
## Summary of sample sizes: 1500
## Resampling results across tuning parameters:
##
##    k    Accuracy  Kappa
##    1  0.620      0.2382571
##   21  0.718      0.4342076
##   41  0.722      0.4418388
```

##	61	0.718	0.4344073
##	81	0.720	0.4383195
##	101	0.714	0.4263847
##	121	0.716	0.4304965
##	141	0.718	0.4348063
##	161	0.718	0.4348063
##	181	0.718	0.4348063
##	201	0.720	0.4387158
##	221	0.718	0.4350056
##	241	0.718	0.4350056
##	261	0.722	0.4428232
##	281	0.714	0.4267894
##	301	0.714	0.4269915
##	321	0.710	0.4183621
##	341	0.696	0.3893130
##	361	0.696	0.3893130
##	381	0.688	0.3727988
##	401	0.684	0.3645329
##	421	0.686	0.3686666
##	441	0.686	0.3679956
##	461	0.684	0.3638574
##	481	0.680	0.3558050

Accuracy was used to select the optimal model using the largest value.

The final value used for the model was k = 261.

```
> plot(e1)
```



Validation croisée

```
> library(doMC)
> registerDoMC(cores = 3)
> ctrl12 <- trainControl(method="cv",number=10)
> e2 <- train(Y~.,data=dapp,method="knn",trControl=ctrl12,tuneGrid=K_cand)
> e2

## k-Nearest Neighbors
##
## 1500 samples
##    2 predictor
##    2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1350, 1350, 1350, 1350, 1350, 1350, ...
## Resampling results across tuning parameters:
##
##    k    Accuracy    Kappa
##    1  0.6240000  0.2446251
##   21  0.7393333  0.4745290
##   41  0.7306667  0.4570024
##   61  0.7340000  0.4636743
```



```
##      81  0.7333333  0.4632875
##     101  0.7313333  0.4593480
##     121  0.7326667  0.4624249
##     141  0.7333333  0.4640787
##     161  0.7366667  0.4708178
##     181  0.7313333  0.4602309
##     201  0.7326667  0.4626618
##     221  0.7293333  0.4559741
##     241  0.7306667  0.4585960
##     261  0.7353333  0.4676751
##     281  0.7286667  0.4537842
##     301  0.7253333  0.4463516
##     321  0.7173333  0.4294524
##     341  0.7113333  0.4168003
##     361  0.7080000  0.4099303
##     381  0.7140000  0.4213569
##     401  0.7073333  0.4073761
##     421  0.7100000  0.4126434
##     441  0.7066667  0.4054984
##     461  0.6966667  0.3844183
##     481  0.6860000  0.3612515
##
```

```
## Accuracy was used to select the optimal model using the largest value.
```

```
## The final value used for the model was k = 21.
```

Validation croisée répétée

```
> ctrl3 <- trainControl(method="repeatedcv",repeats=5,number=10)
> e3 <- train(Y~.,data=dapp,method="knn",trControl=ctrl3,tuneGrid=K_cand)
> e3

## k-Nearest Neighbors
##
## 1500 samples
##      2 predictor
##      2 classes: '0', '1'
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 1350, 1350, 1350, 1350, 1350, 1350, ...
## Resampling results across tuning parameters:
##
##      k      Accuracy      Kappa
##      1  0.6232000  0.2438066
##     21  0.7354667  0.4665640
##     41  0.7314667  0.4585144
##     61  0.7317333  0.4592608
##     81  0.7302667  0.4568784
##    101  0.7310667  0.4589567
```

```
## 121 0.7320000 0.4609326
## 141 0.7322667 0.4616077
## 161 0.7336000 0.4643374
## 181 0.7340000 0.4649895
## 201 0.7332000 0.4632905
## 221 0.7325333 0.4620114
## 241 0.7316000 0.4600484
## 261 0.7305333 0.4578098
## 281 0.7286667 0.4536040
## 301 0.7238667 0.4434101
## 321 0.7189333 0.4330787
## 341 0.7136000 0.4215865
## 361 0.7122667 0.4183400
## 381 0.7098667 0.4131761
## 401 0.7090667 0.4112403
## 421 0.7058667 0.4043164
## 441 0.7001333 0.3920207
## 461 0.6952000 0.3811374
## 481 0.6872000 0.3636126
##
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was k = 21.
```

Critère AUC

```
> donnees1 <- donnees
> names(donnees1)[3] <- c("Class")
> levels(donnees1$Class) <- c("G0", "G1")
> ctrl11 <- trainControl(method="LGOCV", number=1, index=list(1:1500),
+                        classProbs=TRUE, summary=twoClassSummary)
> e4 <- train(Class~., data=donnees1, method="knn", trControl=ctrl11,
+            metric="ROC", tuneGrid=K_cand)
> e4

## k-Nearest Neighbors
##
## 2000 samples
##      2 predictor
##      2 classes: 'G0', 'G1'
##
## No pre-processing
## Resampling: Repeated Train/Test Splits Estimated (1 reps, 75%)
## Summary of sample sizes: 1500
## Resampling results across tuning parameters:
##
```

##	k	ROC	Sens	Spec
##	1	0.6190866	0.5983264	0.6398467
##	21	0.7171484	0.6903766	0.7432950
##	41	0.7229757	0.6861925	0.7547893
##	61	0.7200500	0.6945607	0.7394636
##	81	0.7255567	0.6945607	0.7432950
##	101	0.7319450	0.6903766	0.7356322
##	121	0.7382452	0.6945607	0.7356322
##	141	0.7353757	0.7029289	0.7318008
##	161	0.7308549	0.7029289	0.7318008
##	181	0.7351272	0.7029289	0.7318008
##	201	0.7340050	0.7029289	0.7356322
##	221	0.7324099	0.7071130	0.7279693
##	241	0.7349028	0.7071130	0.7279693
##	261	0.7365780	0.7071130	0.7356322
##	281	0.7349749	0.6987448	0.7279693
##	301	0.7356963	0.7029289	0.7241379
##	321	0.7341493	0.6861925	0.7318008
##	341	0.7343898	0.6527197	0.7356322
##	361	0.7306385	0.6527197	0.7356322
##	381	0.7301816	0.6359833	0.7394636
##	401	0.7270957	0.6276151	0.7356322
##	421	0.7255487	0.6317992	0.7356322

```
## 441 0.7258933 0.6192469 0.7471264
## 461 0.7220619 0.6150628 0.7471264
## 481 0.7236330 0.6108787 0.7432950
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 121.
```

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

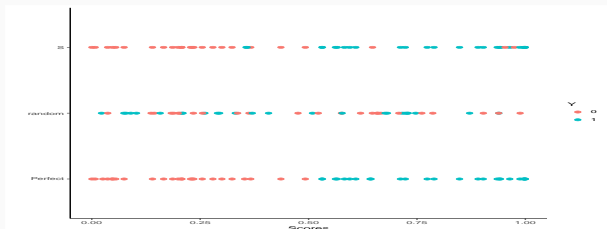
Le sur-apprentissage

Le package caret

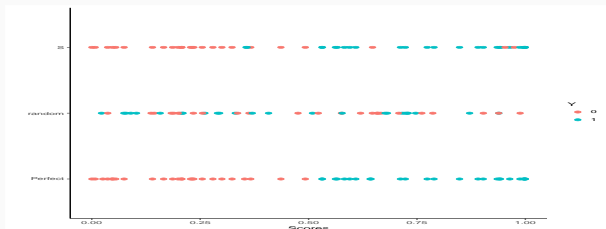
Annexe : compléments sur les scores

Bibliographie

Scores parfait et aléatoire



Scores parfait et aléatoire

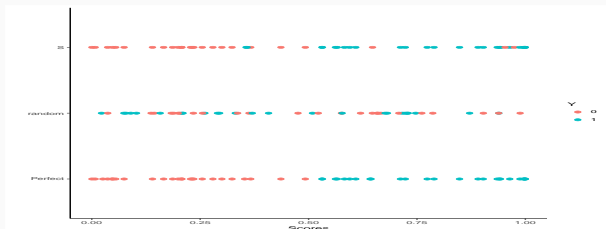


Définition

- Score parfait* : il est tel qu'il existe un seuil s^* tel que

$$P(Y = 1 | S(X) \geq s^*) = 1 \quad \text{et} \quad P(Y = -1 | S(X) < s^*) = 1.$$

Scores parfait et aléatoire



Définition

- *Score parfait* : il est tel qu'il existe un seuil s^* tel que

$$P(Y = 1 | S(X) \geq s^*) = 1 \quad \text{et} \quad P(Y = -1 | S(X) < s^*) = 1.$$

- *Score aléatoire* : il est tel que $S(X)$ et Y sont indépendantes.

Lien score/règle de prévision

- Etant donné un score S , on peut déduire une **règle de prévision** en **fixant un seuil** s (la réciproque n'est pas vraie) :

$$g_s(x) = \begin{cases} 1 & \text{si } S(x) \geq s \\ -1 & \text{sinon.} \end{cases}$$

- Cette règle définit la **table de confusion**

	$g_s(X) = -1$	$g_s(X) = 1$
$Y = -1$	OK	E_1
$Y = 1$	E_2	OK

Lien score/règle de prévision

- Etant donné un score S , on peut déduire une **règle de prévision** en **fixant un seuil** s (la réciproque n'est pas vraie) :

$$g_s(x) = \begin{cases} 1 & \text{si } S(x) \geq s \\ -1 & \text{sinon.} \end{cases}$$

- Cette règle définit la **table de confusion**

	$g_s(X) = -1$	$g_s(X) = 1$
$Y = -1$	OK	E_1
$Y = 1$	E_2	OK

- Pour chaque seuil s , on distingue deux types d'**erreur**

$$\alpha(s) = \mathbf{P}(g_s(X) = 1 | Y = -1) = \mathbf{P}(S(X) \geq s | Y = -1)$$

et

$$\beta(s) = \mathbf{P}(g_s(X) = -1 | Y = 1) = \mathbf{P}(S(X) < s | Y = 1).$$

On définit également

- **Spécificité** : $sp(s) = \mathbf{P}(S(X) < s | Y = -1) = 1 - \alpha(s)$
- **Sensibilité** : $se(s) = \mathbf{P}(S(X) \geq s | Y = 1) = 1 - \beta(s)$

On définit également

- **Spécificité** : $sp(s) = P(S(X) < s | Y = -1) = 1 - \alpha(s)$
- **Sensibilité** : $se(s) = P(S(X) \geq s | Y = 1) = 1 - \beta(s)$

Performance d'un score

Elle se mesure généralement en **visualisant** les erreurs $\alpha(s)$ et $\beta(s)$ et/ou la spécificité et la sensibilité pour **tous les seuils s** .

Courbe ROC

- **Idée** : représenter sur un graphe 2d les deux types d'erreur pour **tous les seuils** s .

Définition

C'est une **courbe paramétrée** par le seuil :

$$\begin{cases} x(s) = \alpha(s) = 1 - sp(s) = \mathbf{P}(S(X) > s | Y = -1) \\ y(s) = 1 - \beta(s) = se(s) = \mathbf{P}(S(X) \geq s | Y = 1) \end{cases}$$

Courbe ROC

- **Idée** : représenter sur un graphe 2d les deux types d'erreur pour **tous les seuils** s .

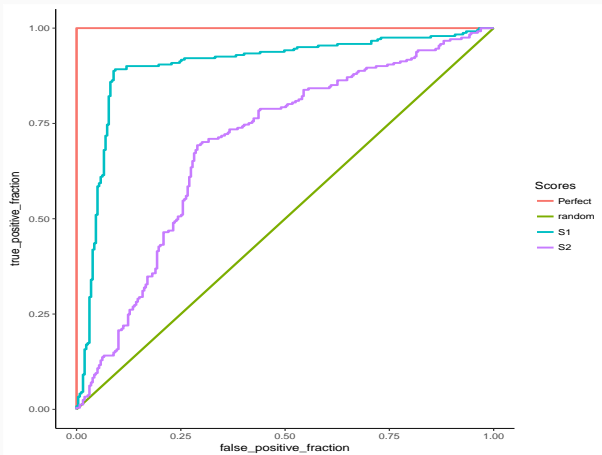
Définition

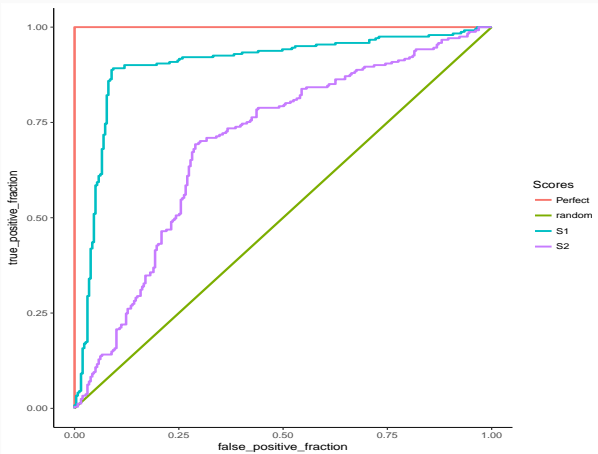
C'est une **courbe paramétrée** par le seuil :

$$\begin{cases} x(s) = \alpha(s) = 1 - sp(s) = \mathbf{P}(S(X) > s | Y = -1) \\ y(s) = 1 - \beta(s) = se(s) = \mathbf{P}(S(X) \geq s | Y = 1) \end{cases}$$

Remarque

- La courbe ROC d'un score **parfait** passe par le point **(0,1)**.
- La courbe ROC d'un score **aléatoire** correspond à la **première bissectrice**.





Interprétation

On mesurera la performance d'un score par sa capacité à se rapprocher de la droite d'équation $y = 1$ le plus vite possible.

Définition

- L'*aire sous la courbe ROC* d'un score S , notée $AUC(S)$ est souvent utilisée pour mesurer sa performance.
- Pour un score *parfait* on a $AUC(S) = 1$, pour un score *aléatoire* $AUC(S) = 1/2$.

Définition

- L'*aire sous la courbe ROC* d'un score S , notée $AUC(S)$ est souvent utilisée pour mesurer sa performance.
- Pour un score *parfait* on a $AUC(S) = 1$, pour un score *aléatoire* $AUC(S) = 1/2$.

Proposition

- Etant données deux observations (X_1, Y_1) et (X_2, Y_2) indépendantes et de même loi que (X, Y) , on a

$$AUC(S) = \mathbf{P}(S(X_1) \geq S(X_2) | (Y_1, Y_2) = (1, -1)).$$

```
> library(pROC)
> df1 %>% group_by(Scores) %>% summarize(auc(D,M))
## # A tibble: 4 x 2
##   Scores   'auc(D, M)'
##   <chr>      <dbl>
## 1 Perfect      1
## 2 random    0.5
## 3 S1        0.896
## 4 S2        0.699
```

Score optimal

- Le critère $AUC(S)$ peut être interprété comme une **fonction de perte** pour un score S ;
- Se pose donc la question d'existence d'un **score optimal** S^* vis-à-vis de ce critère.

Score optimal

- Le critère $AUC(S)$ peut être interprété comme une **fonction de perte** pour un score S ;
- Se pose donc la question d'existence d'un **score optimal** S^* vis-à-vis de ce critère.

Théorème ([Clémenton et al., 2008])

Soit $S^*(x) = \mathbf{P}(Y = 1|X = x)$, on a alors pour toutes fonctions de score S

$$AUC(S^*) \geq AUC(S).$$

Score optimal

- Le critère $AUC(S)$ peut être interprété comme une **fonction de perte** pour un score S ;
- Se pose donc la question d'existence d'un **score optimal** S^* vis-à-vis de ce critère.

Théorème ([Cléménçon et al., 2008])

Soit $S^*(x) = \mathbf{P}(Y = 1|X = x)$, on a alors pour toutes fonctions de score S

$$AUC(S^*) \geq AUC(S).$$

Conséquence

Le problème pratique consistera à trouver un **"bon" estimateur**

$S_n(x) = S_n(x, \mathcal{D}_n)$ de

$$S^*(x) = \mathbf{P}(Y = 1|X = x).$$

Motivations

Cadre mathématique pour l'apprentissage supervisé

Exemples de fonction de perte

Estimation du risque

Le sur-apprentissage

Le package caret

Annexe : compléments sur les scores

Bibliographie



Besse, P. and Laurent, B.

Apprentissage Statistique modélisation, prévision, data mining.

INSA - Toulouse.

http://www.math.univ-toulouse.fr/~besse/pub/Appren_stat.pdf.



Bousquet, O., Boucheron, S., and Lugosi, G. (2003).

Introduction to Statistical Learning Theory, chapter Advanced Lectures on Machine Learning.

Springer.



Cléménçon, S., Lugosi, G., and Vayatis, N. (2008).

Ranking and empirical minimization of u-statistics.

The Annals of Statistics, 36(2) :844–874.



Hastie, T., Tibshirani, R., and Friedman, J. (2009).

The Elements of Statistical Learning : Data Mining, Inference, and Prediction.

Springer, second edition.



James, G., Witten, D., Hastie, T., and Tibshirani, R. (2015).

The Elements of Statistical Learning : Data Mining, Inference, and Prediction.

Springer.



Vapnik, V. (2000).

The Nature of Statistical Learning Theory.

Springer, second edition.

Deuxième partie II

Support vector machine

SVM - cas séparable

SVM : cas non séparable

SVM non linéaire : astuce du noyau

- **Discrimination binaire** : Y à valeurs dans $\{-1, 1\}$ et $X = (X_1, \dots, X_p)$ dans \mathbb{R}^p .

Objectif

- Estimer la **fonction de score** $S(x) = \mathbf{P}(Y = 1|X = x)$;
- En déduire une **règle de classification** $g : \mathbb{R}^p \rightarrow \{-1, 1\}$.

- Elles consistent à **séparer** l'espace des X par un **hyperplan**.
- On classe ensuite 1 d'un côté de l'hyperplan, -1 de l'autre côté.

Règles linéaires

- Elles consistent à **séparer** l'espace des X par un **hyperplan**.
- On classe ensuite 1 d'un côté de l'hyperplan, -1 de l'autre côté.

Mathématiquement

- On cherche une combinaison linéaire des variables $w_1X_1 + \dots + w_pX_p$.
- Règle associée :

$$g(x) = \begin{cases} 1 & \text{si } w_1X_1 + \dots + w_pX_p \geq 0 \\ -1 & \text{sinon.} \end{cases}$$

Exemple 1 : régression logistique

- Modèle :

$$\text{logit} \frac{p(x)}{1 - p(x)} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$$

où $p(x) = \mathbf{P}(Y = 1|X = x)$.

- Règle de classification :

$$g(x) = \begin{cases} 1 & \text{si } p(x) \geq 0.5 \\ -1 & \text{sinon.} \end{cases}$$

Exemple 1 : régression logistique

- Modèle :

$$\text{logit} \frac{p(x)}{1 - p(x)} = \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p$$

où $p(x) = \mathbf{P}(Y = 1 | X = x)$.

- Règle de classification :

$$g(x) = \begin{cases} 1 & \text{si } p(x) \geq 0.5 \\ -1 & \text{sinon.} \end{cases}$$

- équivalent à

$$g(x) = \begin{cases} 1 & \text{si } \beta_0 + \beta_1 x_1 + \dots + \beta_p x_p \geq 0 \\ -1 & \text{sinon.} \end{cases}$$

Exemple 2 : LDA

- **Modèle** : $\mathcal{L}(X|Y = k) = \mathcal{N}(\mu_k, \Sigma), k = 0, 1.$
- **Règle de classification** :

$$g(x) = \begin{cases} 1 & \text{si } p(x) \geq 0.5 \\ -1 & \text{sinon.} \end{cases}$$

Exemple 2 : LDA

- **Modèle** : $\mathcal{L}(X|Y = k) = \mathcal{N}(\mu_k, \Sigma)$, $k = 0, 1$.
- **Règle de classification** :

$$g(x) = \begin{cases} 1 & \text{si } p(x) \geq 0.5 \\ -1 & \text{sinon.} \end{cases}$$

- équivalent à

$$g(x) = \begin{cases} 1 & \text{si } c + x' \Sigma^{-1}(\mu_1 - \mu_0) \geq 0 \\ -1 & \text{sinon.} \end{cases}$$

Illustration avec $p = 2$

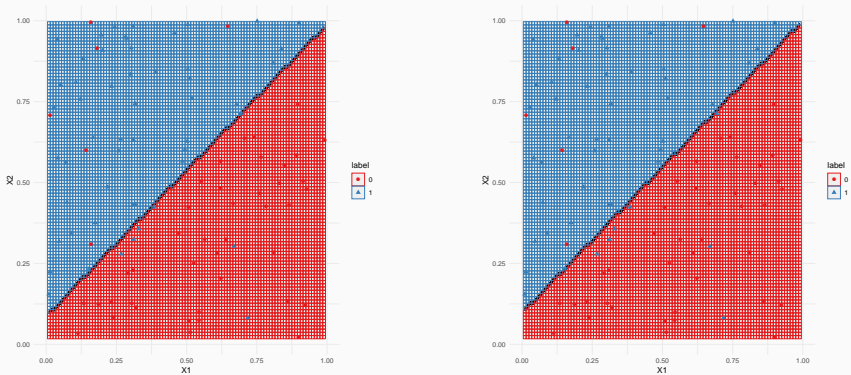


Figure 1 – Règles logistique (gauche) et lda (droite).

- Ces approches linéaires s'obtiennent à partir d'un modèle statistique
 - sur la loi de Y sachant X pour la logistique ;
 - sur la loi de X sachant Y pour la discriminante linéaire.

- Ces approches linéaires s'obtiennent à partir d'un modèle statistique
 - sur la loi de Y sachant X pour la logistique ;
 - sur la loi de X sachant Y pour la discriminante linéaire.
- L'approche SVM repose sur le calcul direct du "meilleur" hyperplan séparateur qui sera déterminé à partir d'algorithmes d'optimisation.

SVM - cas séparable

SVM : cas non séparable

SVM non linéaire : astuce du noyau

Bibliographie

En plus des documents cités précédemment, cette partie s'appuie sur les diapos de cours de

- **Magalie Fromont**, Apprentissage statistique, Université Rennes 2 ([Fromont, 2015]).
- **Jean-Philippe Vert**, *Support vector machines and applications in computational biology*, disponible à l'url <http://cbio.ensmp.fr/~jvert/svn/kernelcourse/slides/kernel2h/kernel2h.pdf>

Remarque

Les aspects techniques ne seront pas présentés ici, on pourra les trouver à l'url

https://www.dropbox.com/s/p5awah6ij8ykiil/slides_apprentissage.pdf?dl=0

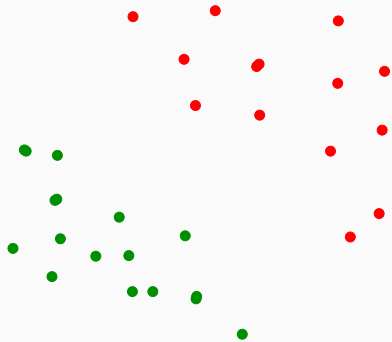
- L'approche SVM [Vapnik, 2000] peut être vue comme une généralisation de "recherche d'hyperplan optimal".

- L'approche SVM [Vapnik, 2000] peut être vue comme une généralisation de "recherche d'hyperplan optimal".

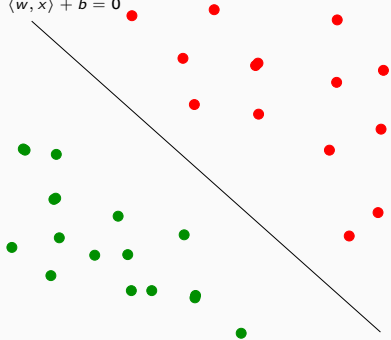
Cas simple

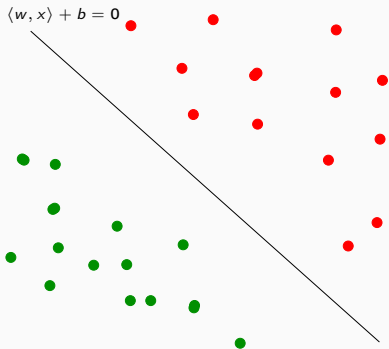
Les données $(x_1, y_1), \dots, (x_n, y_n)$ sont dites linéairement séparables si il existe $(w, b) \in \mathbb{R}^p \times \mathbb{R}$ tel que pour tout i :

- $y_i = 1$ si $\langle w, x_i \rangle + b = w^t x_i + b > 0$;
- $y_i = -1$ si $\langle w, x_i \rangle + b = w^t x_i + b < 0$.



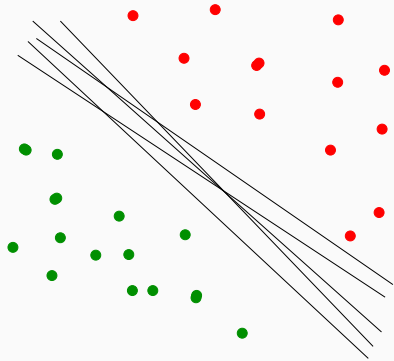
$$\langle w, x \rangle + b = 0$$





Vocabulaire

- L'équation $\langle w, x \rangle + b$ définit un **hyperplan séparateur** de vecteur normal w .
- La fonction $\text{signe}(\langle w, x \rangle + b)$ est une règle de **discrimination** potentielle.

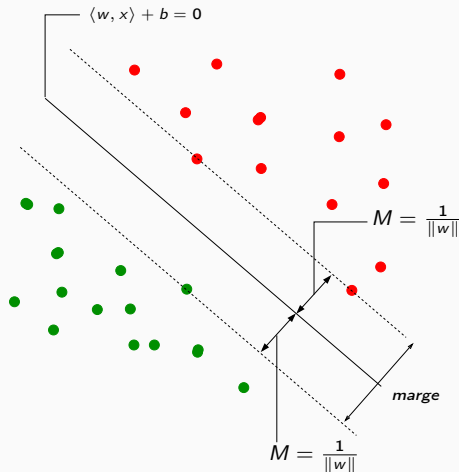


Problème

Il existe une **infinité d'hyperplans séparateurs** donc une **infinité de règles de discrimination potentielles**.

Solution

[Vapnik, 2000] propose de choisir l'hyperplan ayant la **marge maximale**.



Le problème d'optimisation

- On veut trouver l'hyperplan de **marge maximale** qui **sépare** les groupes.

Le problème d'optimisation

- On veut trouver l'hyperplan de **marge maximale** qui **sépare** les groupes.

Hyperplan séparateur optimal

Solution du problème **d'optimisation sous contrainte** :

- **Version 1** :

$$\max_{w, b, \|w\|=1} M$$

sous les contraintes $y_i(w^t x_i + b) \geq M, i = 1, \dots, n.$

Le problème d'optimisation

- On veut trouver l'hyperplan de **marge maximale** qui **sépare** les groupes.

Hyperplan séparateur optimal

Solution du problème **d'optimisation sous contrainte** :

- Version 1** :

$$\max_{w, b, \|w\|=1} M$$

sous les contraintes $y_i(w^t x_i + b) \geq M, i = 1, \dots, n.$

- Version 2** :

$$\min_{w, b} \frac{1}{2} \|w\|^2$$

sous les contraintes $y_i(w^t x_i + b) \geq 1, i = 1, \dots, n.$

- On obtient

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i.$$

où les α_i^* sont des constantes positives qui s'obtiennent en résolvant le **dual** du problème précédent.

- De plus, b^* s'obtient en résolvant

$$\alpha_i^* [y_i (x_i^t w + b) - 1] = 0$$

pour un α_i^* non nul.

- On obtient

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i.$$

où les α_i^* sont des constantes positives qui s'obtiennent en résolvant le **dual** du problème précédent.

- De plus, b^* s'obtient en résolvant

$$\alpha_i^* [y_i (x_i^t w + b) - 1] = 0$$

pour un α_i^* non nul.

Remarque

w^* s'écrit comme une **combinaison linéaire** des x_i .

Propriété (conditions KKT)

$$\alpha_i^*[y_i(x_i^t w^* + b) - 1] = 0, i = 1, \dots, n.$$

Propriété (conditions KKT)

$$\alpha_i^* [y_i(x_i^t w^* + b) - 1] = 0, i = 1, \dots, n.$$

Conséquence (importante)

- Si $\alpha_i^* = 0$ alors $y_i(x_i^t w^* + b) = 1$ et x_i est **sur la marge**.
- w^* se calcule **uniquement** à partir de ces points là.

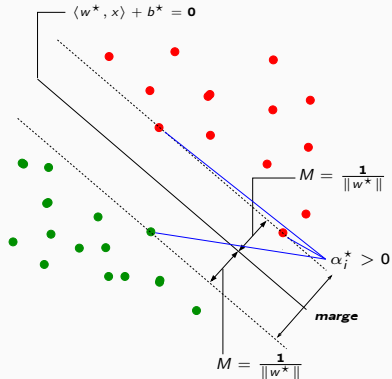
Propriété (conditions KKT)

$$\alpha_i^* [y_i(x_i^t w^* + b) - 1] = 0, i = 1, \dots, n.$$

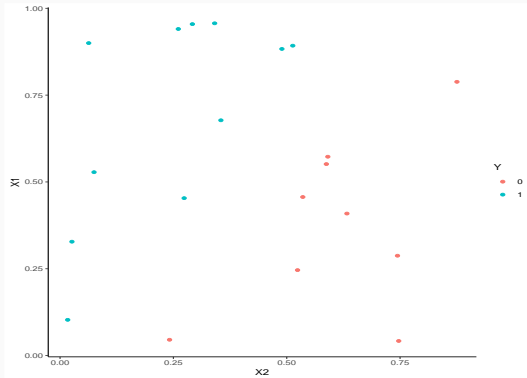
Conséquence (importante)

- Si $\alpha_i^* = 0$ alors $y_i(x_i^t w^* + b) = 1$ et x_i est **sur la marge**.
- w^* se calcule **uniquement** à partir de ces points là.
- Ces points sont appelés les **vecteurs supports** de la SVM.

Représentation



- La fonction `svm` du package `e1071` permet d'ajuster des SVM.



```
> library(e1071)
> mod.svm <- svm(Y~.,data=df,kernel="linear",cost=10000000000)
```

La fonction svm

- Les vecteurs supports :

```
> mod.svm$index  
## [1] 6 14 12
```

- `mod.svm$coefs` = $\alpha_i^* y_i$ pour chaque vecteur support

```
> mod.svm$coefs  
##           [,1]  
## [1,] 1.898982  
## [2,] 1.905497  
## [3,] -3.804479
```

La fonction svm

- Les vecteurs supports :

```
> mod.svm$index
## [1] 6 14 12
```

- `mod.svm$coefs` = $\alpha_i^* y_i$ pour chaque vecteur support

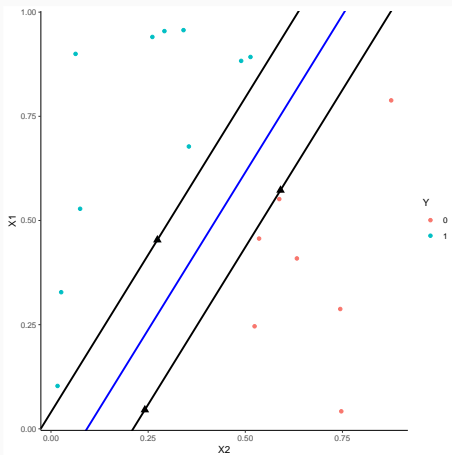
```
> mod.svm$coefs
##           [,1]
## [1,]  1.898982
## [2,]  1.905497
## [3,] -3.804479
```

- On peut en déduire l'hyperplan séparateur

```
> w <- apply(mod.svm$coefs*df[mod.svm$index,2:3],2,sum)
> b <- -mod.svm$rho
> w
##           X1           X2
## -0.5470382  0.5427583
> b
## [1] -0.4035113
```

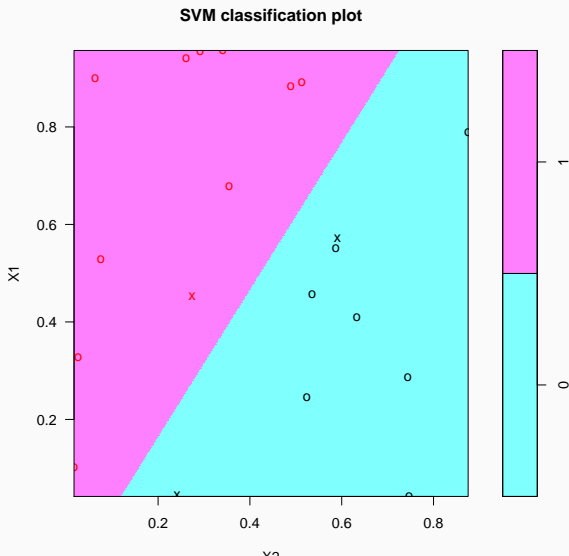
On peut ainsi visualiser

- les vecteurs supports ;
- l'hyperplan séparateur ;
- la marge.



- La fonction `plot` donne aussi une représentation de l'**hyperplan séparateur**.

```
> plot(mod.svm,data=df,fill=TRUE,grid=500)
```



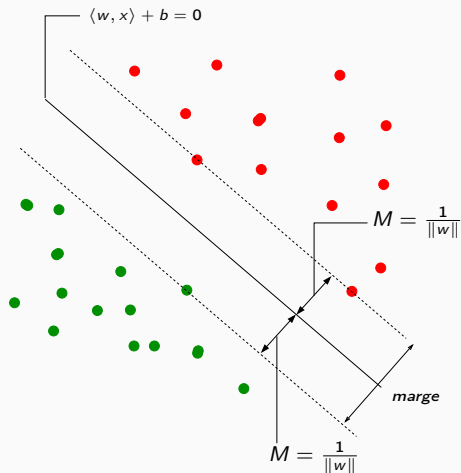
SVM - cas séparable

SVM : cas non séparable

SVM non linéaire : astuce du noyau

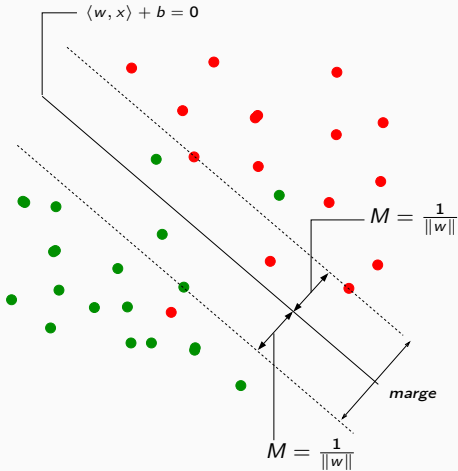
Problème

Dans la vraie vie, les données ne sont (quasiment) **jamais linéairement séparables**...



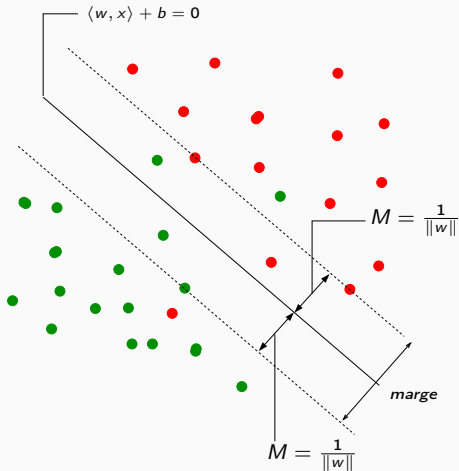
Problème

Dans la vraie vie, les données ne sont (quasiment) **jamais linéairement séparables**...



Problème

Dans la vraie vie, les données ne sont (quasiment) **jamais linéairement séparables**...



Idée

Autoriser certains points

1. à être **bien classés** mais à l'**intérieur** de la marge ;
2. et/ou à être **mal classés**.

Rappel : cas séparable

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

sous les contraintes $y_i(w^t x_i + b) \geq 1, i = 1, \dots, n.$

- Les contraintes $y_i(w^t x_i + b) \geq 1$ signifient que tous les points se trouvent en dehors de la frontière définie par la **marge** ;

Slack variables

Rappel : cas séparable

$$\min_{w,b} \frac{1}{2} \|w\|^2$$

sous les contraintes $y_i(w^t x_i + b) \geq 1, i = 1, \dots, n.$

- Les contraintes $y_i(w^t x_i + b) \geq 1$ signifient que tous les points se trouvent en dehors de la frontière définie par la **marge** ;
- **Cas non séparable** : le problème ci-dessus n'admet pas de solution !

Variables ressorts

On introduit des **variables ressorts (slack variables)** positives ξ_1, \dots, ξ_n telles que $y_i(w^t x_i + b) \geq 1 - \xi_i$. 2 cas sont à distinguer :

1. $\xi_i \in [0, 1] \implies$ bien classé mais **dans** la région définie par la **marge** ;
2. $\xi_i > 1 \implies$ **mal classé**.

- Bien entendu, on souhaite avoir le **maximum** de variables ressorts ξ_i **nulles**;
- Lorsque $\xi_i > 0$, on souhaite que ξ_i soit le **plus petit possible**.

- Bien entendu, on souhaite avoir le **maximum** de variables ressorts ξ_i nulles;
- Lorsque $\xi_i > 0$, on souhaite que ξ_i soit le **plus petit possible**.

Cas non séparable : problème d'optimisation (primal)

- Il s'agit de minimiser en (w, b, ξ)

$$\frac{1}{2} \|w\|^2$$

$$\text{sous les contraintes } \left\{ \begin{array}{l} y_i(w^t x_i + b) \geq 1 \end{array} \right.$$

- Bien entendu, on souhaite avoir le **maximum** de variables ressorts ξ_i nulles ;
- Lorsque $\xi_i > 0$, on souhaite que ξ_i soit le **plus petit possible**.

Cas non séparable : problème d'optimisation (primal)

- Il s'agit de minimiser en (w, b, ξ)

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{sous les contraintes } \begin{cases} y_i(w^t x_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0, i = 1, \dots, n. \end{cases}$$

- Bien entendu, on souhaite avoir le **maximum** de variables ressorts ξ_i nulles ;
- Lorsque $\xi_i > 0$, on souhaite que ξ_i soit le **plus petit possible**.

Cas non séparable : problème d'optimisation (primal)

- Il s'agit de minimiser en (w, b, ξ)

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{sous les contraintes } \begin{cases} y_i(w^t x_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0, i = 1, \dots, n. \end{cases}$$

- $C > 0$ est un paramètre à calibrer (**paramètre de coût**).

- Bien entendu, on souhaite avoir le **maximum** de variables ressorts ξ_i nulles;
- Lorsque $\xi_i > 0$, on souhaite que ξ_i soit le **plus petit possible**.

Cas non séparable : problème d'optimisation (primal)

- Il s'agit de minimiser en (w, b, ξ)

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i$$

$$\text{sous les contraintes } \begin{cases} y_i(w^t x_i + b) \geq 1 - \xi_i \\ \xi_i \geq 0, i = 1, \dots, n. \end{cases}$$

- $C > 0$ est un paramètre à calibrer (**paramètre de coût**).
- Le **cas séparable** correspond à $C \rightarrow \infty$.

- Les **solutions** de ce nouveau problème d'optimisation s'obtiennent de la **même façon** que dans le cas séparable (Lagrangien, problème dual...).
- L'**hyperplan optimal** est défini par

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$$

et b^* est solution de $y_i(\langle w^*, x_i \rangle + b^*) = 1$ pour tout i tel que $0 < \alpha_i^* < C$.

- Les **solutions** de ce nouveau problème d'optimisation s'obtiennent de la **même façon** que dans le cas séparable (Lagrangien, problème dual...).
- L'**hyperplan optimal** est défini par

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$$

et b^* est solution de $y_i(\langle w^*, x_i \rangle + b^*) = 1$ pour tout i tel que $0 < \alpha_i^* < C$.

Vecteurs supports

- Les x_i tels que $\alpha_i^* > 0$ sont les vecteurs supports ;
- On en distingue 2 types :
 1. ceux **sur la frontière** définie par la marge : $\xi_i^* = 0$;
 2. ceux **en dehors** : $\xi_i^* > 0$ et $\alpha_i^* = C$.

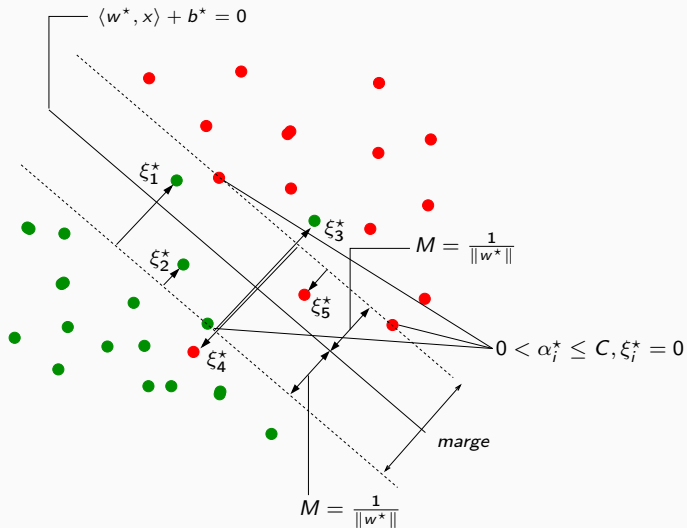
- Les **solutions** de ce nouveau problème d'optimisation s'obtiennent de la **même façon** que dans le cas séparable (Lagrangien, problème dual...).
- L'**hyperplan optimal** est défini par

$$w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$$

et b^* est solution de $y_i(\langle w^*, x_i \rangle + b^*) = 1$ pour tout i tel que $0 < \alpha_i^* < C$.

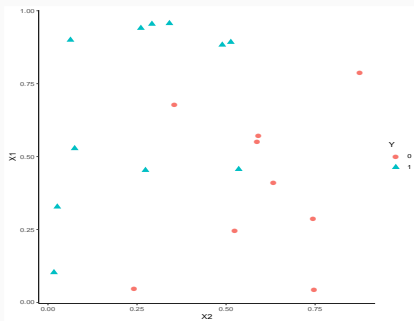
Vecteurs supports

- Les x_i tels que $\alpha_i^* > 0$ sont les vecteurs supports ;
- On en distingue 2 types :
 1. ceux **sur la frontière** définie par la marge : $\xi_i^* = 0$;
 2. ceux **en dehors** : $\xi_i^* > 0$ et $\alpha_i^* = C$.
- Les vecteurs **non supports** vérifient $\alpha_i^* = 0$ et $\xi_i^* = 0$.



Le coin R

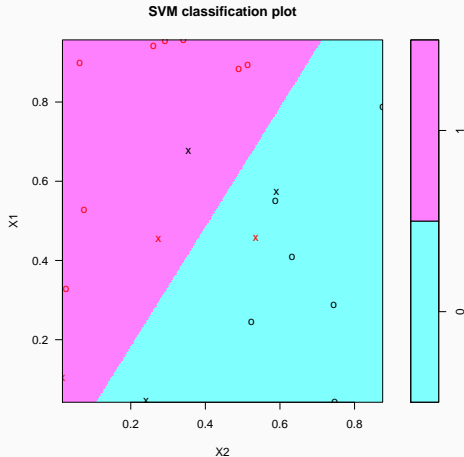
- On utilise la même fonction que dans le **cas séparable** (svm du package e1071);
- L'argument cost correspond à la **constante de régularisation C** .



```
> mod.svm1 <- svm(Y~.,data=df1,kernel="linear",cost=1000)
> mod.svm1$index
## [1] 6 13 14 10 12 15
```

Visualisation de l'hyperplan séparateur

```
> plot(mod.svm1,data=df1,fill=TRUE,grid=500)
```



Ce paramètre régule le **compromis biais/variance** de la svm :

- $C \searrow$: la marge est privilégiée et les $\xi_i \nearrow \Rightarrow$

Choix de C

Ce paramètre régule le **compromis biais/variance** de la svm :

- $C \searrow$: la marge est privilégiée et les $\xi_i \nearrow \implies$ beaucoup d'observations dans la marge ou **mal classées** (et donc beaucoup de vecteurs supports).

Ce paramètre règle le **compromis biais/variance** de la svm :

- $C \searrow$: la marge est privilégiée et les $\xi_i \nearrow \implies$ beaucoup d'observations dans la marge ou **mal classées** (et donc beaucoup de vecteurs supports).
- $C \nearrow \implies \xi_i \searrow$ donc moins d'observations mal classées \implies

Choix de C

Ce paramètre régule le **compromis biais/variance** de la svm :

- $C \searrow$: la marge est privilégiée et les $\xi_i \nearrow \implies$ beaucoup d'observations dans la marge ou **mal classées** (et donc beaucoup de vecteurs supports).
- $C \nearrow \implies \xi_i \searrow$ donc moins d'observations mal classées \implies **meilleur ajustement** mais petite marge \implies risque de **surajustement**.

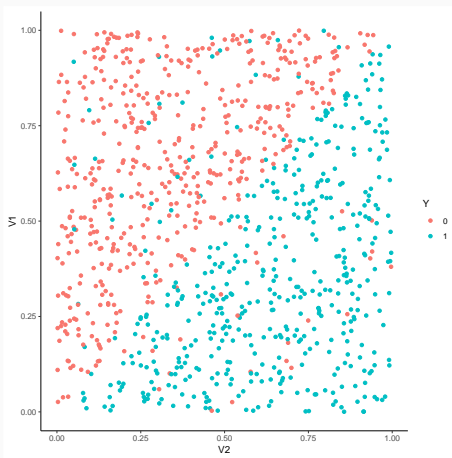
Conclusion

Il est donc très important de bien choisir ce paramètre.

- Le choix est souvent effectué de façon "classique" :
 1. On se donne un **critère de performance** (taux de mal classés par exemple) ;
 2. On **estime la valeur du critère** pour différentes valeurs de C ;
 3. On choisit la valeur de C pour laquelle le **critère estimé est minimum**.

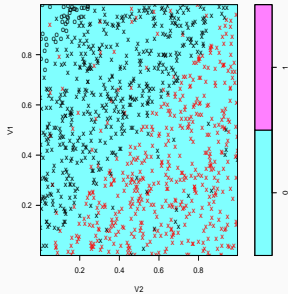
- Le choix est souvent effectué de façon "classique" :
 1. On se donne un **critère de performance** (taux de mal classés par exemple) ;
 2. On **estime la valeur du critère** pour différentes valeurs de C ;
 3. On choisit la valeur de C pour laquelle le **critère estimé est minimum**.
- La fonction **tune.svm** permet de choisir C en estimant le taux de mal classés par **validation croisée**. On peut aussi (bien entendu) utiliser la fonction **train** du package **caret**.

Un exemple

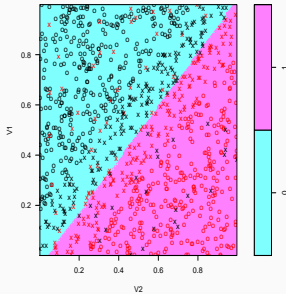


```
> mod.svm1 <- svm(Y~.,data=df3,kernel="linear",cost=0.000001)
> mod.svm2 <- svm(Y~.,data=df3,kernel="linear",cost=0.1)
> mod.svm3 <- svm(Y~.,data=df3,kernel="linear",cost=5)
```

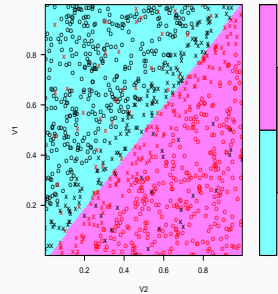
SVM classification plot



SVM classification plot



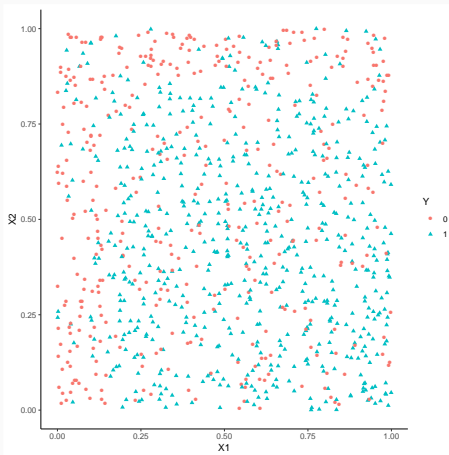
SVM classification plot



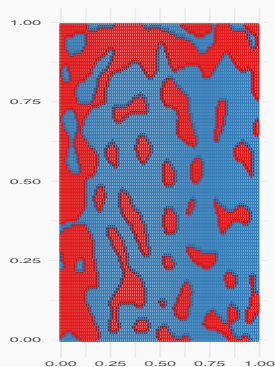
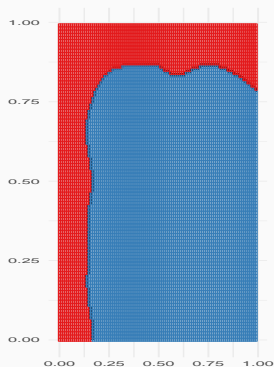
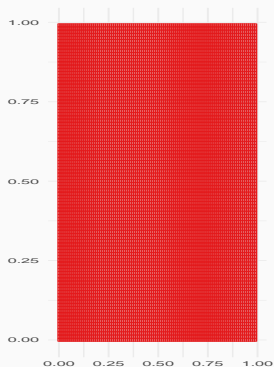
```
> mod.svm1$nSV
## [1] 480 480
> mod.svm2$nSV
## [1] 190 190
> mod.svm3$nSV
## [1] 166 165
```

Un autre exemple

- $n = 1000$ observations.




```
> model1 <- svm(Y~.,data=donnees,cost=0.001,kernel="radial",gamma=5)
> model2 <- svm(Y~.,data=donnees,cost=1,kernel="radial",gamma=5)
> model3 <- svm(Y~.,data=donnees,cost=100000,kernel="radial",gamma=5)
```



Choix de C avec tune

```
> tune.out <- tune(svm,Y~.,data=df3,kernel="linear",
+                 ranges=list(cost=c(0.001,0.01,1,10,100,1000)))
> summary(tune.out)
## Parameter tuning of 'svm':
## - sampling method: 10-fold cross validation
## - best parameters:
##   cost
##     1
## - best performance: 0.071
## - Detailed performance results:
##   cost error dispersion
## 1 1e-03 0.142 0.03675746
## 2 1e-02 0.084 0.03373096
## 3 1e+00 0.071 0.02766867
## 4 1e+01 0.072 0.02820559
## 5 1e+02 0.072 0.02820559
## 6 1e+03 0.071 0.02766867
```

```

> bestmod <- tune.out$best.model
> summary(bestmod)

##
## Call:
## best.tune(method = svm, train.x = Y ~ ., data = df3, ranges =
##      list(cost = c(0.001, 0.01, 1, 10, 100, 1000)), kernel = "linear")
##
## Parameters:
##      SVM-Type:  C-classification
##      SVM-Kernel:  linear
##              cost:  1
##              gamma: 0.5
##
## Number of Support Vectors: 336
##
## ( 168 168 )
##
## Number of Classes: 2
##
## Levels:
## 0 1

```

Choix de C avec caret

```
> library(caret)
> gr <- data.frame(C=c(0.001,0.01,1,10,100,1000))
> ctrl <- trainControl(method="repeatedcv",number=10,repeats=5)
> train(Y~.,data=df3,method="svmLinear",trControl=ctrl,tuneGrid=gr)

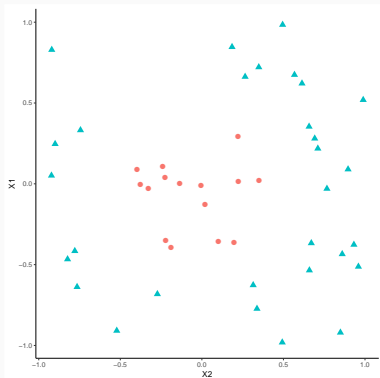
## Support Vector Machines with Linear Kernel
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 900, 900, 900, 900, 900, 900, ...
## Resampling results across tuning parameters:
##      C      Accuracy  Kappa
##  1e-03  0.8700      0.7377051
##  1e-02  0.9188      0.8369121
##  1e+00  0.9304      0.8604317
##  1e+01  0.9292      0.8580356
##  1e+02  0.9294      0.8584333
##  1e+03  0.9294      0.8584333
## Accuracy was used to select the optimal model using the largest value.
## The final value used for the model was C = 1.
```

SVM - cas séparable

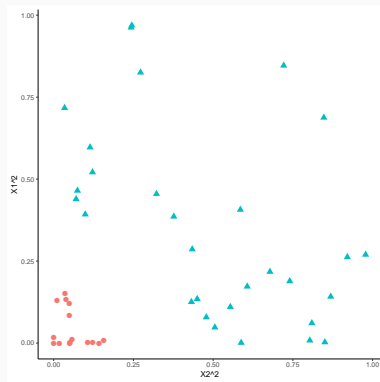
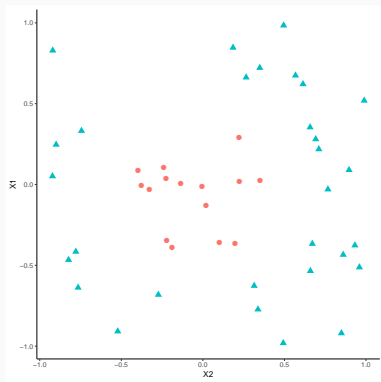
SVM : cas non séparable

SVM non linéaire : astuce du noyau

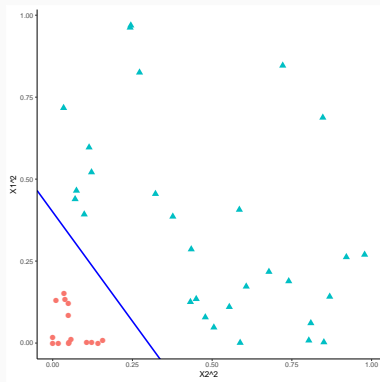
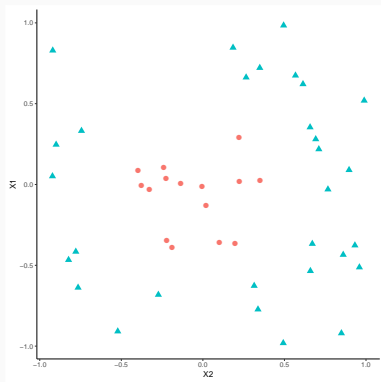
- Les solutions linéaires ne sont pas toujours intéressantes.



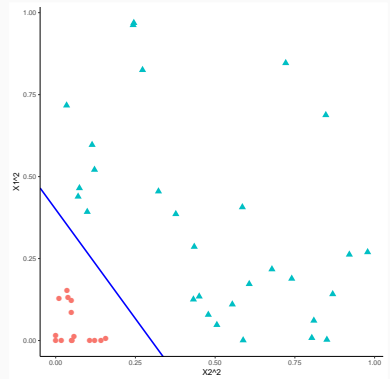
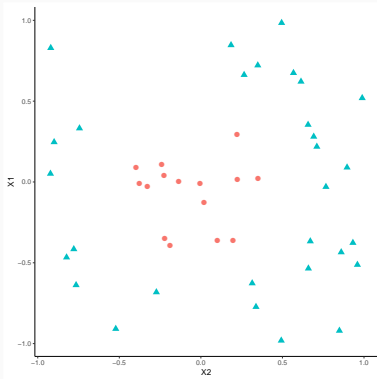
- Les solutions linéaires ne sont pas toujours intéressantes.



- Les solutions linéaires ne sont pas toujours intéressantes.



- Les solutions linéaires ne sont pas toujours intéressantes.



Idée

Trouver une transformation des données telle que les données transformées soient linéairement séparables.

Définition

Soit $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ une application qui va de l'espace des observations \mathcal{X} dans un Hilbert \mathcal{H} . Le **noyau** K entre x et x' associé à Φ est le produit scalaire entre $\Phi(x)$ et $\Phi(x')$:

$$\begin{aligned} K : \mathcal{X} \times \mathcal{X} &\rightarrow \mathbb{R} \\ (x, x') &\mapsto \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}. \end{aligned}$$

Définition

Soit $\Phi : \mathcal{X} \rightarrow \mathcal{H}$ une application qui va de l'espace des observations \mathcal{X} dans un Hilbert \mathcal{H} . Le **noyau** K entre x et x' associé à Φ est le produit scalaire entre $\Phi(x)$ et $\Phi(x')$:

$$\begin{aligned} K : \mathcal{X} \times \mathcal{X} &\rightarrow \mathbb{R} \\ (x, x') &\mapsto \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}. \end{aligned}$$

Exemple

Si $\mathcal{X} = \mathcal{H} = \mathbb{R}^2$ et $\varphi(x_1, x_2) = (x_1^2, x_2^2)$ alors

$$K(x, x') = (x_1 x'_1)^2 + (x_2 x'_2)^2.$$

L'astuce noyau

- L'astuce consiste donc à envoyer les observations x_i dans un espace de Hilbert \mathcal{H} appelé espace de représentation ou feature space...

L'astuce noyau

- L'astuce consiste donc à envoyer les observations x_i dans un espace de Hilbert \mathcal{H} appelé espace de représentation ou feature space...
- en espérant que les données $(\Phi(x_1), y_1), \dots, (\Phi(x_n), y_n)$ soient (presque) linéairement séparables de manière à appliquer une svm sur ces données transformées.

L'astuce noyau

- L'astuce consiste donc à envoyer les observations x_i dans un espace de Hilbert \mathcal{H} appelé espace de représentation ou feature space...
- en espérant que les données $(\Phi(x_1), y_1), \dots, (\Phi(x_n), y_n)$ soient (presque) linéairement séparables de manière à appliquer une svm sur ces données transformées.

Remarque

1. Beaucoup d'algorithmes linéaires (en particulier les SVM) peuvent être appliqués sur $\Phi(x)$ sans calculer explicitement Φ ! Il suffit de pouvoir calculer le noyau $K(x, x')$;

L'astuce noyau

- L'astuce consiste donc à envoyer les observations x_i dans un espace de Hilbert \mathcal{H} appelé espace de représentation ou feature space...
- en espérant que les données $(\Phi(x_1), y_1), \dots, (\Phi(x_n), y_n)$ soient (presque) linéairement séparables de manière à appliquer une svm sur ces données transformées.

Remarque

1. Beaucoup d'algorithmes linéaires (en particulier les SVM) peuvent être appliqués sur $\Phi(x)$ sans calculer explicitement Φ ! Il suffit de pouvoir calculer le noyau $K(x, x')$;
2. On n'a pas besoin de connaître l'espace \mathcal{H} ni l'application Φ , il suffit de se donner un noyau K !

- Le **problème dual** consiste à maximiser

$$L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^n \alpha_i \alpha_k y_i y_k \langle \mathbf{x}_i, \mathbf{x}_k \rangle$$

sous les contraintes
$$\begin{cases} 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0. \end{cases}$$

- La règle de décision s'obtient en calculant le signe de

$$f(x) = \sum_{i=1}^n \alpha_i^* y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b^*.$$

- Le **problème dual** consiste à maximiser

$$L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^n \alpha_i \alpha_k y_i y_k \langle \Phi(x_i), \Phi(x_k) \rangle$$

sous les contraintes
$$\begin{cases} 0 \leq \alpha_i \leq C, \quad i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0. \end{cases}$$

- La règle de décision s'obtient en calculant le signe de

$$f(x) = \sum_{i=1}^n \alpha_i^* y_i \langle \Phi(x_i), \Phi(x) \rangle + b^*.$$

- Le **problème dual** consiste à maximiser

$$L_D(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{k=1}^n \alpha_i \alpha_k y_i y_k K(x_i, x_k)$$

sous les contraintes
$$\begin{cases} 0 \leq \alpha_i \leq C, & i = 1, \dots, n \\ \sum_{i=1}^n \alpha_i y_i = 0. \end{cases}$$

- La règle de décision s'obtient en calculant le signe de

$$f(x) = \sum_{i=1}^n \alpha_i^* y_i K(x_i, x) + b^*.$$

Conclusion

- Pour calculer la svm, on n'a pas besoin de connaître \mathcal{H} ou Φ , il suffit de connaître K !

Conclusion

- Pour calculer la svm, on n'a pas besoin de connaître \mathcal{H} ou Φ , il suffit de connaître K !

Questions

Qu'est-ce qu'un noyau ? Comment construire un noyau ?

Conclusion

- Pour calculer la svm, on n'a pas besoin de connaître \mathcal{H} ou Φ , il suffit de connaître K !

Questions

Qu'est-ce qu'un noyau ? Comment construire un noyau ?

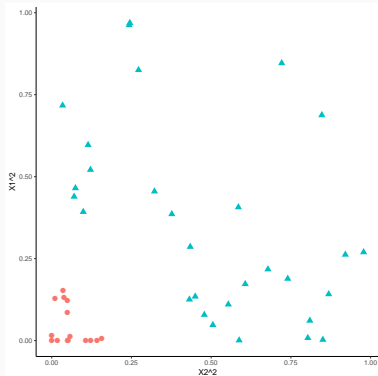
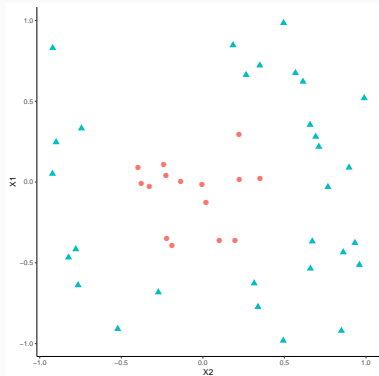
Théorème ([Aronszajn, 1950])

Une fonction $K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$ est un *noyau* si et seulement si elle est (symétrique) *définie positive*, c'est-à-dire ssi

1. $K(x, x') = K(x', x) \quad \forall (x, x') \in \mathcal{X}^2$;
2. $\forall (x_1, \dots, x_N) \in \mathcal{X}^N$ et $\forall (a_1, \dots, a_N) \in \mathbb{R}^N$

$$\sum_{i=1}^N \sum_{j=1}^N a_i a_j K(x_i, x_j) \geq 0.$$

Exemple



- Si

$$\begin{aligned}\Phi : \quad \mathbb{R}^2 &\rightarrow \mathbb{R}^3 \\ (x_1, x_2) &\mapsto (x_1^2, \sqrt{2}x_1x_2, x_2^2)\end{aligned}$$

alors $K(x, x') = (x^t x')^2$ (noyau polynomial de degré 2).

Exemples de noyau

1. **Linéaire** (sur \mathbb{R}^d) : $K(x, x') = x^t x'$.
2. **Polynomial** (sur \mathbb{R}^d) : $K(x, x') = (x^t x' + 1)^d$.
3. **Gaussien** (Gaussian radial basis function ou RBF) (sur \mathbb{R}^d)

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right).$$

4. **Laplace** (sur \mathbb{R}) : $K(x, x') = \exp(-\gamma|x - x'|)$.
5. Noyau **min** (sur \mathbb{R}^+) : $K(x, x') = \min(x, x')$.

Exemples de noyau

1. **Linéaire** (sur \mathbb{R}^d) : $K(x, x') = x^t x'$.
2. **Polynomial** (sur \mathbb{R}^d) : $K(x, x') = (x^t x' + 1)^d$.
3. **Gaussien** (Gaussian radial basis function ou RBF) (sur \mathbb{R}^d)

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right).$$

4. **Laplace** (sur \mathbb{R}) : $K(x, x') = \exp(-\gamma|x - x'|)$.
5. Noyau **min** (sur \mathbb{R}^+) : $K(x, x') = \min(x, x')$.

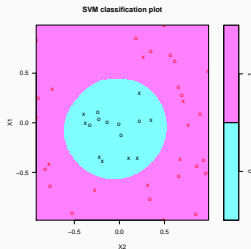
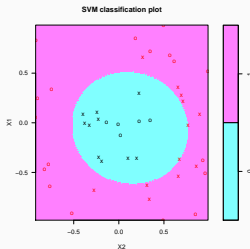
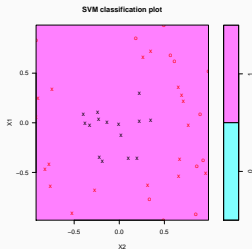
Remarque

N'importe quelle **fonction définie positive** fait l'affaire... Possibilité de construire des noyaux (et donc de faire des svm) sur des **objets plus complexes** (courbes, images, séquences de lettres...).

Le coin R - exemple 1

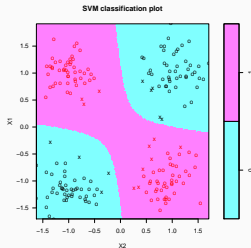
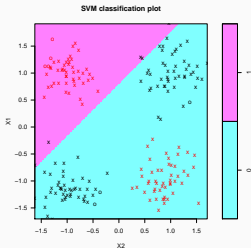
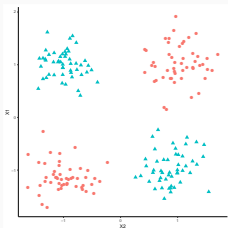
- Argument **kernel** dans la fonction **svm**.

```
> svm(Y~.,data=donnees,cost=1,kernel="linear")  
> svm(Y~.,data=donnees,cost=1,kernel="polynomial",degree=2)  
> svm(Y~.,data=donnees,cost=1,kernel="radial",gamma=1)
```



Le coin R - exemple 2

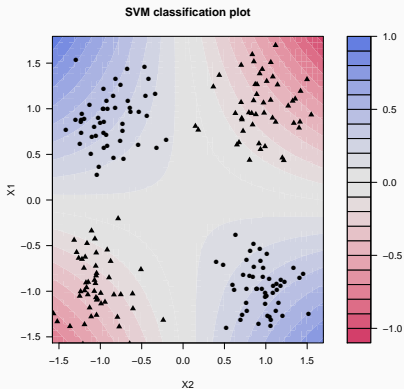
```
> svm(Y~.,data=donnees,kernel="linear",cost=1)
> svm(Y~.,data=donnees,kernel="polynomial",degree=2,cost=1)
```



Le package kernlab

- Il propose un choix plus large de noyaux.

```
> library(kernlab)
> mod.ksvm <- ksvm(Y~.,data=donnees,kernel="polydot",
+                  kpar=list(degree=2),C=0.001)
> plot(mod.ksvm)
```



Troisième partie III

Arbres

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

- Les arbres sont des algorithmes de prédiction qui fonctionnent en régression et en discrimination.
- Il existe différentes variantes permettant de construire des prédicteurs par arbres.
- Nous nous focalisons dans cette partie sur la méthode CART [Breiman et al., 1984] qui est la plus utilisée. La méthode CHAID est proposée en annexe.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Élagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

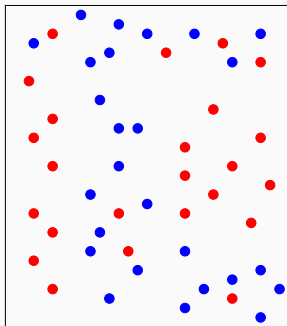
- On cherche à expliquer une variable Y par p variables explicatives X_1, \dots, X_p .

- On cherche à expliquer une variable Y par p variables explicatives X_1, \dots, X_p .
- Y peut admettre un nombre quelconque de modalités et les variables X_1, \dots, X_p peuvent être qualitatives et/ou quantitatives.

- On cherche à expliquer une variable Y par p variables explicatives X_1, \dots, X_p .
- Y peut admettre un nombre quelconque de modalités et les variables X_1, \dots, X_p peuvent être qualitatives et/ou quantitatives.
- Néanmoins, pour simplifier on se place dans un premier temps en discrimination binaire : Y admet 2 modalités (-1 ou 1). On suppose de plus que l'on a simplement 2 variables explicatives quantitatives.

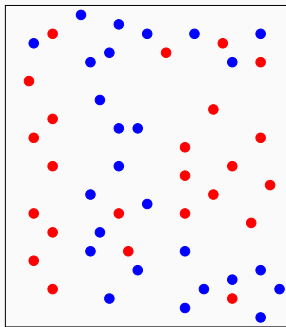
Représentation des données

- On dispose de n observations $(X_1, Y_1), \dots, (X_n, Y_n)$ où $X_i \in \mathbb{R}^2$ et $Y_i \in \{-1, 1\}$.



Représentation des données

- On dispose de n observations $(X_1, Y_1), \dots, (X_n, Y_n)$ où $X_i \in \mathbb{R}^2$ et $Y_i \in \{-1, 1\}$.



Approche par arbres

Trouver une **partition** des observations qui **sépare** "au mieux" les points rouges des points bleus.

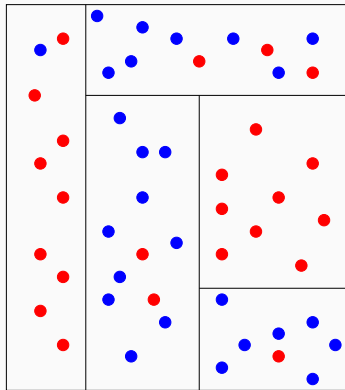
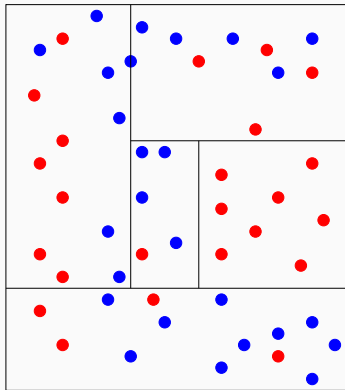
Arbre binaire

Un **arbre binaire de décision** CART est

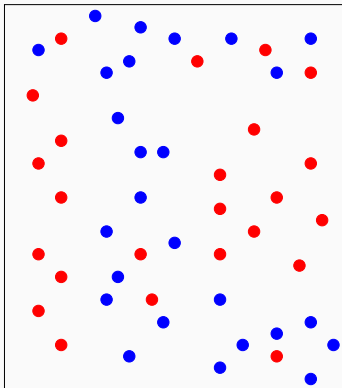
- un algorithme de **moyennage local** par partition (moyenne ou vote à la majorité sur les éléments de la partition),
- dont la partition est construite par **divisions successives** au moyen d'**hyperplans orthogonaux aux axes** de \mathbb{R}^P , dépendant des données (X_i, Y_i) .

Arbres binaires

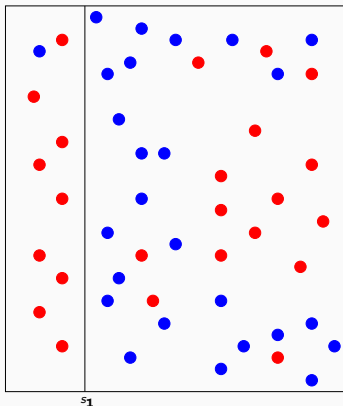
- La **méthode CART** propose de construire une partition basée sur des divisions **successives parallèles aux axes**.
- 2 exemples de partition :



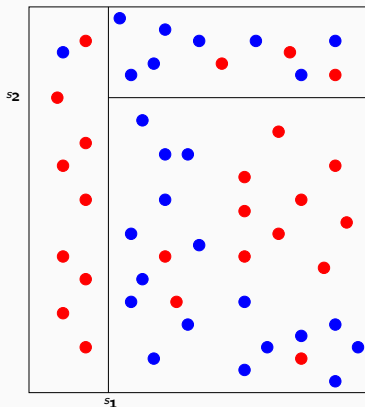
- A chaque étape, la méthode cherche une **nouvelle division** : une **variable** et un **seuil** de coupure.



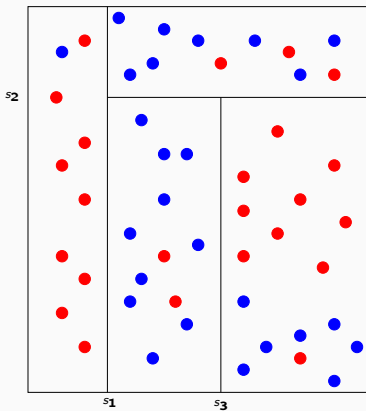
- A chaque étape, la méthode cherche une **nouvelle division** : une **variable** et un **seuil** de coupure.



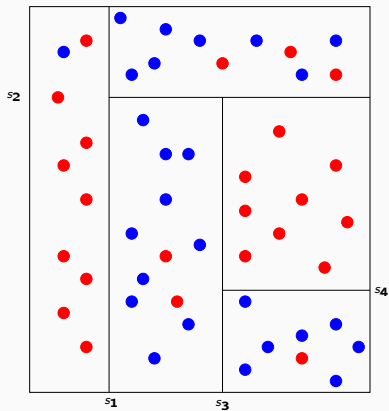
- A chaque étape, la méthode cherche une **nouvelle division** : une **variable** et un **seuil** de coupure.



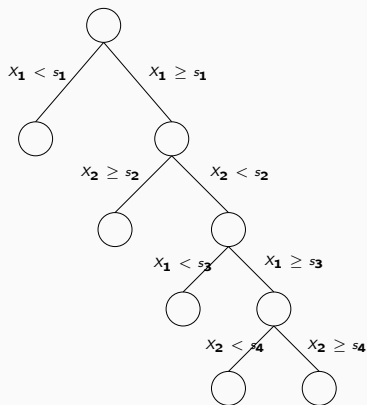
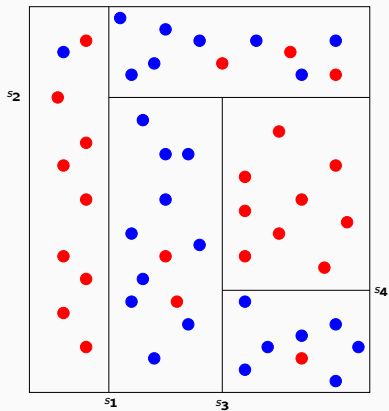
- A chaque étape, la méthode cherche une **nouvelle division** : une **variable** et un **seuil** de coupure.



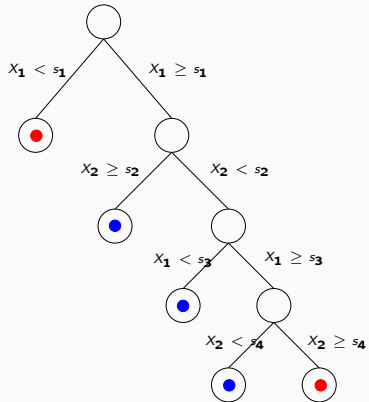
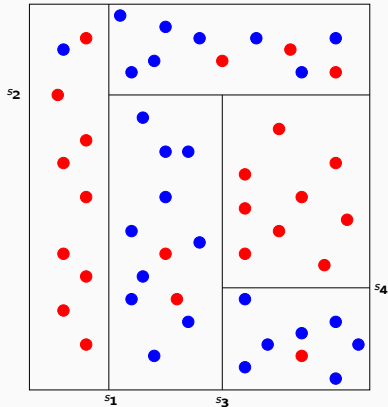
- A chaque étape, la méthode cherche une **nouvelle division** : une **variable** et un **seuil** de coupure.



Représentation de l'arbre



Représentation de l'arbre



Règle de classification

On effectue un **vote à la majorité** dans les nœuds terminaux de l'arbre.

Définition

- Les éléments de la partition d'un arbre sont appelés les **nœuds terminaux** ou les **feuilles** de l'arbre.
- L'ensemble \mathbb{R}^p constitue le **nœud racine**.
- Chaque division définit deux nœuds, les **nœuds fils à gauche et à droite**.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

Questions

1. Comment choisir les découpes ?
2. Faut-il stopper les découpes ? Si oui, quand ?

Questions

1. Comment choisir les découpes ?
 2. Faut-il stopper les découpes ? Si oui, quand ?
- A chaque étape, on cherche un couple (j, s) qui split un noeud \mathcal{N} en deux nœuds fils :
- $$\mathcal{N}_1(j, s) = \{X \in \mathcal{N} | X_j \leq s\} \quad \text{et} \quad \mathcal{N}_2(j, s) = \{X \in \mathcal{N} | X_j > s\}.$$
- La sélection du couple (j, s) s'effectue en optimisant un critère qui mesure l'(im)pureté ou l'hétérogénéité des deux nœuds fils.

- L'impureté \mathcal{I} d'un nœud doit être :
 1. faible lorsque un nœud est homogène : les valeurs de Y dans le nœud sont proches.
 2. élevée lorsque un nœud est hétérogène : les valeurs de Y dans le nœud sont dispersées.

Critère de découpe

- L'**impureté** \mathcal{I} d'un nœud doit être :
 1. **faible** lorsque un nœud est homogène : les valeurs de Y dans le nœud sont **proches**.
 2. **élevée** lorsque un nœud est hétérogène : les valeurs de Y dans le nœud sont **dispersées**.

L'idée

Une fois \mathcal{I} définie, on choisira le couple (j, s) qui **maximise le gain d'impureté** :

$$\Delta(\mathcal{I}) = \mathbf{P}(\mathcal{N})\mathcal{I}(\mathcal{N}) - (\mathbf{P}(\mathcal{N}_1)\mathcal{I}(\mathcal{N}_1(j, s)) + \mathbf{P}(\mathcal{N}_2)\mathcal{I}(\mathcal{N}_2(j, s)))$$

où $\mathbf{P}(\mathcal{N})$ représente la proportion d'observations dans le nœud \mathcal{N} .

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

- Une mesure naturelle de l'impureté d'un nœud \mathcal{N} en régression est la variance du nœud :

$$\mathcal{I}(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i: X_i \in \mathcal{N}} (Y_i - \bar{Y}_{\mathcal{N}})^2,$$

où $\bar{Y}_{\mathcal{N}}$ désigne la moyenne des Y_i dans \mathcal{N} .

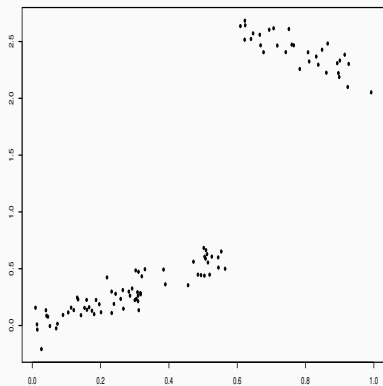
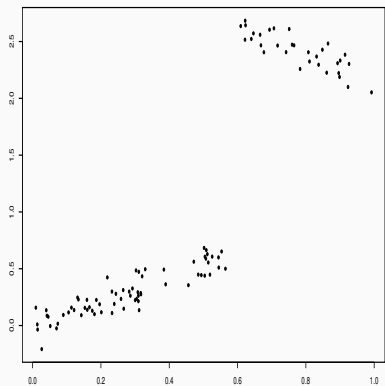
Découpe en régression

A chaque étape, on choisit le couple (j, s) qui minimise

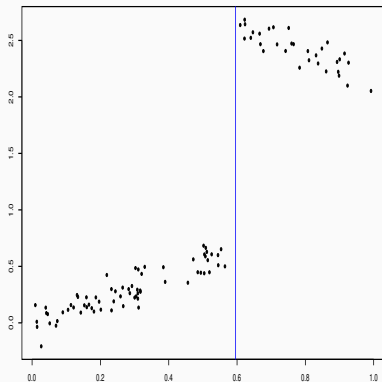
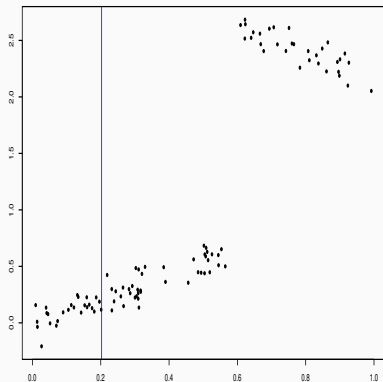
$$\sum_{X_i \in \mathcal{N}_1(j,s)} (Y_i - \bar{Y}_1)^2 + \sum_{X_i \in \mathcal{N}_2(j,s)} (Y_i - \bar{Y}_2)^2$$

où $\bar{Y}_k = \frac{1}{|\mathcal{N}_k(j,s)|} \sum_{X_i \in \mathcal{N}_k(j,s)} Y_i, k = 1, 2.$

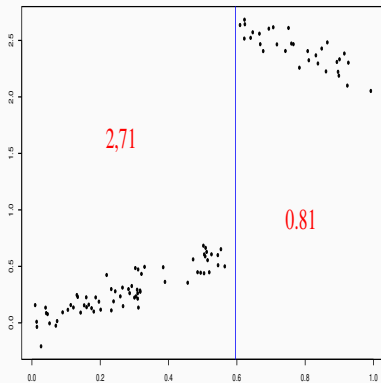
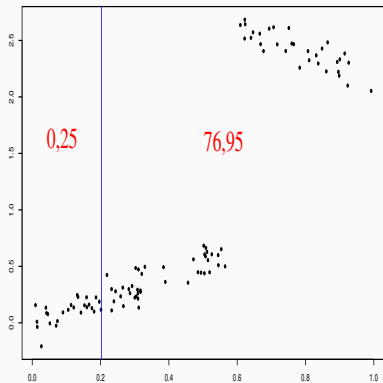
Example



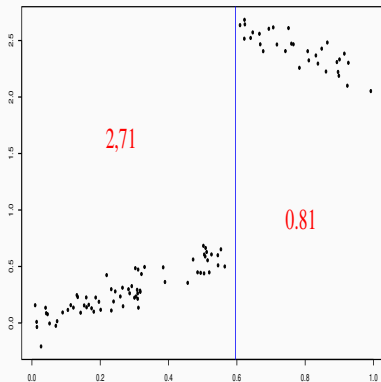
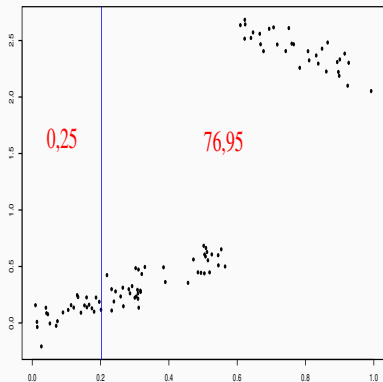
Example



Exemple



Exemple



Sélection

On choisira le seuil de droite.

Arbres binaires

Choix des découpes

Cas de la régression

Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

Regroupement des modalités

Division d'un nœud

Choix des paramètres

Bibliographie

- On se place ici dans le cas **binaire**, Y dans $\{0, 1\}$ (voir Annexe pour le cas multiclasse).
- Un nœud est **pur** si
 - il contient beaucoup de 0 et peu de 1 (ou l'inverse) ;
 - la proportion de 1 est proche de 1 (ou de 0).

- On se place ici dans le cas **binaire**, Y dans $\{0, 1\}$ (voir Annexe pour le cas multiclasse).
- Un nœud est **pur** si
 - il contient beaucoup de 0 et peu de 1 (ou l'inverse) ;
 - la proportion de 1 est proche de 1 (ou de 0).

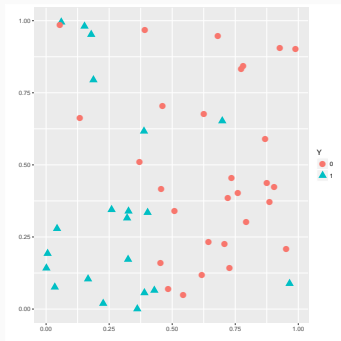
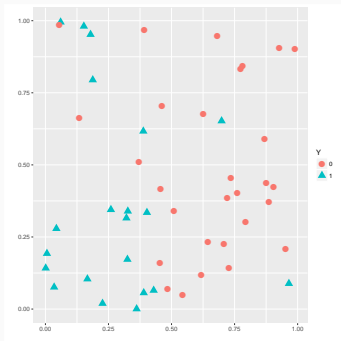
Impureté de Gini

$$\mathcal{I}(\mathcal{N}) = 2p(\mathcal{N})(1 - p(\mathcal{N}))$$

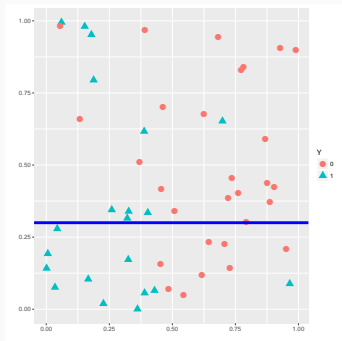
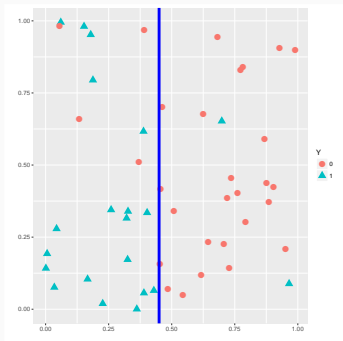
où $p(\mathcal{N})$ représente la proportion de 1 dans \mathcal{N} .

Exemple

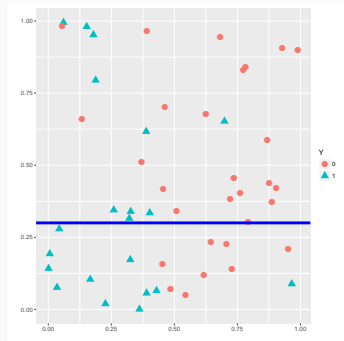
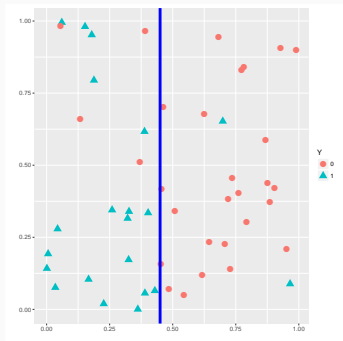
$$\mathcal{I}(\mathcal{N}) = 0.4872$$



Exemple

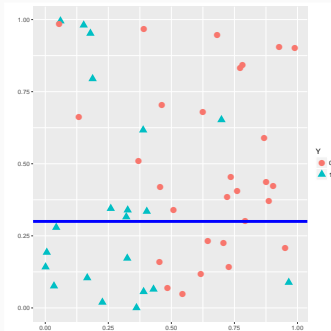
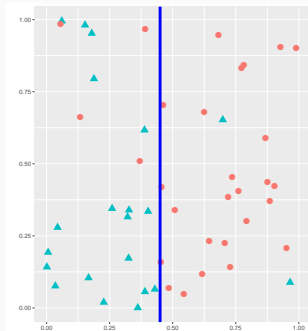


Exemple



	$\mathcal{I}(\mathcal{N}_1)$	$\mathcal{I}(\mathcal{N}_2)$	$\Delta(\mathcal{I})$
Gauche	0.287	0.137	0.281
Droite	0.488	0.437	0.031

Exemple



	$\mathcal{I}(\mathcal{N}_1)$	$\mathcal{I}(\mathcal{N}_2)$	$\Delta(\mathcal{I})$
Gauche	0.287	0.137	0.281
Droite	0.488	0.437	0.031

Conclusion

On choisira la découpe de **gauche**.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

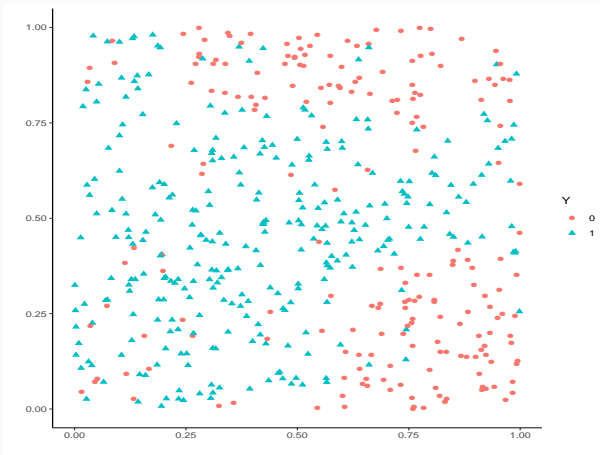
- Comment construire un "bon" arbre ?

- Comment construire un "bon" arbre ?
- Construire l'arbre maximal ? (on découpe les nœuds jusqu'à ce qu'on ne puisse plus).

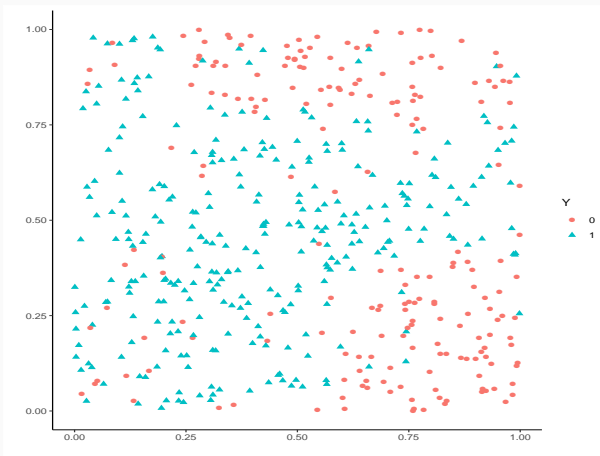
- Comment construire un "bon" arbre ?
- Construire l'arbre maximal ? (on découpe les nœuds jusqu'à ce qu'on ne puisse plus).
- Faut-il se donner un critère d'arrêt ?

- Comment construire un "bon" arbre ?
- Construire l'arbre maximal ? (on découpe les nœuds jusqu'à ce qu'on ne puisse plus).
- Faut-il se donner un critère d'arrêt ?
- Faut-il construire un arbre grand et choisir un sous-arbre de ce dernier ?

Un exemple en discrimination



Un exemple en discrimination

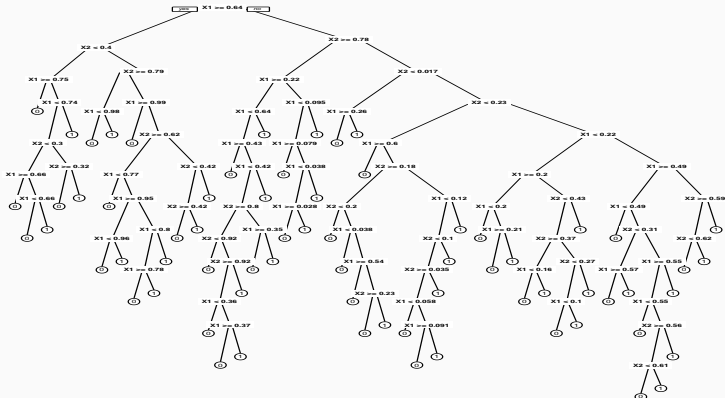


Arbre optimal ?

Intuitivement, on a envie de faire à peu près 5 classes.

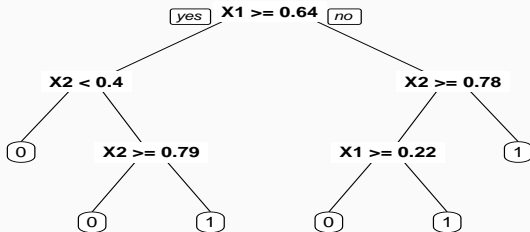
Arbre « maximal »

```
> library(rpart)
> library(rpart.plot)
> arbre1 <- rpart(Y~.,data=donnees[1:350,],cp=0.0001,minsplitlevel=2)
> prp(arbre1)
```



Un arbre plus petit

```
> arbre2 <- rpart(Y~.,data=donnees[1:350,])  
> prp(arbre2)
```



Comparaison des deux arbres

- On compare les performances des deux arbres en estimant leur **probabilité de mauvais classement** sur un échantillon test :

```
> prev <- data.frame(arbre1=predict(arbre1,newdata=donnees[351:500,],type="class"),
+                   arbre2=predict(arbre2,newdata=donnees[351:500,],type="class"),
+                   obs=donnees[351:500,]$Y)
> prev %>% summarize_at(1:2,funs(mean(.!=obs))) %>% round(3)
##   arbre1 arbre2
## 1  0.133  0.127
```

Comparaison des deux arbres

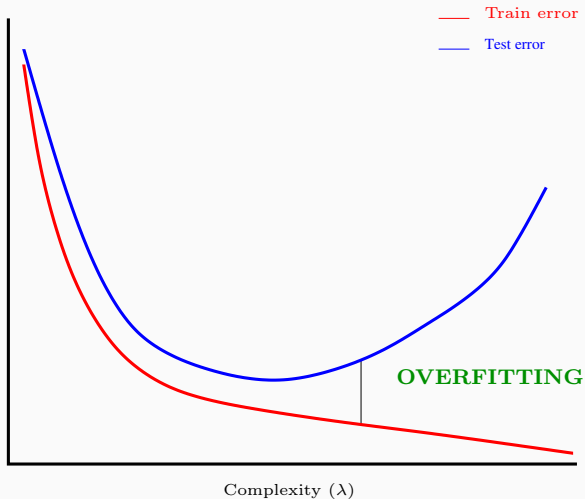
- On compare les performances des deux arbres en estimant leur **probabilité de mauvais classement** sur un échantillon test :

```
> prev <- data.frame(arbre1=predict(arbre1,newdata=donnees[351:500,],type="class"),
+                   arbre2=predict(arbre2,newdata=donnees[351:500,],type="class"),
+                   obs=donnees[351:500,]$Y)
> prev %>% summarize_at(1:2,funs(mean(.!=obs))) %>% round(3)
##   arbre1 arbre2
## 1  0.133  0.127
```

Conclusion

La performance **n'augmente pas forcément** avec la profondeur.

Sur-ajustement pour les arbres



Remarque

La complexité d'un arbre est mesurée par sa taille ou profondeur.

Biais et variance

La **profondeur** régule le compromis biais/variance :

1. **Peu de découpes** (arbres peu profonds) \implies arbres stables \implies **peu de variance**... mais... **beaucoup de biais**.
2. **Beaucoup de découpes** (arbres profonds) \implies arbres instables \implies **peu de biais**... mais... **beaucoup de variance (surapprentissage)**.

Biais et variance

La **profondeur** régule le compromis biais/variance :

1. **Peu de découpes** (arbres peu profonds) \implies arbres stables \implies **peu de variance**... mais... **beaucoup de biais**.
2. **Beaucoup de découpes** (arbres profonds) \implies arbres instables \implies **peu de biais**... mais... **beaucoup de variance** (surapprentissage).

Principe d'élagage [Breiman et al., 1984]

Plutôt que de choisir « quand couper » on raisonne en 3 temps :

1. On construit un **arbre maximal** (très profond) \mathcal{T}_{max} ;
2. On sélectionne une **suite d'arbres emboîtés** :

$$\mathcal{T}_{max} = \mathcal{T}_0 \supset \mathcal{T}_1 \supset \dots \supset \mathcal{T}_K.$$

3. On **sélectionne un arbre** dans cette sous-suite.

- La construction de la suite de sous-arbres emboîtées est détaillée en Annexe.
- Sur **R**, on obtient cette sous-suite à l'aide de la fonction **printcp** :

```
> arbre <- rpart(Y~.,data=donnees,cp=0.0001,minsplit=2)
> printcp(arbre)
##
## Classification tree:
## rpart(formula = Y ~ ., data = donnees, cp = 1e-04, minsplit = 2)
##
## Variables actually used in tree construction:
## [1] X1 X2
##
## Root node error: 204/500 = 0.408
##
## n= 500
##
##          CP nsplit rel error  xerror   xstd
## 1  0.2941176      0  1.000000 1.00000 0.053870
## 2  0.1225490      1  0.705882 0.72059 0.049938
## 3  0.0931373      3  0.460784 0.51471 0.044646
## 4  0.0637255      4  0.367647 0.42647 0.041555
## 5  0.0122549      5  0.303922 0.35294 0.038483
## 6  0.0098039      7  0.279412 0.35294 0.038483
## 7  0.0049020      9  0.259804 0.35784 0.038704
## 8  0.0040107     25  0.181373 0.39216 0.040184
## 9  0.0036765     41  0.112745 0.39706 0.040386
## 10 0.0032680     49  0.083333 0.40196 0.040586
## 11 0.0024510     52  0.073529 0.41667 0.041174
## 12 0.0001000     82  0.000000 0.45098 0.042473
```

Sorties printcp

- Suite de 12 arbres emboîtés.
- CP : complexity parameter, il mesure la complexité de l'arbre : CP $\searrow \implies$ complexité \nearrow .
- nsplit : nombre de coupures de l'arbre.
- rel.error : erreur (normalisée) calculée sur les données d'apprentissage \implies erreur d'ajustement.
- xerror : erreur (normalisée) calculée par validation croisée 10 blocs \implies erreur de prévision.
- xstd : écart-type associé à l'erreur de validation croisée.

Sorties printcp

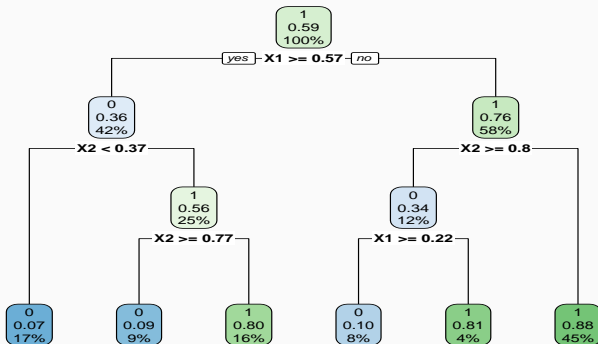
- Suite de 12 arbres emboîtés.
- CP : complexity parameter, il mesure la complexité de l'arbre : CP $\searrow \Rightarrow$ complexité \nearrow .
- nsplit : nombre de coupures de l'arbre.
- rel.error : erreur (normalisée) calculée sur les données d'apprentissage \Rightarrow erreur d'ajustement.
- xerror : erreur (normalisée) calculée par validation croisée 10 blocs \Rightarrow erreur de prévision.
- xstd : écart-type associé à l'erreur de validation croisée.

Choix de l'arbre final.

On choisira l'arbre qui a la plus petite erreur de prévision (calculée par validation croisée).

Tracé de l'arbre final

```
> cp_opt <- arbre$sptable %>% as.data.frame() %>%  
+   filter(xerror==min(xerror)) %>% dplyr::select(CP) %>%  
+   slice(1) %>% as.numeric()  
> cp_opt  
## [1] 0.0122549  
> arbre_final <- prune(arbre,cp = cp_opt)  
> rpart.plot(arbre_final)
```



- L'arbre final \mathcal{T} renvoie une **partition** de \mathbb{R}^p en $|\mathcal{T}|$ nœuds terminaux $\mathcal{N}_1, \dots, \mathcal{N}_{|\mathcal{T}|}$.

Règle de classification et score par arbre

- L'arbre final \mathcal{T} renvoie une **partition** de \mathbb{R}^p en $|\mathcal{T}|$ nœuds terminaux $\mathcal{N}_1, \dots, \mathcal{N}_{|\mathcal{T}|}$.
- **Règle de classification** :

$$\hat{g}(x) = \begin{cases} 1 & \text{si } \sum_{i: X_i \in \mathcal{N}(x)} \mathbf{1}_{Y_i=1} \geq \sum_{i: X_i \in \mathcal{N}(x)} \mathbf{1}_{Y_i=0} \\ 0 & \text{sinon,} \end{cases}$$

où $\mathcal{N}(x)$ désigne le nœud terminal qui contient x .

Règle de classification et score par arbre

- L'arbre final \mathcal{T} renvoie une **partition** de \mathbb{R}^p en $|\mathcal{T}|$ nœuds terminaux $\mathcal{N}_1, \dots, \mathcal{N}_{|\mathcal{T}|}$.
- **Règle de classification** :

$$\hat{g}(x) = \begin{cases} 1 & \text{si } \sum_{i: X_i \in \mathcal{N}(x)} \mathbf{1}_{Y_i=1} \geq \sum_{i: X_i \in \mathcal{N}(x)} \mathbf{1}_{Y_i=0} \\ 0 & \text{sinon,} \end{cases}$$

où $\mathcal{N}(x)$ désigne le nœud terminal qui contient x .

- **Score** :

$$\hat{S}(x) = \hat{\mathbf{P}}(Y = 1 | X = x) = \frac{1}{n} \sum_{i: X_i \in \mathcal{N}(x)} \mathbf{1}_{Y_i=1}.$$

Fonction predict

- La fonction `predict` (`predict.rpart`) permet d'estimer la classe ou le score :

```
> x_new <- data.frame(X1=0.5,X2=0.85)
> predict(arbre_final,newdata=x_new)
##      0      1
## 1 0.9 0.1
> predict(arbre_final,newdata=x_new,type="class")
## 1
## 0
## Levels: 0 1
```

- Méthode « simple » relativement facile à mettre en œuvre.
- Fonctionne en régression et en discrimination.
- Résultats interprétables (à condition que l'arbre ne soit pas trop profond).

- Méthode « simple » relativement facile à mettre en œuvre.
- Fonctionne en **régression** et en **discrimination**.
- Résultats **interprétables** (à condition que l'arbre ne soit pas trop profond).
- **Un inconvénient** : méthode connue pour être **instable**, sensible à de légères perturbations de l'échantillon.

- Méthode « simple » relativement facile à mettre en œuvre.
- Fonctionne en régression et en discrimination.
- Résultats interprétables (à condition que l'arbre ne soit pas trop profond).
- Un inconvénient : méthode connue pour être instable, sensible à de légères perturbations de l'échantillon.
- Cet inconvénient sera un avantage pour des agrégations bootstrap \Rightarrow forêts aléatoires.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Élagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

- Les $Y_i, i = 1, \dots, n$ sont à valeurs dans $\{1, \dots, K\}$.

- Les $Y_i, i = 1, \dots, n$ sont à valeurs dans $\{1, \dots, K\}$.
- On cherche une fonction \mathcal{I} telle que $\mathcal{I}(\mathcal{N})$ soit
 - petite si un label majoritaire se distingue clairement dans \mathcal{N} ;
 - grande sinon.

- Les $Y_i, i = 1, \dots, n$ sont à valeurs dans $\{1, \dots, K\}$.
- On cherche une fonction \mathcal{I} telle que $\mathcal{I}(\mathcal{N})$ soit
 - **petite** si un **label majoritaire** se distingue clairement dans \mathcal{N} ;
 - **grande** sinon.

Impureté

L'**impureté** d'un nœud \mathcal{N} en classification se mesure selon

$$\mathcal{I}(\mathcal{N}) = \sum_{j=1}^K f(p_j(\mathcal{N}))$$

où

- $p_j(\mathcal{N})$ représente la proportion d'observations de la classe j dans le nœud \mathcal{N} .
- f est une fonction (concave) $[0, 1] \rightarrow \mathbb{R}^+$ telle que $f(0) = f(1) = 0$.

Exemples de fonctions f

- Si \mathcal{N} est pur, on veut $\mathcal{I}(\mathcal{N}) = 0$

Exemples de fonctions f

- Si \mathcal{N} est pur, on veut $\mathcal{I}(\mathcal{N}) = 0 \implies$ c'est pourquoi f doit vérifier $f(0) = f(1) = 0$.

Exemples de fonctions f

- Si \mathcal{N} est pur, on veut $\mathcal{I}(\mathcal{N}) = 0 \implies$ c'est pourquoi f doit vérifier $f(0) = f(1) = 0$.
- Les 2 mesures d'impureté les plus classiques sont :
 1. Gini : $f(p) = p(1 - p)$;
 2. Information : $f(p) = -p \log(p)$.

Exemples de fonctions f

- Si \mathcal{N} est pur, on veut $\mathcal{I}(\mathcal{N}) = 0 \implies$ c'est pourquoi f doit vérifier $f(0) = f(1) = 0$.
- Les 2 mesures d'impureté les plus classiques sont :
 1. Gini : $f(p) = p(1 - p)$;
 2. Information : $f(p) = -p \log(p)$.

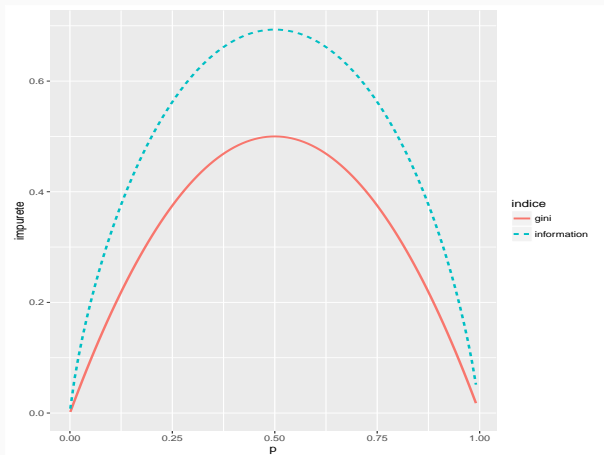
Cas binaire

Dans ce cas on a

1. $\mathcal{I}(\mathcal{N}) = 2p(1 - p)$ pour Gini
2. $\mathcal{I}(\mathcal{N}) = -p \log p - (1 - p) \log(1 - p)$ pour Information

où p désigne la proportion de 1 (ou -1) dans \mathcal{N} .

Impureté dans le cas binaire



- On rappelle que pour un nœud \mathcal{N} donné et un couple (j, s) , on note

$$\mathcal{N}_1(j, s) = \{X \in \mathcal{N} | X_j \leq s\} \quad \text{et} \quad \mathcal{N}_2(j, s) = \{X \in \mathcal{N} | X_j > s\}.$$

- On rappelle que pour un nœud \mathcal{N} donné et un couple (j, s) , on note

$$\mathcal{N}_1(j, s) = \{X \in \mathcal{N} | X_j \leq s\} \quad \text{et} \quad \mathcal{N}_2(j, s) = \{X \in \mathcal{N} | X_j > s\}.$$

Choix de (j, s)

Pour une mesure d'impureté \mathcal{I} donnée, on choisira le couple (j, s) qui maximise le gain d'impureté :

$$\Delta(\mathcal{I}) = \mathbf{P}(\mathcal{N})\mathcal{I}(\mathcal{N}) - (\mathbf{P}(\mathcal{N}_1)\mathcal{I}(\mathcal{N}_1(j, s)) + \mathbf{P}(\mathcal{N}_2)\mathcal{I}(\mathcal{N}_2(j, s))).$$

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Élagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

Construction de la suite de sous arbres

- Soit T un arbre à $|T|$ nœuds terminaux $\mathcal{N}_1, \dots, \mathcal{N}_{|T|}$.
- Soit $R(\mathcal{N})$ le risque (l'erreur) dans le nœud \mathcal{N} :

- Régression :

$$R(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i: X_i \in \mathcal{N}} (Y_i - \bar{Y}_{\mathcal{N}})^2.$$

- Classification binaire :

$$R(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i: X_i \in \mathcal{N}} \mathbf{1}_{Y_i \neq Y_{\mathcal{N}}}.$$

Construction de la suite de sous arbres

- Soit T un arbre à $|T|$ nœuds terminaux $\mathcal{N}_1, \dots, \mathcal{N}_{|T|}$.
- Soit $R(\mathcal{N})$ le risque (l'erreur) dans le nœud \mathcal{N} :
 - Régression :

$$R(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i: X_i \in \mathcal{N}} (Y_i - \bar{Y}_{\mathcal{N}})^2.$$

- Classification binaire :

$$R(\mathcal{N}) = \frac{1}{|\mathcal{N}|} \sum_{i: X_i \in \mathcal{N}} \mathbf{1}_{Y_i \neq Y_{\mathcal{N}}}.$$

Définition

Soit $\alpha > 0$, on pose

$$C_{\alpha}(T) = \sum_{m=1}^{|T|} N_m R(\mathcal{N}_m) + \alpha |T|.$$

Idée

- $C_\alpha(T)$ est un critère qui prend en compte l'adéquation d'un arbre et sa complexité.
- L'idée est de chercher un arbre T_α qui minimise $C_\alpha(T)$ pour une valeur de α bien choisie.

Idée

- $C_\alpha(T)$ est un critère qui prend en compte l'adéquation d'un arbre et sa complexité.
- L'idée est de chercher un arbre T_α qui minimise $C_\alpha(T)$ pour une valeur de α bien choisie.

Remarque

- $\alpha = 0 \implies T_\alpha = T_0 = T_{max}$.
- $\alpha = +\infty \implies T_\alpha = T_{+\infty} = \text{arbre sans coupure}$.

Idée

- $C_\alpha(T)$ est un critère qui prend en compte l'adéquation d'un arbre et sa complexité.
- L'idée est de chercher un arbre T_α qui minimise $C_\alpha(T)$ pour une valeur de α bien choisie.

Remarque

- $\alpha = 0 \implies T_\alpha = T_0 = T_{max}$.
- $\alpha = +\infty \implies T_\alpha = T_{+\infty} = \text{arbre sans coupure}$.
- α est appelé paramètre de complexité et $C_\alpha(T)$ le cout de l'arbre T .

Théorème ([Breiman et al., 1984])

*Il existe une sous-suite finie $\alpha_0 = 0 < \alpha_1 < \dots < \alpha_M$ avec $M < |T_{\max}|$ et une suite associée d'**arbres emboîtés***

$$T_{\max} = T_0 \supset T_1 \supset \dots \supset T_M$$

telles que $\forall \alpha \in [\alpha_m, \alpha_{m+1}[$

$$T_m = \operatorname{argmin}_T C_\alpha(T).$$

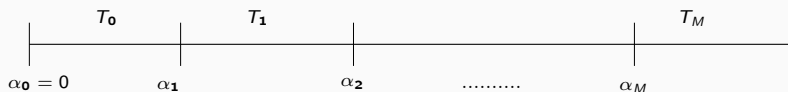
Théorème ([Breiman et al., 1984])

Il existe une sous-suite finie $\alpha_0 = 0 < \alpha_1 < \dots < \alpha_M$ avec $M < |T_{\max}|$ et une suite associée d'*arbres emboîtés*

$$T_{\max} = T_0 \supset T_1 \supset \dots \supset T_M$$

telles que $\forall \alpha \in [\alpha_m, \alpha_{m+1}[$

$$T_m = \underset{T}{\operatorname{argmin}} C_\alpha(T).$$



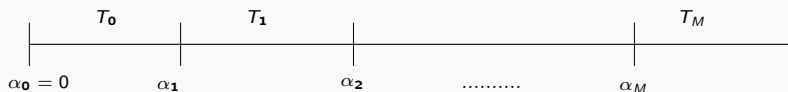
Théorème ([Breiman et al., 1984])

Il existe une sous-suite finie $\alpha_0 = 0 < \alpha_1 < \dots < \alpha_M$ avec $M < |T_{\max}|$ et une suite associée d'**arbres emboîtés**

$$T_{\max} = T_0 \supset T_1 \supset \dots \supset T_M$$

telles que $\forall \alpha \in [\alpha_m, \alpha_{m+1}[$

$$T_m = \operatorname{argmin}_T C_\alpha(T).$$



Conséquences

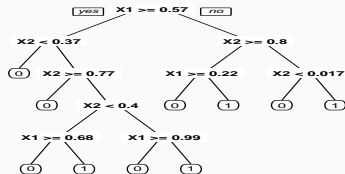
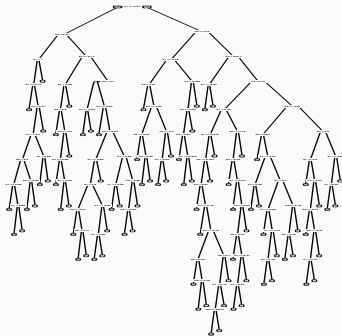
- On se ramène à une **sous-suite finie** d'arbres (emboîtés).
- Il reste à choisir un arbre (ou **une valeur de α**).

Exemple

```
> printcp(arbre)
Classification tree:
rpart(formula = Y ~ ., data = donnees, cp = 1e-04, minsplit = 2)
Variables actually used in tree construction:
[1] X1 X2
Root node error: 204/500 = 0.408
n= 500
```

	CP	nsplit	rel error	xerror	xstd
1	0.2941176	0	1.000000	1.00000	0.053870
2	0.1225490	1	0.705882	0.71569	0.049838
3	0.0931373	3	0.460784	0.49020	0.043844
4	0.0637255	4	0.367647	0.43627	0.041928
5	0.0122549	5	0.303922	0.34314	0.038034
6	0.0098039	7	0.279412	0.34314	0.038034
7	0.0049020	9	0.259804	0.36275	0.038923
8	0.0040107	25	0.181373	0.34804	0.038260
9	0.0036765	41	0.112745	0.39216	0.040184
10	0.0032680	49	0.083333	0.40196	0.040586
11	0.0024510	52	0.073529	0.41176	0.040980
12	0.0001000	82	0.000000	0.43137	0.041742

```
> arbre1 <- prune(arbre, cp=0.005)
> prp(arbre)
> prp(arbre1)
```



Choix d'un arbre

Il reste à sélectionner un arbre dans la suite

$$T_{max} = T_0 \supset T_1 \supset \dots \supset T_M$$

Sélection d'un arbre

Choix d'un risque

La sélection de l'**arbre final** s'effectue en choisissant l'élément de la suite qui minimise le risque moyen $\mathbf{E}[R(Y, T_m(X))]$. Par exemple,

1. l'**erreur quadratique** $\mathbf{E}[(Y - T_m(X))^2]$ en **régression** ;
2. la **probabilité d'erreur** $\mathbf{P}(Y \neq T_m(X))$ en **discrimination binaire**.

Ce risque (inconnu) est estimé par **validation croisée**.

Sélection d'un arbre

Choix d'un risque

La sélection de l'**arbre final** s'effectue en choisissant l'élément de la suite qui minimise le risque moyen $\mathbf{E}[R(Y, T_m(X))]$. Par exemple,

1. l'**erreur quadratique** $\mathbf{E}[(Y - T_m(X))^2]$ en **régression** ;
2. la **probabilité d'erreur** $\mathbf{P}(Y \neq T_m(X))$ en **discrimination binaire**.

Ce risque (inconnu) est estimé par **validation croisée**.

Choix de l'arbre final

L'approche consiste à

1. **estimer le risque** pour chaque α_m .
2. choisir le α_m qui **minimise le risque estimé** $\implies T_{\alpha_m}$.

Algorithme

1. Calculer la suite $\alpha_0 = 0 < \alpha_1 < \dots < \alpha_M$ et poser

$$\beta_1 = 0, \quad \beta_2 = \sqrt{\alpha_1 \alpha_2}, \quad \beta_3 = \sqrt{\alpha_2 \alpha_3}, \quad \dots, \quad \beta_{M+1} = \infty.$$

2. **Séparer les données** en K blocs G_1, \dots, G_k de taille k/n . Pour $i = 1, \dots, k$:

- 2.1 Construire les arbres $T_{\beta_1}, \dots, T_{\beta_{M+1}}$ sur l'ensemble des observations **privé du i ème bloc**.

- 2.2 En déduire pour tout $j \in G_i$ et tout $m \leq M+1$, $\hat{Y}_j(\beta_m) = T_{\beta_m}(X_j)$.

3. Calculer $\mathcal{R}(m) = \frac{1}{n} \sum_{i=1}^n R(Y_i, \hat{Y}_i(\beta_m))$ pour $m = 1, \dots, M+1$.

4. **Choisir α_{m^*}** tel que $\beta_{m^*+1} = \operatorname{argmin}_{m \leq M+1} \mathcal{R}(m)$.

- Les estimations $\mathcal{R}(m)$ se trouvent dans la colonne **xerror** de la fonction **printcp** :

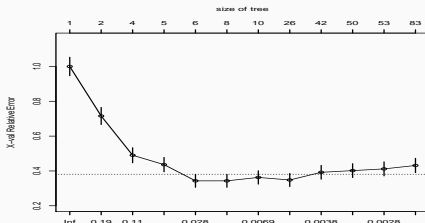
	CP	nsplit	rel error	xerror	xstd
1	0.2941176	0	1.000000	1.00000	0.053870
2	0.1225490	1	0.705882	0.71569	0.049838
3	0.0931373	3	0.460784	0.49020	0.043844
4	0.0637255	4	0.367647	0.43627	0.041928
5	0.0122549	5	0.303922	0.34314	0.038034
6	0.0098039	7	0.279412	0.34314	0.038034
7	0.0049020	9	0.259804	0.36275	0.038923

- Les estimations $\mathcal{R}(m)$ se trouvent dans la colonne **xerror** de la fonction **printcp** :

	CP	nsplit	rel error	xerror	xstd
1	0.2941176	0	1.000000	1.00000	0.053870
2	0.1225490	1	0.705882	0.71569	0.049838
3	0.0931373	3	0.460784	0.49020	0.043844
4	0.0637255	4	0.367647	0.43627	0.041928
5	0.0122549	5	0.303922	0.34314	0.038034
6	0.0098039	7	0.279412	0.34314	0.038034
7	0.0049020	9	0.259804	0.36275	0.038923

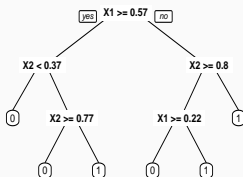
- On peut représenter les erreurs en fonction des α_m à l'aide de **plotcp**

```
> plotcp(arbre3)
```



Tracé de l'arbre final

```
> alpha_opt <- arbre$cptable[which.min(arbre$cptable[, "xerror"]), "CP"]  
> arbre_final <- prune(arbre, cp=alpha_opt)  
> prp(arbre_final)
```



Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

- CHAID : Chi2 Automatic Interaction Detection [[Kass, 1980](#)].

- CHAID : Chi2 Automatic Interaction Detection [Kass, 1980].
- 2 étapes χ^2 dans le procédé de division d'un nœud :
 - regrouper les modalités peu discriminantes de chaque variable explicative X_j ;
 - choisir la variable à utiliser pour scinder le nœud.

χ^2 d'indépendance : rappel

- Soient X et Y deux variables aléatoires à valeurs dans E et F . On souhaite tester au niveau α les hypothèses H_0 : " X et Y sont indépendantes" contre H_1 : " X et Y ne sont pas indépendantes".

χ^2 d'indépendance : rappel

- Soient X et Y deux variables aléatoires à valeurs dans E et F . On souhaite tester au niveau α les hypothèses H_0 : " X et Y sont indépendantes" contre H_1 : " X et Y ne sont pas indépendantes".
- On se donne (E_1, \dots, E_I) et (F_1, \dots, F_J) deux partitions de E et F .

χ^2 d'indépendance : rappel

- Soient X et Y deux variables aléatoires à valeurs dans E et F . On souhaite tester au niveau α les hypothèses H_0 : " X et Y sont indépendantes" contre H_1 : " X et Y ne sont pas indépendantes".
- On se donne (E_1, \dots, E_I) et (F_1, \dots, F_J) deux partitions de E et F .
- On dispose de n mesures du couple (X, Y) et on désigne par N_{ij} l'effectif observé dans la classe $E_i \times F_j$.

χ^2 d'indépendance : rappel

- Soient X et Y deux variables aléatoires à valeurs dans E et F . On souhaite tester au niveau α les hypothèses H_0 : " X et Y sont indépendantes" contre H_1 : " X et Y ne sont pas indépendantes".
- On se donne (E_1, \dots, E_I) et (F_1, \dots, F_J) deux partitions de E et F .
- On dispose de n mesures du couple (X, Y) et on désigne par N_{ij} l'effectif observé dans la classe $E_i \times F_j$.

	F_1	\dots	F_j	\dots	F_J	Total
E_1	N_{11}	\dots	N_{1j}	\dots	N_{1J}	$N_{1\bullet}$
\vdots						\vdots
E_i	N_{i1}	\dots	N_{ij}	\dots	N_{iJ}	$N_{i\bullet}$
\vdots						\vdots
E_I	N_{I1}	\dots	N_{Ij}	\dots	N_{IJ}	$N_{I\bullet}$
Total	$N_{\bullet 1}$	\dots	$N_{\bullet j}$	\dots	$N_{\bullet J}$	n

Propriété

Sous H_0 la statistique

$$X_n = \sum_{i=1}^I \sum_{j=1}^J \frac{\left(\frac{N_{i\bullet} N_{\bullet j}}{n} - N_{ij} \right)^2}{\frac{N_{i\bullet} N_{\bullet j}}{n}}$$

converge en loi vers la loi $\chi^2_{(I-1)(J-1)}$.

Propriété

Sous H_0 la statistique

$$X_n = \sum_{i=1}^I \sum_{j=1}^J \frac{\left(\frac{N_{i\bullet} N_{\bullet j}}{n} - N_{ij} \right)^2}{\frac{N_{i\bullet} N_{\bullet j}}{n}}$$

converge en loi vers la loi $\chi^2_{(I-1)(J-1)}$.

Conséquence

- Au niveau α , on **rejettera l'hypothèse H_0** si X_{obs} est supérieure au quantile d'ordre $1 - \alpha$ de la loi du $\chi^2_{(I-1)(J-1)}$.

Propriété

Sous H_0 la statistique

$$X_n = \sum_{i=1}^I \sum_{j=1}^J \frac{\left(\frac{N_{i\bullet} N_{\bullet j}}{n} - N_{ij} \right)^2}{\frac{N_{i\bullet} N_{\bullet j}}{n}}$$

converge en loi vers la loi $\chi^2_{(I-1)(J-1)}$.

Conséquence

- Au niveau α , on **rejettera l'hypothèse H_0** si X_{obs} est supérieure au quantile d'ordre $1 - \alpha$ de la loi du $\chi^2_{(I-1)(J-1)}$.
- Une **forte valeur de X_{obs}** (ou une **faible valeur de la probabilité critique**) signifiera un **lien fort** entre les deux variables.

Chaid : le principe

- On suppose dans un premier temps que toutes les variables explicatives $X_j, j = 1, \dots, p$ sont qualitatives à M_j modalités.

Division d'un nœud

1. **Regroupement** des modalités peu discriminantes de chaque variable X_j ;
2. **Choix** de la variable X_j la plus **discriminante**
3. Le nœud est alors **divisé** en un nombre de nœuds fils égal au nombre de modalités créées à l'étape 1.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

1. On se place dans un nœud \mathcal{N} et on considère une variable X_j à M_j modalités ;
2. Les observations dans le nœud définissent la table de contingence suivante

	M_1	\dots	M_j
1			
\vdots			
K			

3. $\forall (M_i, M_\ell) \in \{M_1, \dots, M_j\}^2$, on calcule la statistique du χ^2 croisant Y et les modalités $(M_i, M_\ell) \implies \chi^2(M_i, M_\ell)$ et $p(M_i, M_\ell)$ la probabilité critique associée.

1. On se place dans un nœud \mathcal{N} et on considère une variable X_j à M_j modalités ;
2. Les observations dans le nœud définissent la table de contingence suivante

	M_1	\dots	M_j
1			
\vdots			
K			

3. $\forall (M_i, M_\ell) \in \{M_1, \dots, M_j\}^2$, on calcule la statistique du χ^2 croisant Y et les modalités $(M_i, M_\ell) \Rightarrow \chi^2(M_i, M_\ell)$ et $p(M_i, M_\ell)$ la probabilité critique associée.

Remarque

- 2 modalités discriminantes \Rightarrow dépendance forte dans le test avec $Y \Rightarrow$ "Fort rejet" de $H_0 \Rightarrow \chi^2$ élevé ou pc faible ;

1. On se place dans un nœud \mathcal{N} et on considère une variable X_j à M_j modalités ;
2. Les observations dans le nœud définissent la table de contingence suivante

	M_1	\dots	M_j
1			
\vdots			
K			

3. $\forall (M_i, M_\ell) \in \{M_1, \dots, M_j\}^2$, on calcule la statistique du χ^2 croisant Y et les modalités $(M_i, M_\ell) \Rightarrow \chi^2(M_i, M_\ell)$ et $p(M_i, M_\ell)$ la probabilité critique associée.

Remarque

- 2 modalités discriminantes \Rightarrow dépendance forte dans le test avec $Y \Rightarrow$ "Fort rejet" de $H_0 \Rightarrow \chi^2$ élevé ou pc faible ;
- Regrouper les modalités peu discriminantes revient donc à regrouper celles qui ont un χ^2 faible ou une pc grande.

4. On choisit la **paire de modalités** qui minimise le χ^2 :

$$(\tilde{M}_i, \tilde{M}_\ell) = \underset{(M_i, M_\ell) \in \{M_1, \dots, M_j\}^2}{\operatorname{argmin}} \chi^2(M_i, M_\ell) = \underset{(M_i, M_\ell) \in \{M_1, \dots, M_j\}^2}{\operatorname{argmax}} p(M_i, M_\ell).$$

4. On choisit la **paire de modalités** qui minimise le χ^2 :

$$(\tilde{M}_i, \tilde{M}_\ell) = \underset{(M_i, M_\ell) \in \{M_1, \dots, M_j\}^2}{\operatorname{argmin}} \chi^2(M_i, M_\ell) = \underset{(M_i, M_\ell) \in \{M_1, \dots, M_j\}^2}{\operatorname{argmax}} p(M_i, M_\ell).$$

5. Si $p(\tilde{M}_i, \tilde{M}_\ell) > \alpha_2$ ($\alpha_2 \in]0, 1[$ **fixé par l'utilisateur**) alors on **regroupe les modalités \tilde{M}_i et \tilde{M}_ℓ** et on retourne à l'étape 2 avec le tableau à $M_j - 1$ modalités

	M_1	\dots	$M_j - 1$
1			
\vdots			
K			

Sinon, on stoppe les regroupements.

Exemple i

- On considère la variable **marstat** :

```
> aa <- table(USvoteS$vote3,USvoteS$marstat)
> aa
```

	married	widowed	divorced	never married
Gore	246	57	82	111
Bush	315	44	48	60

- On calcule les **probabilités critiques** pour les **6 croisements** :

Exemple ii

```
> res <- matrix(0,nrow=4,ncol=4)
> rownames(res) <- levels(USvoteS$marstat)
> colnames(res) <- levels(USvoteS$marstat)
> for (i in 1:3)
+   for (j in (i+1):4)
+     res[i,j] <- chisq.test(aa[,c(i,j)])$p.value
+
+
> res
```

	married	widowed	divorced	never married
married	0	0.0194	7.64e-05	1.41e-06
widowed	0	0.0000	3.06e-01	1.65e-01
divorced	0	0.0000	0.00e+00	7.42e-01
never married	0	0.0000	0.00e+00	0.00e+00

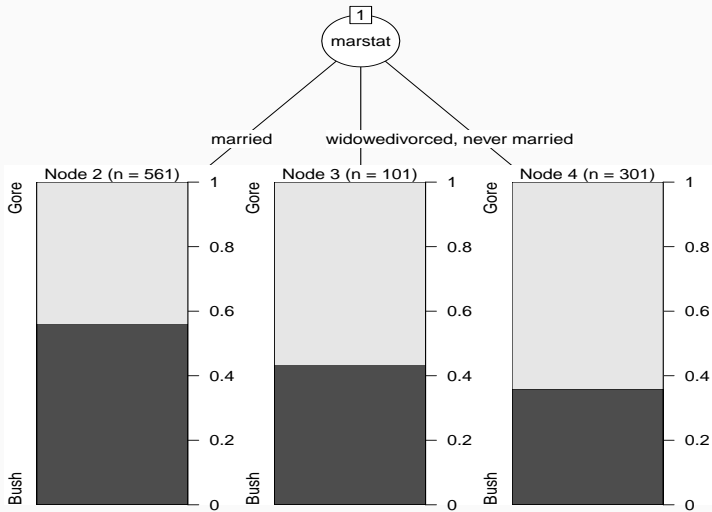
Exemple de regroupement

Les modalités **divorced** et **never married** sont regroupées (si $\alpha_2 < 0.742$).

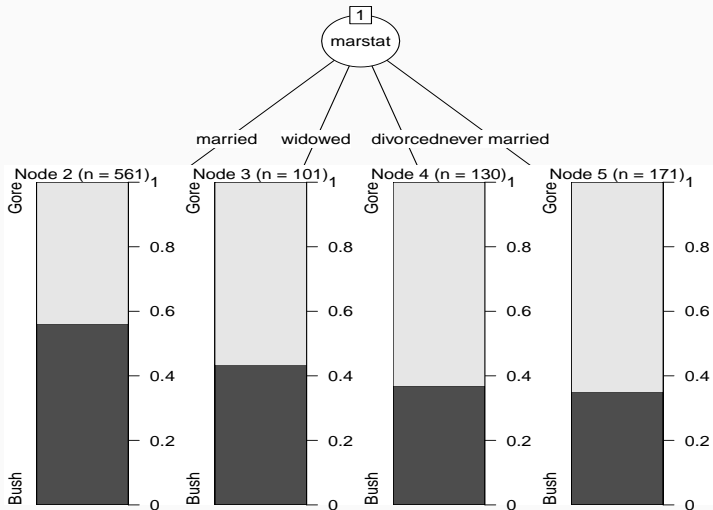
- En effet

```
> ctrl <- chaid_control(minsplit = 20,alpha2=0.74)
> a1 <- chaid(vote3~marstat,data=USvoteS,control = ctrl)
> plot(a1)
```

Exemple iv



```
> ctrl1 <- chaid_control(minsplit = 20,alpha2=0.75)
> a2 <- chaid(vote3~marstat,data=USvoteS,control = ctrl1)
> plot(a2)
```



- Variables **ordinales** : le traitement est **identique**. Seules les **modalités contiguës** peuvent être regroupées.

- Variables **ordinales** : le traitement est **identique**. Seules les **modalités contiguës** peuvent être regroupées.
- Variables **continues** : traitées comme des variables ordinales. Penser à utiliser **as.ordered** sur **R**.

Arbres binaires

Choix des découpes

Cas de la régression

Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

Regroupement des modalités

Division d'un nœud

Choix des paramètres

Bibliographie

Un autre χ^2 pour choisir la variable

- La phase **regroupement** effectuée, il faut choisir une variable parmi les p variables regroupées pour **diviser** le nœud.

Un autre χ^2 pour choisir la variable

- La phase **regroupement** effectuée, il faut choisir une variable parmi les p variables regroupées pour **diviser** le nœud.
- Idée** : faire un χ^2 pour chaque variable :

	(X_1, M_1)	...	(X_1, M_{1j})	(X_2, M_1)	...	(X_2, M_{2j})	...
1							
\vdots							
K							

$\Rightarrow p$ probabilités critiques $p(X_1), \dots, p(X_p)$ et

Un autre χ^2 pour choisir la variable

- La phase **regroupement** effectuée, il faut choisir une variable parmi les p variables regroupées pour **diviser** le nœud.
- Idée** : faire un χ^2 pour chaque variable :

	(X_1, M_1)	...	(X_1, M_{1j})	(X_2, M_1)	...	(X_2, M_{2j})	...
1							
\vdots							
K							

$\Rightarrow p$ probabilités critiques $p(X_1), \dots, p(X_p)$ et

- X_j discriminante \Rightarrow rejet de $H_0 \Rightarrow p(X_j)$ petite.

Un autre χ^2 pour choisir la variable

- La phase **regroupement** effectuée, il faut choisir une variable parmi les p variables regroupées pour **diviser** le nœud.
- Idée** : faire un χ^2 pour chaque variable :

	(X_1, M_1)	...	(X_1, M_{1j})	(X_2, M_1)	...	(X_2, M_{2j})	...
1							
\vdots							
K							

$\implies p$ probabilités critiques $p(X_1), \dots, p(X_p)$ et

- X_j discriminante \implies rejet de $H_0 \implies p(X_j)$ petite.
- On **choisit la variable j** qui possède la plus **petite probabilité critique**.

Correction de Bonferroni

- Tendance à **favoriser** les variables ayant subi le **plus de regroupements** (erreur de type 1).

Correction de Bonferroni

- Tendance à **favoriser** les variables ayant subi le **plus de regroupements** (erreur de type 1).
- Pour rééquilibrer, les probabilités critiques sont multipliées par le **coefficient de Bonferroni** :

$$p'(X_j) = b_j p(X_j)$$

où b_j correspond au nombre de manières les regrouper les M_j modalités initiales de X_j en \tilde{M}_j modalités finales.

Correction de Bonferroni

- Tendance à **favoriser** les variables ayant subi le **plus de regroupements** (erreur de type 1).
- Pour rééquilibrer, les probabilités critiques sont multipliées par le **coefficient de Bonferroni** :

$$p'(X_j) = b_j p(X_j)$$

où b_j correspond au nombre de manières les regrouper les M_j modalités initiales de X_j en \tilde{M}_j modalités finales.

- Variable **qualitative** et **ordinaire** :

$$b_j = \sum_{i=0}^{\tilde{M}_j-1} (-1)^i \frac{(\tilde{M}_j - i)^{M_j}}{i! (\tilde{M}_j - i)!} \quad b_j = \binom{M_j - 1}{\tilde{M}_j - 1}.$$

- On choisira la variable j^* qui minimise $p'(X_j)$...

- On choisira la variable j^* qui minimise $p'(X_j)$...
- à condition que $p'(X_j)$ soit plus petit qu'un certain seuil α_4 fixé par l'utilisateur.

- On choisira la variable j^* qui minimise $p'(X_j)$...
- à condition que $p'(X_j)$ soit plus petit qu'un certain seuil α_4 fixé par l'utilisateur.
- Le nœud sera scindé en autant de groupes que X_j possède de modalités (après la phase de regroupement).

Un nœud ne sera pas divisé si :

- $p'(X_j) > \alpha_4$ pour tout $j = 1, \dots, p$.
- le nœud est pur ou quasiment pur.
- le nœud contient trop peu d'observations...

Critère d'arrêt

Un nœud ne sera pas divisé si :

- $p'(X_j) > \alpha_4$ pour **tout** $j = 1, \dots, p$.
- le nœud est **pur** ou quasiment pur.
- le nœud contient **trop peu d'observations**...

Remarque

Sur **R**, on pourra regarder la fonction **chaid.control** :

```
chaid_control(alpha2 = 0.05, alpha3 = -1, alpha4 = 0.05,  
              minsplitlet = 20, minbucket = 7, minprob = 0.01,  
              stump = FALSE, maxheight = -1)
```

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres**

Bibliographie

- En plus des paramètres associés au critère d'arrêt, deux paramètres sont à calibrer pour construire l'arbre : les niveaux α_2 et α_4 .

- En plus des paramètres associés au critère d'arrêt, deux paramètres sont à calibrer pour construire l'arbre : les niveaux α_2 et α_4 .
- Il en existe un troisième (α_3) qui concerne la remise en cause des regroupements des modalités.

- En plus des paramètres associés au critère d'arrêt, deux paramètres sont à calibrer pour construire l'arbre : les niveaux α_2 et α_4 .
- Il en existe un troisième (α_3) qui concerne la remise en cause des regroupements des modalités.

Choix de α_4

Degrés d'exigence pour couper un nœud :

- petit : très exigeant \implies arbres peu profonds (beaucoup de biais et peu de variance) ;
- grand : peu exigeant \implies arbres profonds (beaucoup de variance et peu de biais).

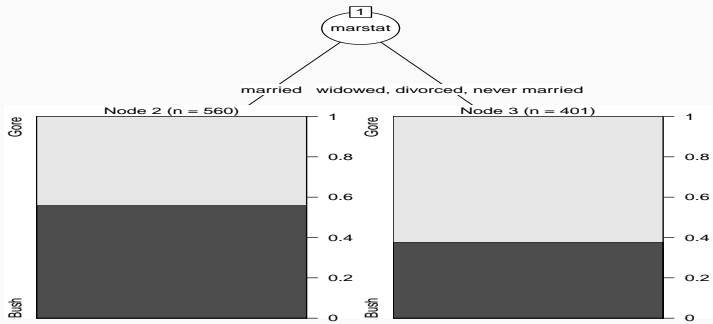
Choix de α_2

Degrés d'exigence pour regrouper des modalités :

- **petit** : peu exigeant \implies beaucoup de regroupements (on se rapproche des arbres binaires) ;
- **grand** : très exigeant \implies peu de regroupements.

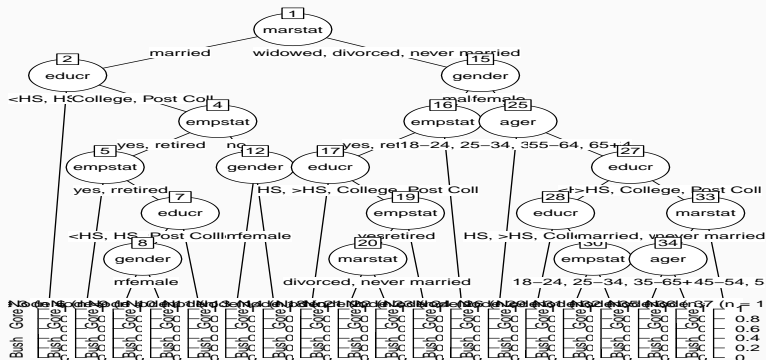
```
> ctrl <- chaid_control(minsplit = 20,alpha4=0.0005)
> a1 <- chaid(vote3~.,data=USvoteS,control=ctrl)
> plot(a1)
```


Illustration α_4 ii



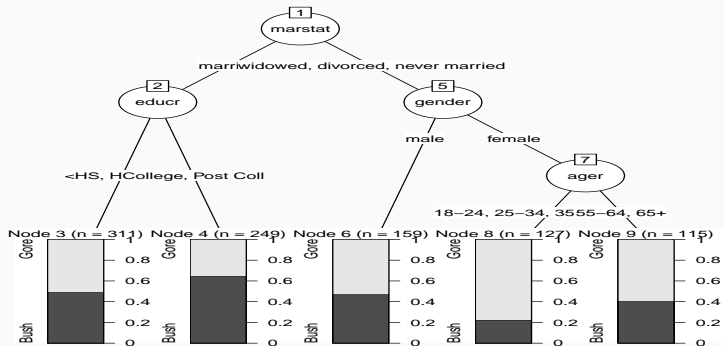
```
> ctrl <- chaid_control(minsplit = 20,alpha4=0.25)
> a2 <- chaid(vote3~.,data=USvoteS,control=ctrl)
> plot(a2)
```

Illustration α_4 iii

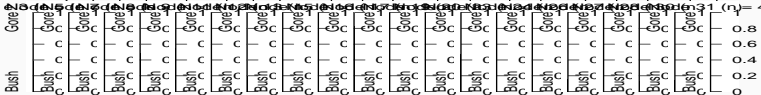


```
> ctrl <- chaid_control(minsplit = 20,alpha2=0.005)
> a3 <- chaid(vote3~.,data=USvoteS,control=ctrl)
> plot(a3)
```

Illustration α_2 ii



```
> ctrl <- chaid_control(minsplit = 20,alpha2=0.5)
> a4 <- chaid(vote3~.,data=USvoteS,control=ctrl)
> plot(a4)
```



- L'influence de ces deux paramètres est bien entendu conjointe.

- L'influence de ces deux paramètres est bien entendu conjointe.
- Il n'est pas facile de les calibrer simultanément.

- L'influence de ces deux paramètres est bien entendu conjointe.
- Il n'est pas facile de les calibrer simultanément.
- Approche classique : évaluer les performances (erreur de classification AUC...) pour plusieurs valeurs de (α_2, α_4) sur un échantillon test ou par validation croisée.

Exemple i

- On veut expliquer avec un arbre CHAID la variable **chd** par les autres variables du jeu de données **SAheart**.

```
> donnees <- SAheart  
> donnees$chd <- as.factor(donnees$chd)  
> for (i in c(1:4,6:9)){donnees[,i] <- as.ordered(donnees[,i])}
```

- On va séparer l'échantillon en 2 et estimer l'erreur de classification sur une grille de valeur de α_2 et α_4 :

Exemple ii

```
> alpha2 <- seq(0.01,0.35,by=0.05)
> alpha4 <- seq(0.01,0.35,by=0.05)
> gr.alpha <- expand.grid(alpha2,alpha4)
> names(gr.alpha) <- c("alpha2","alpha4")
> gr.alpha$perf <- 0
> set.seed(1234)
> perm <- sample(nrow(SAheart))
> dapp <- donnees[perm[1:300],]
> dtest <- donnees[-perm[1:300],]
```

- On estime l'erreur de classification sur les données test :

Exemple iii

```
> for (i in 1:nrow(gr.alpha)){  
>   ctrl <- chaid_control(alpha2=gr.alpha[i,1],alpha4=gr.alpha[i,2])  
>   a <- chaid(chd~.,data=dapp,control=ctrl)  
>   prev <- predict(a,newdata = dtest)  
>   gr.alpha$perf[i] <- mean(prev!=dtest$chd)  
> }
```

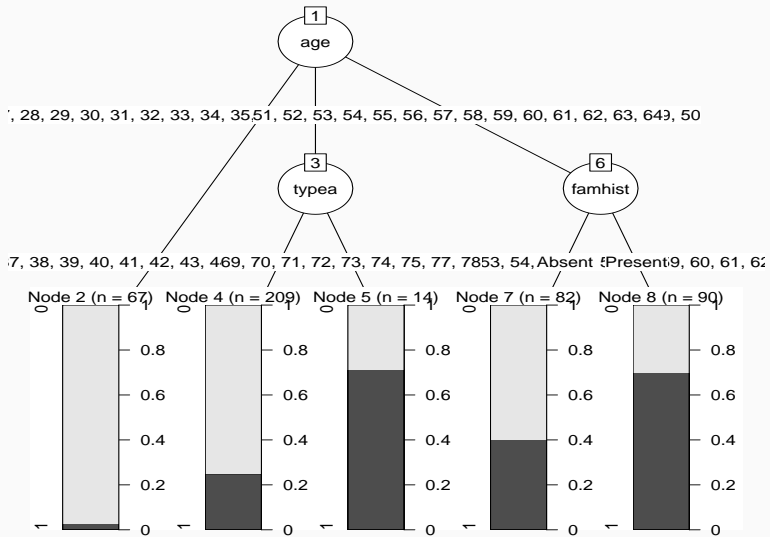
- On récupère les valeurs de α_2 et α_4 qui minimisent l'erreur estimée :

```
> alpha_opt <- gr.alpha[which.min(gr.alpha$perf),]  
> alpha_opt  
  alpha2 alpha4      perf  
1    0.01    0.01 0.2716049
```

- On peut tracer l'arbre sélectionné :

```
> ctrl <- chaid_control(alpha2=alpha_opt[1],alpha4=alpha_opt[2])  
> arbre_final <- chaid(chd~.,data=donnees,control=ctrl)  
> plot(arbre_final)
```

Example iv



- On peut faire la même chose avec **caret** (en plus efficace) :

```
> grille <- gr.alpha[,1:2]
> grille$alpha3 <- -1
> library(doMC)
> registerDoMC(cores = 3)
> bb <- train(donnees[,-10],donnees$chd,method="chaid",
             tuneGrid=grille,trControl=ctrl1,metric="Accuracy")
> bb
```

CHi-squared Automated Interaction Detection

462 samples

9 predictor

2 classes: '0', '1'

No pre-processing

Resampling: Repeated Train/Test Splits Estimated (1 reps, 75%)

Summary of sample sizes: 300

Resampling results across tuning parameters:

alpha2	alpha4	Accuracy	Kappa
0.01	0.01	0.7283951	0.3847747
0.01	0.06	0.7283951	0.3847747
0.01	0.11	0.7283951	0.3847747
0.01	0.16	0.7283951	0.3847747
0.01	0.21	0.7283951	0.3847747
0.01	0.26	0.6851852	0.2528486
0.01	0.31	0.6851852	0.2528486
0.06	0.01	0.6851852	0.2528486
0.06	0.06	0.6851852	0.2528486
0.06	0.11	0.6851852	0.2528486
0.06	0.16	0.6851852	0.2528486
0.06	0.21	0.6851852	0.2528486
0.06	0.26	0.6728395	0.3284843
0.06	0.31	0.6728395	0.2302313
0.11	0.01	0.6419753	0.2394366

0.11	0.06	0.6419753	0.2839506
0.11	0.11	0.6419753	0.2839506
0.11	0.16	0.6419753	0.2839506
0.11	0.21	0.6419753	0.2839506
0.11	0.26	0.6296296	0.2646391
0.11	0.31	0.6419753	0.2839506
0.16	0.01	0.6419753	0.2394366
0.16	0.06	0.6419753	0.2839506
0.16	0.11	0.6419753	0.2839506
0.16	0.16	0.6419753	0.2839506
0.16	0.21	0.6419753	0.2839506
0.16	0.26	0.6296296	0.2646391
0.16	0.31	0.6419753	0.2839506
0.21	0.01	0.6419753	0.2394366
0.21	0.06	0.6419753	0.2394366
0.21	0.11	0.6419753	0.2394366
0.21	0.16	0.6419753	0.2394366
0.21	0.21	0.6419753	0.2394366

0.21	0.26	0.6419753	0.2394366
0.21	0.31	0.6419753	0.2394366
0.26	0.01	0.6419753	0.2394366
0.26	0.06	0.6419753	0.2394366
0.26	0.11	0.6419753	0.2394366
0.26	0.16	0.6419753	0.2394366
0.26	0.21	0.6419753	0.2394366
0.26	0.26	0.6419753	0.2394366
0.26	0.31	0.6419753	0.2394366
0.31	0.01	0.6419753	0.2394366
0.31	0.06	0.6419753	0.2394366
0.31	0.11	0.6419753	0.2394366
0.31	0.16	0.6419753	0.2394366
0.31	0.21	0.6419753	0.2394366
0.31	0.26	0.6419753	0.2394366
0.31	0.31	0.6419753	0.2394366

Tuning parameter 'alpha3' was held constant at a value of -1

Accuracy was used to select the optimal model using the largest value. The final values used for the model were $\alpha_2 = 0.01$, $\alpha_3 = -1$ and $\alpha_4 = 0.21$.

Arbres binaires

Choix des découpes

- Cas de la régression

- Cas de la classification supervisée

Elagage

Annexe 1 : impureté, cas multiclassés

Annexe 2 : algorithme élagage

Annexe 3 : arbres Chaid

- Regroupement des modalités

- Division d'un nœud

- Choix des paramètres

Bibliographie

-  Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984).
Classification and regression trees.
Wadsworth & Brooks.
-  Cornillon, P. and Matzner-Løber, E. (2011).
Régression avec R.
Springer.
-  Devroye, L., Györfi, L., and Lugosi, G. (1996).
A Probabilistic Theory of Pattern Recognition.
Springer.



Devroye, L. and Krzyżak, A. (1989).

An equivalence theorem for l_1 convergence of the kernel regression estimate.

Journal of statistical Planning Inference, 23 :71–82.



Fahrmeir, L. and Kaufmann, H. (1985).

Consistency and asymptotic normality of the maximum likelihood estimator in generalized linear models.

The Annals of Statistics, 13 :342–368.



Grob, J. (2003).

Linear regression.

Springer.




Györfi, L., Kohler, M., Krzyzak, A., and Harro, W. (2002).
A Distribution-Free Theory of Nonparametric Regression.
Springer.



Hastie, T., Tibshirani, R., and Friedman, J. (2009).
***The Elements of Statistical Learning : Data Mining, Inference,
and Prediction.***
Springer, second edition.



Kass, G. (1980).
**An exploratory technique for investigating large quantities of
categorical data.**
Applied Statistics, 29(2) :119–127.

-  Stone, C. J. (1977).
Consistent nonparametric regression.
Annals of Statistics, 5 :595–645.

Quatrième partie IV

Agrégation

Bagging et forêts aléatoires

- Bagging

- Forêts aléatoires

Boosting

- Algorithmes de gradient boosting

- Choix des paramètres

Bibliographie

Les approches que nous allons étudier sont basées sur l'agrégation :

1. construire un grand nombre de classifieurs "simples" g_1, \dots, g_B

Les approches que nous allons étudier sont basées sur l'agrégation :

1. construire un grand nombre de classifieurs "simples" g_1, \dots, g_B
2. que l'on agrège

$$\hat{g}(x) = \frac{1}{B} \sum_{k=1}^B g_k(x).$$

Les approches que nous allons étudier sont basées sur l'agrégation :

1. construire un grand nombre de classifieurs "simples" g_1, \dots, g_B
2. que l'on agrège

$$\hat{g}(x) = \frac{1}{B} \sum_{k=1}^B g_k(x).$$

Questions

1. Intérêt d'agréger ?
2. Comment construire les g_k pour que \hat{g} soit performant ?

Bagging et forêts aléatoires

- Bagging

- Forêts aléatoires

Boosting

- Algorithmes de gradient boosting

- Choix des paramètres

Bibliographie

- Idem que précédemment, on cherche à expliquer une variable Y par d variables explicatives X_1, \dots, X_d .

- Idem que précédemment, on cherche à **expliquer** une variable Y par d variables explicatives X_1, \dots, X_d .
- Pour simplifier on se place en **régression** : Y est à valeurs dans \mathbb{R} mais tout ce qui va être fait s'étant directement à la **classification binaire ou multiclass**.

- Idem que précédemment, on cherche à **expliquer** une variable Y par d variables explicatives X_1, \dots, X_d .
- Pour simplifier on se place en **régression** : Y est à valeurs dans \mathbb{R} mais tout ce qui va être fait s'étant directement à la **classification binaire ou multiclassées**.
- **Notations** :
 - (X, Y) un couple aléatoire à valeurs dans $\mathbb{R}^d \times \mathbb{R}$.
 - $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$ un n -échantillon i.i.d. de même loi que (X, Y) .

Bagging et forêts aléatoires

- Bagging

- Forêts aléatoires

Boosting

- Algorithmes de gradient boosting

- Choix des paramètres

Bibliographie

- Le **bagging** désigne un ensemble de méthodes introduit par Léo Breiman [[Breiman, 1996](#)].
- **Bagging** : vient de la contraction de **B**ootstrap **A**ggregating.
- **Idée** : plutôt que de construire un seul estimateur, en construire un grand nombre (sur des échantillons **bootstrap**) et les **agréger**.

Pourquoi agréger ?

- On se place dans le modèle de **régression**.

$$Y = m(X) + \varepsilon.$$

- On note

$$\hat{m}_B(x) = \frac{1}{B} \sum_{k=1}^B m_k(x)$$

un estimateur de m obtenu en **agrégeant** B estimateurs m_1, \dots, m_B .

Pourquoi agréger ?

- On se place dans le modèle de **régression**.

$$Y = m(X) + \varepsilon.$$

- On note

$$\hat{m}_B(x) = \frac{1}{B} \sum_{k=1}^B m_k(x)$$

un estimateur de m obtenu en **agrégeant** B estimateurs m_1, \dots, m_B .

- **Rappels** : $\hat{m}_B(x) = \hat{m}_B(x; (X_1, Y_1), \dots, (X_n, Y_n))$ et $m_k(x) = m_k(x; (X_1, Y_1), \dots, (X_n, Y_n))$ sont des **variables aléatoires**.

Pourquoi agréger ?

- On se place dans le modèle de **régression**.

$$Y = m(X) + \varepsilon.$$

- On note

$$\hat{m}_B(x) = \frac{1}{B} \sum_{k=1}^B m_k(x)$$

un estimateur de m obtenu en **agrégeant** B estimateurs m_1, \dots, m_B .

- Rappels** : $\hat{m}_B(x) = \hat{m}_B(x; (X_1, Y_1), \dots, (X_n, Y_n))$ et $m_k(x) = m_k(x; (X_1, Y_1), \dots, (X_n, Y_n))$ sont des **variables aléatoires**.
- On peut **mesurer l'intérêt d'agréger** en comparant les performances de $\hat{m}_B(x)$ à celles des $m_k(x)$, $k = 1, \dots, B$ (en comparant, par exemple, le **biais** et la **variance** de ces estimateurs).

- **Hypothèse** : les variables aléatoires m_1, \dots, m_B sont i.i.d.

- **Hypothèse** : les variables aléatoires m_1, \dots, m_B sont i.i.d.
 - **Biais** :

$$\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x)].$$

Conclusion

Agréger ne modifie pas le biais.

Biais et variance

- **Hypothèse** : les variables aléatoires m_1, \dots, m_B sont i.i.d.
 - **Biais** :

$$\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x)].$$

Conclusion

Agréger ne modifie pas le biais.

- **Variance** :

$$\mathbf{V}[\hat{m}_B(x)] = \frac{1}{B} \mathbf{V}[m_k(x)].$$

Conclusion

Agréger tue la variance.

- Les conclusions précédentes sont vraies sous l'hypothèse que les variables aléatoires m_1, \dots, m_B sont i.i.d.

- Les conclusions précédentes sont vraies sous l'hypothèse que les variables aléatoires m_1, \dots, m_B sont i.i.d.
- Les estimateurs m_1, \dots, m_B étant construits sur le même échantillon, l'hypothèse d'indépendance n'est clairement pas raisonnable !

- Les conclusions précédentes sont vraies sous l'hypothèse que les variables aléatoires m_1, \dots, m_B sont i.i.d.
- Les estimateurs m_1, \dots, m_B étant construits sur le même échantillon, l'hypothèse d'indépendance n'est clairement pas raisonnable !

Idée

Atténuer la dépendance entre les estimateurs $m_k, k = 1, \dots, B$ en introduisant de nouvelles sources d'aléa.

Idée : échantillons bootstrap

- Echantillon **initial** :

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

Idée : échantillons bootstrap

- Echantillon **initial** :

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- Echantillons **bootstrap** :

3	4	6	10	3	9	10	7	7	1	m_1
2	8	6	2	10	10	2	9	5	6	m_2
2	9	4	4	7	7	2	3	6	7	m_3
6	1	3	3	9	3	8	10	10	1	m_4
3	7	10	3	2	8	6	9	10	2	m_5
	\vdots								\vdots	
7	10	3	4	9	10	10	8	6	1	m_B

Idée : échantillons bootstrap

- Echantillon **initial** :

1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

- Echantillons **bootstrap** :

3	4	6	10	3	9	10	7	7	1	m_1
2	8	6	2	10	10	2	9	5	6	m_2
2	9	4	4	7	7	2	3	6	7	m_3
6	1	3	3	9	3	8	10	10	1	m_4
3	7	10	3	2	8	6	9	10	2	m_5
	\vdots								\vdots	
7	10	3	4	9	10	10	8	6	1	m_B

- A la fin, on **agrège** :

$$\hat{m}_B(x) = \frac{1}{B} \sum_{k=1}^B m_k(x).$$

- Les m_k ne vont pas être construits sur l'échantillon $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$, mais sur des **échantillons bootstrap** de \mathcal{D}_n .

Bagging

- Les m_k ne vont pas être construits sur l'échantillon $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$, mais sur des **échantillons bootstrap** de \mathcal{D}_n .

Bagging

Entrées :

- $x \in \mathbb{R}^d$ l'observation à prévoir, \mathcal{D}_n l'échantillon
- un régresseur (arbre CART, 1 plus proche voisin...)
- B le nombre d'estimateurs que l'on agrège.

Bagging

- Les m_k ne vont pas être construits sur l'échantillon $\mathcal{D}_n = (X_1, Y_1), \dots, (X_n, Y_n)$, mais sur des **échantillons bootstrap** de \mathcal{D}_n .

Bagging

Entrées :

- $x \in \mathbb{R}^d$ l'observation à prévoir, \mathcal{D}_n l'échantillon
- un régresseur (arbre CART, 1 plus proche voisin...)
- B le nombre d'estimateurs que l'on agrège.

Pour $k = 1, \dots, B$:

1. Tirer un échantillon **bootstrap** dans \mathcal{D}_n
2. **Ajuster le régresseur** sur cet échantillon bootstrap : $m_k(x)$

Sortie : L'estimateur $\hat{m}_B(x) = \frac{1}{B} \sum_{k=1}^B m_k(x)$.

Tirage de l'échantillon bootstrap

- Les tirages bootstrap sont représentés par B variables aléatoires $\theta_k, k = 1, \dots, B$.

Tirage de l'échantillon bootstrap

- Les tirages bootstrap sont représentés par B variables aléatoires $\theta_k, k = 1, \dots, B$.
- Les tirages bootstrap sont généralement effectués selon la même loi et de façon indépendante : $\theta_1, \dots, \theta_B$ sont i.i.d. de même loi que θ .

Tirage de l'échantillon bootstrap

- Les tirages bootstrap sont représentés par B variables aléatoires $\theta_k, k = 1, \dots, B$.
- Les tirages bootstrap sont généralement effectués selon la même loi et de façon indépendante : $\theta_1, \dots, \theta_B$ sont i.i.d. de même loi que θ .
- 2 techniques sont généralement utilisées :
 1. tirage de n observations avec remise ;
 2. tirage de $\ell < n$ observation sans remise.

Tirage de l'échantillon bootstrap

- Les tirages bootstrap sont représentés par B variables aléatoires $\theta_k, k = 1, \dots, B$.
- Les tirages bootstrap sont généralement effectués selon la même loi et de façon indépendante : $\theta_1, \dots, \theta_B$ sont i.i.d. de même loi que θ .
- 2 techniques sont généralement utilisées :
 1. tirage de n observations avec remise ;
 2. tirage de $\ell < n$ observation sans remise.

Conséquence

Les estimateurs agrégés contiennent 2 sources d'aléa (échantillon et tirage bootstrap) :

$$m_k(x) = m(x, \theta_k, \mathcal{D}_n).$$

Choix du nombre d'itérations

- Deux paramètres sont à choisir : le nombre d'itérations B et le régresseur.

Choix du nombre d'itérations

- Deux paramètres sont à choisir : le nombre d'itérations B et le régresseur.
- On a d'après la loi des grands nombres

$$\begin{aligned}\lim_{B \rightarrow \infty} \hat{m}_B(x) &= \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m_k(x) = \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m(x, \theta_k, \mathcal{D}_n) \\ &= \mathbf{E}_{\theta}[m(x, \theta, \mathcal{D}_n)] = \bar{m}(x, \mathcal{D}_n) \quad p.s|\mathcal{D}_n.\end{aligned}$$

Choix du nombre d'itérations

- Deux paramètres sont à choisir : le nombre d'itérations B et le régresseur.
- On a d'après la loi des grands nombres

$$\begin{aligned}\lim_{B \rightarrow \infty} \hat{m}_B(x) &= \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m_k(x) = \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m(x, \theta_k, \mathcal{D}_n) \\ &= \mathbf{E}_{\theta}[m(x, \theta, \mathcal{D}_n)] = \bar{m}(x, \mathcal{D}_n) \quad p.s|\mathcal{D}_n.\end{aligned}$$

- Lorsque B est grand, \hat{m}_B se "stabilise" vers l'estimateur bagging $\bar{m}(x, \mathcal{D}_n)$.

Choix du nombre d'itérations

- Deux paramètres sont à choisir : le nombre d'itérations B et le régresseur.
- On a d'après la loi des grands nombres

$$\begin{aligned}\lim_{B \rightarrow \infty} \hat{m}_B(x) &= \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m_k(x) = \lim_{B \rightarrow \infty} \frac{1}{B} \sum_{k=1}^B m(x, \theta_k, \mathcal{D}_n) \\ &= \mathbf{E}_{\theta}[m(x, \theta, \mathcal{D}_n)] = \bar{m}(x, \mathcal{D}_n) \quad p.s|\mathcal{D}_n.\end{aligned}$$

- Lorsque B est grand, \hat{m}_B se "stabilise" vers l'estimateur bagging $\bar{m}(x, \mathcal{D}_n)$.

Conséquence importante

Le nombre d'itérations B n'est pas un paramètre à calibrer, il est conseillé de le prendre le plus grand possible en fonction du temps de calcul.

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x)\mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Choix du régresseur

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x)\mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Conclusion

- Bagging ne modifie pas le biais.

Choix du régresseur

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x) \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B} \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Conclusion

- Bagging ne modifie pas le biais.
- B grand $\implies \mathbf{V}[\hat{m}_B(x)] \approx \rho(x) \mathbf{V}[\hat{m}_k(x, \theta_k(\mathcal{D}_n))]$

Choix du régresseur

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x) \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B} \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Conclusion

- Bagging ne modifie pas le biais.
- B grand $\implies \mathbf{V}[\hat{m}_B(x)] \approx \rho(x) \mathbf{V}[\hat{m}_k(x, \theta_k(\mathcal{D}_n))] \implies$ la **variance diminue d'autant plus que la corrélation entre les prédicteurs diminue.**

Choix du régresseur

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x) \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B} \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Conclusion

- Bagging ne modifie pas le biais.
- B grand $\implies \mathbf{V}[\hat{m}_B(x)] \approx \rho(x) \mathbf{V}[\hat{m}_k(x, \theta_k(\mathcal{D}_n))] \implies$ la **variance diminue d'autant plus que la corrélation entre les prédicteurs diminue**.
- Il est donc nécessaire d'agréger des estimateurs **sensibles à de légères perturbations de l'échantillon**.

Choix du régresseur

Propriété : biais et variance

On a $\mathbf{E}[\hat{m}_B(x)] = \mathbf{E}[m_k(x, \theta_k, \mathcal{D}_n)]$ et

$$\mathbf{V}[\hat{m}_B(x)] = \rho(x) \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B} \mathbf{V}[m(x, \theta_k, \mathcal{D}_n)]$$

où $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$ pour $k \neq k'$.

Conclusion

- Bagging ne modifie pas le biais.
- B grand $\implies \mathbf{V}[\hat{m}_B(x)] \approx \rho(x) \mathbf{V}[\hat{m}_k(x, \theta_k(\mathcal{D}_n))] \implies$ la **variance diminue d'autant plus que la corrélation entre les prédicteurs diminue**.
- Il est donc nécessaire d'agréger des estimateurs **sensibles à de légères perturbations de l'échantillon**.
- Les **arbres** sont connus pour posséder de telles propriétés.

Bagging et forêts aléatoires

Bagging

Forêts aléatoires

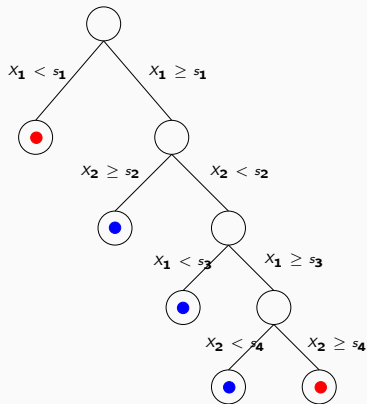
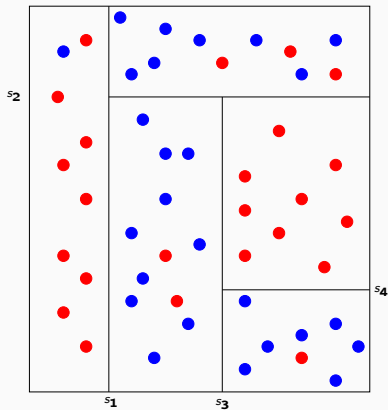
Boosting

Algorithmes de gradient boosting

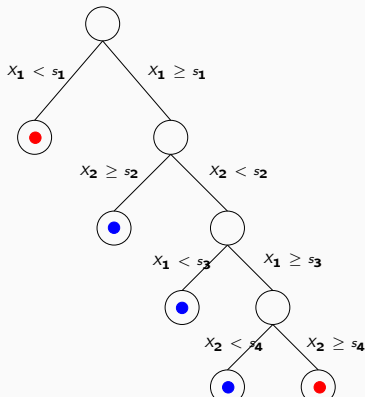
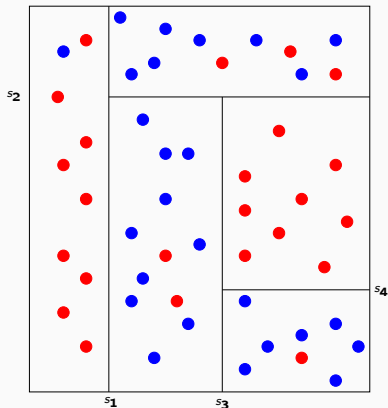
Choix des paramètres

Bibliographie

Rappels sur les arbres



Rappels sur les arbres



Paramètre à calibrer

Profondeur

- petite : biais ↗, variance ↘
- grande : biais ↘, variance ↗

- Comme son nom l'indique, une forêt aléatoire est définie à partir d'un ensemble d'arbres.

- Comme son nom l'indique, une forêt aléatoire est définie à partir d'un ensemble d'arbres.

Définition

Soit $T_k(x)$, $k = 1, \dots, B$ des prédicteurs par arbre ($T_k : \mathbb{R}^d \rightarrow \mathbb{R}$). Le prédicteur des forêts aléatoires est obtenu par agrégation de cette collection d'arbres :

$$\hat{T}_B(x) = \frac{1}{B} \sum_{k=1}^B T_k(x).$$

- Forêts aléatoires = collection d'arbres.

Forêts aléatoires

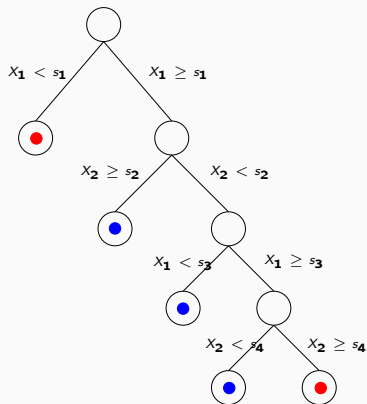
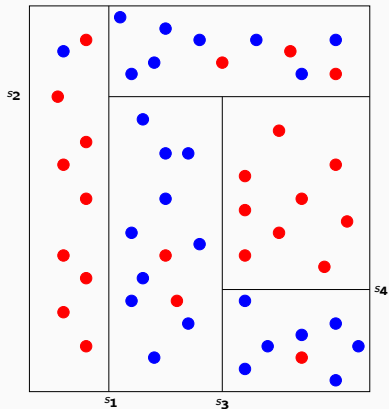
- Forêts aléatoires = collection d'arbres.
- Les forêts aléatoires les plus utilisées sont (de loin) celles proposées par Léo Breiman (au début des années 2000).

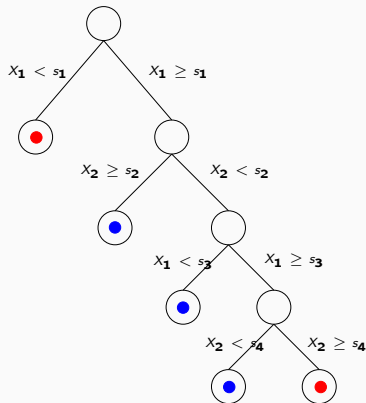
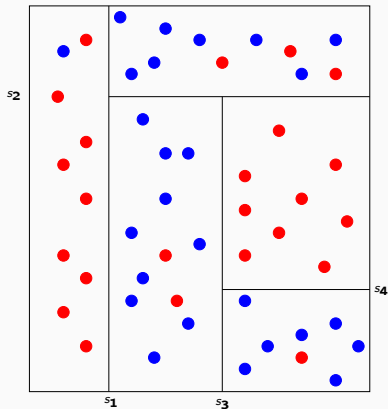
Forêts aléatoires

- Forêts aléatoires = collection d'arbres.
- Les forêts aléatoires les plus utilisées sont (de loin) celles proposées par Léo Breiman (au début des années 2000).
- Elles consistent à agréger des arbres construits sur des échantillons bootstrap.

Forêts aléatoires

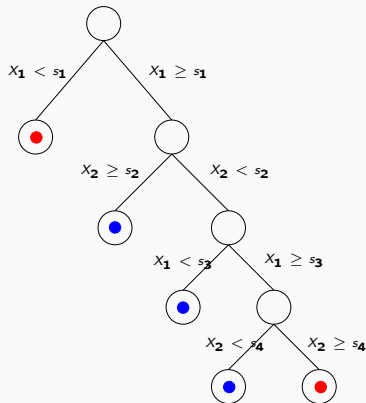
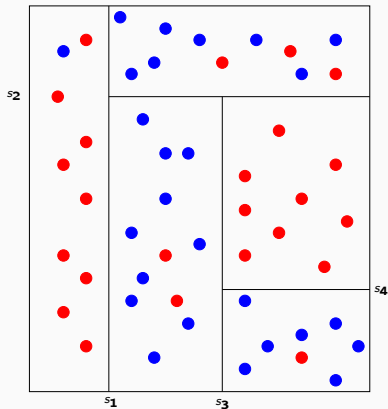
- Forêts aléatoires = collection d'arbres.
- Les forêts aléatoires les plus utilisées sont (de loin) celles proposées par Léo Breiman (au début des années 2000).
- Elles consistent à agréger des arbres construits sur des échantillons bootstrap.
- On pourra trouver de la doc à l'url
<http://www.stat.berkeley.edu/~breiman/RandomForests/>
et consulter la thèse de Robin Genuer [Genuer, 2010].





Arbres pour forêt

- Breiman propose de sélectionner la "meilleure" variable dans un ensemble composé **uniquement de m variables choisies aléatoirement parmi les d variables initiales.**



Arbres pour forêt

- Breiman propose de sélectionner la "meilleure" variable dans un ensemble composé **uniquement de m variables choisies aléatoirement parmi les d variables initiales.**
- Objectif :** **diminuer la corrélation** entre les arbres que l'on agrège.

Algorithme : randomforest

Entrées :

- $x \in \mathbb{R}^d$ l'observation à prévoir, \mathcal{D}_n l'échantillon ;
- B nombre d'arbres ; n_{max} nombre max d'observations par nœud
- $m \in \{1, \dots, d\}$ le nombre de variables candidates pour découper un nœud.

Algorithme : randomforest

Entrées :

- $x \in \mathbb{R}^d$ l'observation à prévoir, \mathcal{D}_n l'échantillon ;
- B nombre d'arbres ; n_{max} nombre max d'observations par nœud
- $m \in \{1, \dots, d\}$ le nombre de variables candidates pour découper un nœud.

Pour $k = 1, \dots, B$:

1. Tirer un échantillon **bootstrap** dans \mathcal{D}_n
2. Construire un **arbre CART sur cet échantillon bootstrap**, chaque coupure est sélectionnée en minimisant la fonction de coût de CART sur un ensemble de **m variables choisies au hasard** parmi les d . On note $T(., \theta_k, \mathcal{D}_n)$ l'arbre construit.

Sortie : l'estimateur $T_B(x) = \frac{1}{B} \sum_{k=1}^B T(x, \theta_k, \mathcal{D}_n)$.

- Si on est en discrimination (Y qualitative), l'étape d'agrégation consiste à faire voter les arbres à la majorité.

- Si on est en discrimination (Y qualitative), l'étape d'agrégation consiste à faire voter les arbres à la majorité.
- Il y a deux sources d'aléa présentes dans θ_k : le tirage bootstrap et les m variables sélectionnées à chaque étape de la construction de l'arbre.

- Si on est en discrimination (Y qualitative), l'étape d'agrégation consiste à faire voter les arbres à la majorité.
- Il y a deux sources d'aléa présentes dans θ_k : le tirage bootstrap et les m variables sélectionnées à chaque étape de la construction de l'arbre.
- Méthode simple à mettre en oeuvre et déjà implémentée sur la plupart des logiciels statistiques (sur R, il suffit de lancer la fonction randomForest du package randomForest).

- Si on est en discrimination (Y qualitative), l'étape d'agrégation consiste à faire voter les arbres à la majorité.
- Il y a deux sources d'aléa présentes dans θ_k : le tirage bootstrap et les m variables sélectionnées à chaque étape de la construction de l'arbre.
- Méthode simple à mettre en oeuvre et déjà implémentée sur la plupart des logiciels statistiques (sur R, il suffit de lancer la fonction randomForest du package randomForest).
- Estimateur connu pour fournir des estimations précises sur des données complexes (beaucoup de variables, données manquantes...).

- Si on est en discrimination (Y qualitative), l'étape d'agrégation consiste à faire voter les arbres à la majorité.
- Il y a deux sources d'aléa présentes dans θ_k : le tirage bootstrap et les m variables sélectionnées à chaque étape de la construction de l'arbre.
- Méthode simple à mettre en oeuvre et déjà implémentée sur la plupart des logiciels statistiques (sur R, il suffit de lancer la fonction randomForest du package randomForest).
- Estimateur connu pour fournir des estimations précises sur des données complexes (beaucoup de variables, données manquantes...).
- Estimateur peu sensible au choix de ses paramètres (B , n_{max} , $m...$)

Choix des paramètres

- B : réglé... le plus grand possible.

Choix des paramètres

- B : réglé... le plus grand possible.

Intérêt du bagging (rappel)

Diminuer la variance des estimateurs qu'on agrège :

$$\mathbf{V}[\hat{T}_B(x)] = \rho(x)\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)]$$

Choix des paramètres

- B : réglé... le plus grand possible.

Intérêt du bagging (rappel)

Diminuer la variance des estimateurs qu'on agrège :

$$\mathbf{V}[\hat{T}_B(x)] = \rho(x)\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)]$$

Conséquence

- Le biais n'étant pas amélioré par "l'agrégation bagging", il est recommandé d'agréger des estimateurs qui possèdent un **biais faible** (contrairement au boosting).

Choix des paramètres

- B : réglé... le plus grand possible.

Intérêt du bagging (rappel)

Diminuer la variance des estimateurs qu'on agrège :

$$\mathbf{V}[\hat{T}_B(x)] = \rho(x)\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)]$$

Conséquence

- Le biais n'étant pas amélioré par "l'agrégation bagging", il est recommandé d'agréger des estimateurs qui possèdent un **biais faible** (contrairement au boosting).
- Arbres "profonds", peu d'observations dans les nœuds terminaux.

Choix des paramètres

- B : réglé... le plus grand possible.

Intérêt du bagging (rappel)

Diminuer la variance des estimateurs qu'on agrège :

$$\mathbf{V}[\hat{T}_B(x)] = \rho(x)\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)] + \frac{1 - \rho(x)}{B}\mathbf{V}[T(x, \theta_k, \mathcal{D}_n)]$$

Conséquence

- Le biais n'étant pas amélioré par "l'agrégation bagging", il est recommandé d'agréger des estimateurs qui possèdent un biais faible (contrairement au boosting).
- Arbres "profonds", peu d'observations dans les nœuds terminaux.
- Par défaut dans randomForest, $n_{max} = 5$ en régression et 1 en classification.

- Il est en relation avec la corrélation entre les arbres $\rho(x)$.

Choix de m

- Il est en relation avec la corrélation entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.

Choix de m

- Il est en relation avec la corrélation entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$

Choix de m

- Il est en relation avec la corrélation entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un choix "aléatoire" des variables de découpe des arbres

Choix de m

- Il est en **relation avec la corrélation** entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un **choix "aléatoire"** des variables de découpe des arbres \implies les arbres sont de plus en plus différents

Choix de m

- Il est en **relation avec la corrélation** entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un **choix "aléatoire"** des variables de découpe des arbres \implies les arbres sont de plus en plus différents $\implies \rho(x) \searrow \implies$ **la variance de la forêt diminue**.

Choix de m

- Il est en **relation avec la corrélation** entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un **choix "aléatoire"** des variables de découpe des arbres \implies les arbres sont de plus en plus différents $\implies \rho(x) \searrow \implies$ **la variance de la forêt diminue**.
 2. mais... le biais des arbres \nearrow

Choix de m

- Il est en **relation avec la corrélation** entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un **choix "aléatoire"** des variables de découpe des arbres \implies les arbres sont de plus en plus différents $\implies \rho(x) \searrow \implies$ **la variance de la forêt diminue**.
 2. mais... le biais des arbres $\nearrow \implies$ le **biais de la forêt** \nearrow .

Choix de m

- Il est en relation avec la corrélation entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un choix "aléatoire" des variables de découpe des arbres \implies les arbres sont de plus en plus différents $\implies \rho(x) \searrow \implies$ la variance de la forêt diminue.
 2. mais... le biais des arbres $\nearrow \implies$ le biais de la forêt \nearrow .
- Inversement lorsque $m \nearrow$.

Choix de m

- Il est en **relation avec la corrélation** entre les arbres $\rho(x)$.
- Ce paramètre a une influence sur le compromis biais/variance de la forêt.
- $m \searrow$
 1. tendance à se rapprocher d'un **choix "aléatoire"** des variables de découpe des arbres \implies les arbres sont de plus en plus différents $\implies \rho(x) \searrow \implies$ **la variance de la forêt diminue**.
 2. mais... le biais des arbres $\nearrow \implies$ le **biais de la forêt** \nearrow .
- Inversement lorsque $m \nearrow$.

Conclusion

- Il est recommandé de comparer les performances de la forêt pour **plusieurs valeurs de m** .
- Par défaut $m = d/3$ en régression et \sqrt{d} en classification.

Application sur les données spam

```
> library(randomForest)
> foret1 <- randomForest(type~.,data=spam)
> foret1
##
## Call:
## randomForest(formula = type ~ ., data = spam)
##               Type of random forest: classification
##               Number of trees: 500
## No. of variables tried at each split: 7
##
## OOB estimate of  error rate: 4.52%
## Confusion matrix:
##           nonspam spam class.error
## nonspam    2708   80  0.02869440
## spam        128 1685  0.07060121
```

Mesure de performance

- Comme pour les autres classifieurs et régresseurs il convient de définir des critères qui permettent de mesurer la performance des forêts aléatoires.

- Comme pour les autres classifieurs et régresseurs il convient de définir des critères qui permettent de **mesurer la performance des forêts aléatoires**.
- Exemples :
 - Erreur de prédiction : $\mathbf{E}[(Y - \hat{T}_B(X))^2]$ en régression ;
 - Probabilité d'erreur : $\mathbf{P}(Y \neq \hat{T}_B(X))$ en classification.

- Comme pour les autres classifieurs et régresseurs il convient de définir des critères qui permettent de **mesurer la performance des forêts aléatoires**.
- Exemples :
 - Erreur de prédiction : $\mathbf{E}[(Y - \hat{T}_B(X))^2]$ en régression ;
 - Probabilité d'erreur : $\mathbf{P}(Y \neq \hat{T}_B(X))$ en classification.
- Comme pour les autres méthodes, ces critères peuvent être évalués par **apprentissage/validation** ou **validation croisée**.

Mesure de performance

- Comme pour les autres classifieurs et régresseurs il convient de définir des critères qui permettent de **mesurer la performance des forêts aléatoires**.
- Exemples :
 - Erreur de prédiction : $\mathbf{E}[(Y - \hat{T}_B(X))^2]$ en régression ;
 - Probabilité d'erreur : $\mathbf{P}(Y \neq \hat{T}_B(X))$ en classification.
- Comme pour les autres méthodes, ces critères peuvent être évalués par **apprentissage/validation** ou **validation croisée**.
- La phase **bootstrap** des algorithmes bagging permet de définir une nouvelle méthode d'estimation de ces critères : méthode **OOB (Out Of Bag)**.

- Pour chaque observation (X_i, Y_i) de \mathcal{D}_n , on désigne par \mathcal{I}_B l'ensemble des arbres de la forêt qui **ne contiennent pas cette observation** dans leur échantillon bootstrap.
- La prévision de Y au point X_i se fait selon

$$\hat{Y}_i = \frac{1}{|\mathcal{I}_B|} \sum_{k \in \mathcal{I}_B} T(X_i, \theta_k, \mathcal{D}_n).$$

Erreur Out Of Bag

- Pour chaque observation (X_i, Y_i) de \mathcal{D}_n , on désigne par \mathcal{I}_B l'ensemble des arbres de la forêt qui **ne contiennent pas cette observation** dans leur échantillon bootstrap.
- La prévision de Y au point X_i se fait selon

$$\hat{Y}_i = \frac{1}{|\mathcal{I}_B|} \sum_{k \in \mathcal{I}_B} T(X_i, \theta_k, \mathcal{D}_n).$$

Estimateurs Out Of Bag

- L'**erreur de prédiction** est estimée par $\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2$.
- La **probabilité d'erreur** est estimée par $\frac{1}{n} \sum_{i=1}^n \mathbf{1}_{\hat{Y}_i \neq Y_i}$.

Exemple

3	4	6	10	3	9	10	7	7	1	m_1
2	8	6	2	10	10	2	9	5	6	m_2
2	9	4	4	7	7	2	3	6	7	m_3
6	1	3	3	9	3	8	10	10	1	m_4
3	7	10	3	2	8	6	9	10	2	m_5
7	10	3	4	9	10	10	8	6	1	m_6

Exemple

3	4	6	10	3	9	10	7	7	1	m_1
2	8	6	2	10	10	2	9	5	6	m_2
2	9	4	4	7	7	2	3	6	7	m_3
6	1	3	3	9	3	8	10	10	1	m_4
3	7	10	3	2	8	6	9	10	2	m_5
7	10	3	4	9	10	10	8	6	1	m_6

- Les échantillons 2, 3 et 5 ne contiennent pas la première observation, donc

$$\hat{Y}_1 = \frac{1}{3}(m_2(X_1) + m_3(X_1) + m_5(X_1)).$$

- On fait de même pour toutes les observations $\Rightarrow \hat{Y}_2, \dots, \hat{Y}_n$.

Exemple

3	4	6	10	3	9	10	7	7	1	m_1
2	8	6	2	10	10	2	9	5	6	m_2
2	9	4	4	7	7	2	3	6	7	m_3
6	1	3	3	9	3	8	10	10	1	m_4
3	7	10	3	2	8	6	9	10	2	m_5
7	10	3	4	9	10	10	8	6	1	m_6

- Les échantillons 2, 3 et 5 ne contiennent pas la première observation, donc

$$\hat{Y}_1 = \frac{1}{3}(m_2(X_1) + m_3(X_1) + m_5(X_1)).$$

- On fait de même pour toutes les observations $\Rightarrow \hat{Y}_2, \dots, \hat{Y}_n$.
- On estime l'erreur selon

$$\frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2.$$

Exemple

- On construit la forêt avec $m = 1$:

```
> foret2 <- randomForest(type~.,data=spam,mtry=1)
> foret2
##
## Call:
##  randomForest(formula = type ~ ., data = spam, mtry = 1)
##                Type of random forest: classification
##                Number of trees: 500
## No. of variables tried at each split: 1
##
##          OOB estimate of  error rate: 8.06%
## Confusion matrix:
##           nonspam spam class.error
## nonspam      2725   63  0.02259684
## spam          308 1505  0.16988417
```

Exemple

- On construit la forêt avec $m = 1$:

```
> foret2 <- randomForest(type~.,data=spam,mtry=1)
> foret2
##
## Call:
## randomForest(formula = type ~ ., data = spam, mtry = 1)
##              Type of random forest: classification
##              Number of trees: 500
## No. of variables tried at each split: 1
##
##          OOB estimate of  error rate: 8.06%
## Confusion matrix:
##           nonspam spam class.error
## nonspam      2725   63  0.02259684
## spam          308 1505  0.16988417
```

Remarque

L'erreur OOB est de 8.06%, elle est de 4.52% lorsque $m = 7$.

- Un des reproches souvent fait aux forêts est l'aspect **boîte noire** et **manque d'interprétabilité** par rapport aux modèles paramétriques tels que le modèle logistique.

Importance des variables

- Un des reproches souvent fait aux forêts est l'aspect **boîte noire** et **manque d'interprétabilité** par rapport aux modèles paramétriques tels que le modèle logistique.
- Il existe un **indicateur** qui permet de mesurer l'**importance des variables** présentes dans le modèle.
- Comme l'erreur OOB, ce critère est basé sur le fait que toutes les observations **ne sont pas utilisées** pour construire les arbres de la forêt.

- Soit OOB_k l'échantillon Out Of Bag associé au k^{eme} arbre : il contient les observations qui ne sont pas dans le k^{eme} échantillon bootstrap.

- Soit OOB_k l'échantillon Out Of Bag associé au k^{eme} arbre : il contient les observations qui ne sont pas dans le k^{eme} échantillon bootstrap.
- Soit E_{OOB_k} l'erreur de prédiction de l'arbre k mesurée sur cet échantillon :

$$E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2.$$

- Soit OOB_k l'échantillon Out Of Bag associé au k^{eme} arbre : il contient les observations qui **ne sont pas** dans le k^{eme} échantillon bootstrap.
- Soit E_{OOB_k} l'erreur de prédiction de l'arbre k mesurée sur cet échantillon :

$$E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2.$$

- Soit OOB_k^j l'échantillon OOB_k dans lequel on a **perturbé aléatoirement** les valeurs de la variable j et $E_{OOB_k^j}$ **l'erreur de prédiction** de l'arbre k mesurée sur cet échantillon :

$$E_{OOB_k^j}^j = \frac{1}{|OOB_k^j|} \sum_{i \in OOB_k^j} (T(X_i^j, \theta_k, \mathcal{D}_n) - Y_i)^2,$$

- Soit OOB_k l'échantillon Out Of Bag associé au k^{eme} arbre : il contient les observations qui ne sont pas dans le k^{eme} échantillon bootstrap.
- Soit E_{OOB_k} l'erreur de prédiction de l'arbre k mesurée sur cet échantillon :

$$E_{OOB_k} = \frac{1}{|OOB_k|} \sum_{i \in OOB_k} (T(X_i, \theta_k, \mathcal{D}_n) - Y_i)^2.$$

- Soit OOB_k^j l'échantillon OOB_k dans lequel on a perturbé aléatoirement les valeurs de la variable j et $E_{OOB_k^j}$ l'erreur de prédiction de l'arbre k mesurée sur cet échantillon :

$$E_{OOB_k^j}^j = \frac{1}{|OOB_k^j|} \sum_{i \in OOB_k^j} (T(X_i^j, \theta_k, \mathcal{D}_n) - Y_i)^2,$$

Définition

L'importance de la j^{eme} variable est définie par

$$Imp(X_j) = \frac{1}{B} \sum_{k=1}^B (E_{OOB_k^j}^j - E_{OOB_k}).$$

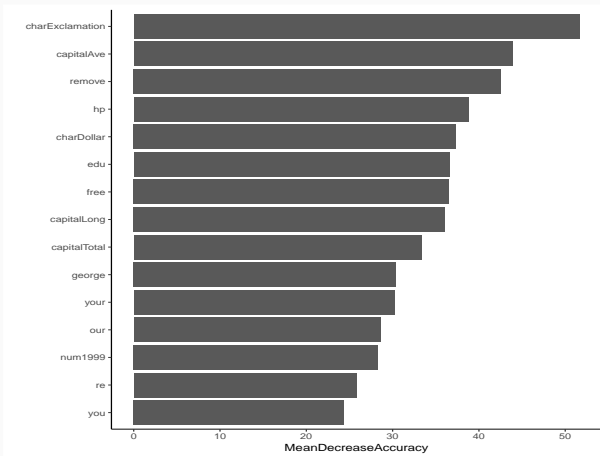
Exemple

- L'importance s'obtient facilement avec le package `randomForest`

```
> foret <- randomForest(type~.,data=spam,importance=TRUE)
> Imp <- importance(foret,type=1) %>% as.data.frame() %>%
+   mutate(variable=names(spam)[-58]) %>% arrange(desc(MeanDecreaseAccuracy))
> head(Imp)
```

##	MeanDecreaseAccuracy	variable
## 1	52.88382	charExclamation
## 2	48.69346	remove
## 3	42.21059	capitalAve
## 4	38.86023	charDollar
## 5	38.85976	hp
## 6	37.77500	capitalLong

```
> ggplot(Imp[1:15,]) + aes(x=reorder(variable, MeanDecreaseAccuracy),
+                           y=MeanDecreaseAccuracy)+
+   geom_bar(stat="identity")+coord_flip()+xlab("")+theme_classic()
```



Bagging et forêts aléatoires

Bagging

Forêts aléatoires

Boosting

Algorithmes de gradient boosting

Choix des paramètres

Bibliographie

- Le terme **Boosting** s'applique à des méthodes générales permettant de produire des décisions précises à partir de **règles faibles** (weaklearner).
- Historiquement, le **premier** algorithme boosting est **adaboost** [Freund and Schapire, 1996].
- Il a ensuite été montré que cet algorithme peut-être vu comme **un cas particulier d'algorithmes** de descente de gradient \implies **gradient boosting**.
- C'est cette **famille d'algorithmes** que nous présentons dans cette partie.

Bagging et forêts aléatoires

Bagging

Forêts aléatoires

Boosting

Algorithmes de gradient boosting

Choix des paramètres

Bibliographie

- (X, Y) couple aléatoire à valeurs dans $\mathbb{R}^d \times \mathcal{Y}$. Etant 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

$$\mathcal{R}(g) = \mathbf{E}[\ell(Y, g(X))].$$

Problème : la fonction de perte n'est pas calculable.

- (X, Y) couple aléatoire à valeurs dans $\mathbb{R}^d \times \mathcal{Y}$. Etant 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

$$\mathcal{R}(g) = \mathbf{E}[\ell(Y, g(X))].$$

Problème : la fonction de perte n'est pas calculable.

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

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

- On considère \mathcal{G} l'ensemble des arbres binaires et on veut trouver la meilleure combinaison linéaire d'abres binaires.

Un premier problème

Trouver $g(x) = \sum_{m=1}^M \alpha_m h_m(x) \in \mathcal{G}$ qui minimise

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

- On considère \mathcal{G} l'ensemble des arbres binaires et on veut trouver la meilleure combinaison linéaire d'arbres binaires.

Un premier problème

Trouver $g(x) = \sum_{m=1}^M \alpha_m h_m(x) \in \mathcal{G}$ qui minimise

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

sans solution...

- Pas de solution explicite.
- Nécessité de trouver un algorithme pour approcher la solution.

Descente de gradient

- **Idée** : utiliser un algorithme de **descente de gradient** de type **Newton-Raphson**.

Descente de gradient

- Idée : utiliser un algorithme de descente de gradient de type Newton-Raphson.
- Algorithme récursif :
 - itération m : g_m
 - itération $m + 1$: on ajoute à g_m un nouvel arbre binaire h_{m+1} tel que le risque $\mathcal{R}_n(g_m + \lambda h_{m+1})$ diminue le plus fortement ($\lambda \in \mathbb{R}^+$ petit).

Descente de gradient

- **Idée** : utiliser un algorithme de **descente de gradient** de type **Newton-Raphson**.
- **Algorithme récursif** :
 - **itération m** : g_m
 - **itération $m + 1$** : on ajoute à g_m un nouvel arbre binaire h_{m+1} tel que le risque $\mathcal{R}_n(g_m + \lambda h_{m+1})$ diminue le plus fortement ($\lambda \in \mathbb{R}^+$ petit).
 - **Approche classique** : utiliser l'opposé du gradient
$$f_{m+1}(x) = -\nabla \mathcal{R}_n(g_m)(x)$$

$$g_{m+1}(x) = g_m(x) + \lambda f_{m+1}(x).$$

Descente de gradient

- **Idée** : utiliser un algorithme de **descente de gradient** de type **Newton-Raphson**.
- **Algorithme récursif** :
 - **itération m** : g_m
 - **itération $m + 1$** : on ajoute à g_m un nouvel arbre binaire h_{m+1} tel que le risque $\mathcal{R}_n(g_m + \lambda h_{m+1})$ diminue le plus fortement ($\lambda \in \mathbb{R}^+$ petit).
 - **Approche classique** : utiliser l'opposé du gradient
$$f_{m+1}(x) = -\nabla \mathcal{R}_n(g_m)(x)$$

$$g_{m+1}(x) = g_m(x) + \lambda f_{m+1}(x).$$

Une restriction

Chaque élément de la suite doit être une **combinaison d'arbres** et f_{m+1} n'est pas (forcément) un arbre.

- Pour trouver l'arbre le plus proche du gradient f_{m+1} , on cherche un arbre h qui minimise

$$\frac{1}{n} \sum_{i=1}^n (f_{m+1}(X_i) - h(X_i))^2.$$

- Si on désigne par

$$U_i = f_{m+1}(X_i) = -\nabla \mathcal{R}_n(g_m)(X_i) = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_m(x_i)},$$

la solution h_{m+1} s'obtient en ajustant un arbre sur l'échantillon $(X_1, U_1) \dots, (X_n, U_n)$.

- Pour trouver l'arbre le plus proche du gradient f_{m+1} , on cherche un arbre h qui minimise

$$\frac{1}{n} \sum_{i=1}^n (f_{m+1}(X_i) - h(X_i))^2.$$

- Si on désigne par

$$U_i = f_{m+1}(X_i) = -\nabla \mathcal{R}_n(g_m)(X_i) = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_m(x_i)},$$

la solution h_{m+1} s'obtient en ajustant un arbre sur l'échantillon $(X_1, U_1) \dots, (X_n, U_n)$.

Mise à jour de l'algorithme

$$g_{m+1}(x) = g_m(x) + \lambda h_{m+1}(x).$$

Gradient Boost Algorithm [Friedman, 2001] : FGD

- $d_n = (x_1, y_1), \dots, (x_n, y_n)$ l'échantillon, λ un paramètre de régularisation tel que $0 < \lambda \leq 1$.
- M le nombre d'itérations.
- paramètres de l'arbre (nombre de coupures...)

1. Initialisation : $g_0(.) = \operatorname{argmin}_c \frac{1}{n} \sum_{i=1}^n \ell(y_i, c)$

2. **Pour** $m = 1$ à M :

a) Calculer l'opposé du gradient $-\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i))$ et l'évaluer aux points $g_{m-1}(x_i)$:

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)}, \quad i = 1, \dots, n.$$

b) **Ajuster un arbre** sur l'échantillon $(x_1, U_1), \dots, (x_n, U_n)$, on note h_m l'arbre ainsi défini.

c) **Mise à jour** : $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$.

3. **Sortie** : la suite de règles $(g_m(x))_m$.

- Il est facile de voir que chaque règle g_m s'écrit

$$g_m(x) = g_0 + \lambda \sum_{k=1}^m h_k(x)$$

où les h_k sont des arbres binaires.

\implies ces règles sont donc bien des combinaisons d'arbres.

- Il est facile de voir que **chaque règle** g_m s'écrit

$$g_m(x) = g_0 + \lambda \sum_{k=1}^m h_k(x)$$

où les h_k sont des arbres binaires.

\implies ces règles sont donc bien des **combinaisons d'arbres**.

- Plusieurs **paramètres** sont à choisir ou à calibrer par l'**utilisateur** :
 - fonction de perte ℓ ;
 - nombre d'itérations M ;
 - paramètre de régularisation λ ;
 - paramètres de l'arbre (nombre de coupures notamment).

Bagging et forêts aléatoires

Bagging

Forêts aléatoires

Boosting

Algorithmes de gradient boosting

Choix des paramètres

Bibliographie

Choix de la fonction de perte

- La fonction de perte $\ell(y, g(x))$ doit remplir 2 conditions :
 1. mesurer une **erreur** : prendre des valeurs élevées lorsque y est loin de $g(x)$ et faibles dans le cas inverse.
 2. être régulière : notamment **dérivable et convexe** en son second argument.

Choix de la fonction de perte

- La fonction de perte $\ell(y, g(x))$ doit remplir 2 conditions :
 1. mesurer une **erreur** : prendre des valeurs élevées lorsque y est loin de $g(x)$ et faibles dans le cas inverse.
 2. être régulière : notamment **dérivable et convexe** en son second argument.

Exemple

- **Classification binaire** avec Y dans $\{-1, 1\}$:
 1. $\ell(y, g(x)) = \exp(-yg(x)) \implies$ **adaboost** ;
 2. $\ell(y, g(x)) = \log(1 + \exp(-2yg(x))) \implies$ **logitboost** ;

Choix de la fonction de perte

- La fonction de perte $\ell(y, g(x))$ doit remplir 2 conditions :
 1. mesurer une **erreur** : prendre des valeurs élevées lorsque y est loin de $g(x)$ et faibles dans le cas inverse.
 2. être régulière : notamment **dérivable et convexe** en son second argument.

Exemple

- **Classification binaire** avec Y dans $\{-1, 1\}$:
 1. $\ell(y, g(x)) = \exp(-yg(x)) \implies$ **adaboost** ;
 2. $\ell(y, g(x)) = \log(1 + \exp(-2yg(x))) \implies$ **logitboost** ;
- **Régression** avec Y dans \mathbb{R} : $\ell(y, g(x)) = 0.5(y - g(x))^2 \implies$ **L_2 -boosting**.

Remarque

- Pour le L_2 -boosting, les U_i de l'étape 2.a de l'algorithme FGD s'écrivent

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)} = y_i - g_{m-1}(x_i).$$

Remarque

- Pour le L_2 -boosting, les U_i de l'étape 2.a de l'algorithme FGD s'écrivent

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)} = y_i - g_{m-1}(x_i).$$

- Ces quantités correspondent aux résidus du régresseur à l'étape $m - 1$.

Remarque

- Pour le L_2 -boosting, les U_i de l'étape 2.a de l'algorithme FGD s'écrivent

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)} = y_i - g_{m-1}(x_i).$$

- Ces quantités correspondent aux résidus du régresseur à l'étape $m - 1$.

Interprétation

- L'estimateur à l'étape m est construit en faisant une régression sur les résidus correspondants à l'estimateur à l'étape $m - 1$.
- On "corrige" g_{m-1} en cherchant à expliquer "l'information restante" qui est contenue dans les résidus.

- L'algorithme L_2 -boosting (simplifié) peut alors s'écrire.

L_2 -boosting - autre version

1. Initialisation g_0 .
2. Pour $m = 1$ à M :
 - a) Calculer les résidus $U_i = y_i - g_{m-1}(x_i)$, $i = 1, \dots, n$.
 - b) Ajuster la règle faible sur $(x_1, U_1), \dots, (x_n, U_n) \implies h_m$.
 - c) Mise à jour : $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$.

- L'algorithme L_2 -boosting (simplifié) peut alors s'écrire.

L_2 -boosting - autre version

1. Initialisation g_0 .
2. Pour $m = 1$ à M :
 - a) Calculer les résidus $U_i = y_i - g_{m-1}(x_i)$, $i = 1, \dots, n$.
 - b) Ajuster la règle faible sur $(x_1, U_1), \dots, (x_n, U_n) \implies h_m$.
 - c) Mise à jour : $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$.

Remarque importante [Bühlmann and Yu, 2003, Bühlmann and Hothorn, 2007, Cornillon et al., 2014]

Il a été montré (sous certaines hypothèses) qu'à chaque itération :

- Le biais **diminue** : $B(g_m) \leq B(g_{m-1})$.
- La variance **augmente** $V(g_m) \geq V(g_{m-1})$.

- L'algorithme L_2 -boosting (simplifié) peut alors s'écrire.

L_2 -boosting - autre version

1. Initialisation g_0 .
2. Pour $m = 1$ à M :
 - a) Calculer les résidus $U_i = y_i - g_{m-1}(x_i)$, $i = 1, \dots, n$.
 - b) Ajuster la règle faible sur $(x_1, U_1), \dots, (x_n, U_n) \Rightarrow h_m$.
 - c) Mise à jour : $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$.

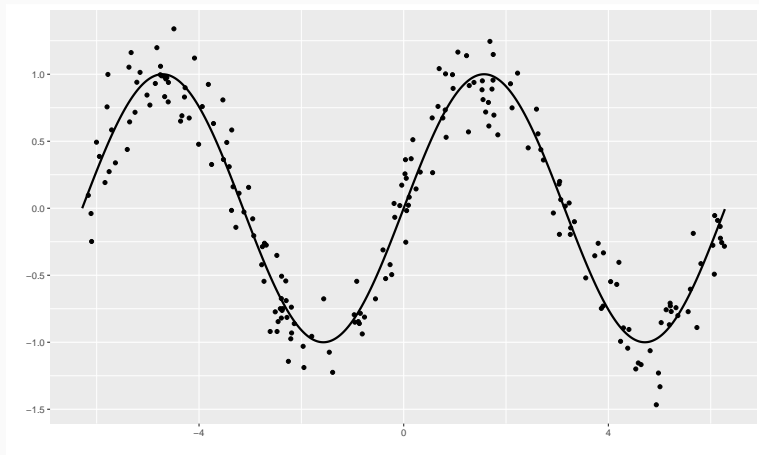
Remarque importante [Bühlmann and Yu, 2003, Bühlmann and Hothorn, 2007, Cornillon et al., 2014]

Il a été montré (sous certaines hypothèses) qu'à chaque itération :

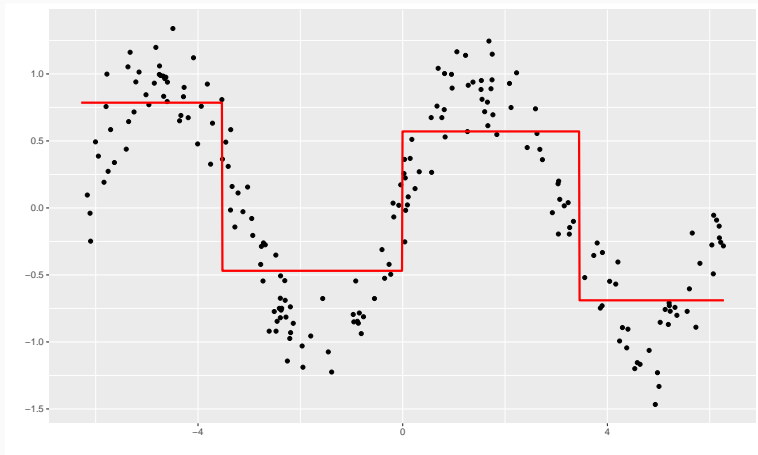
- Le biais **diminue** : $B(g_m) \leq B(g_{m-1})$.
- La variance **augmente** $V(g_m) \geq V(g_{m-1})$.
- D'où l'importance d'utiliser des règles **faibles** : **beaucoup de biais** et peu de variance (des arbres avec peu de nœuds terminaux par exemple).

Illustration

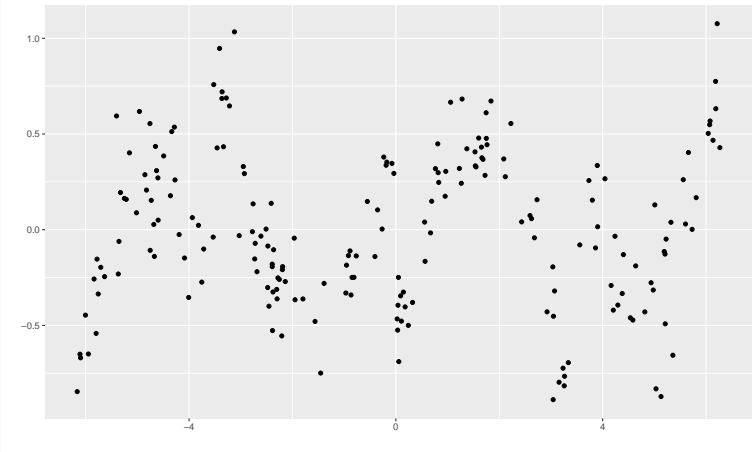
On considère l'échantillon suivant



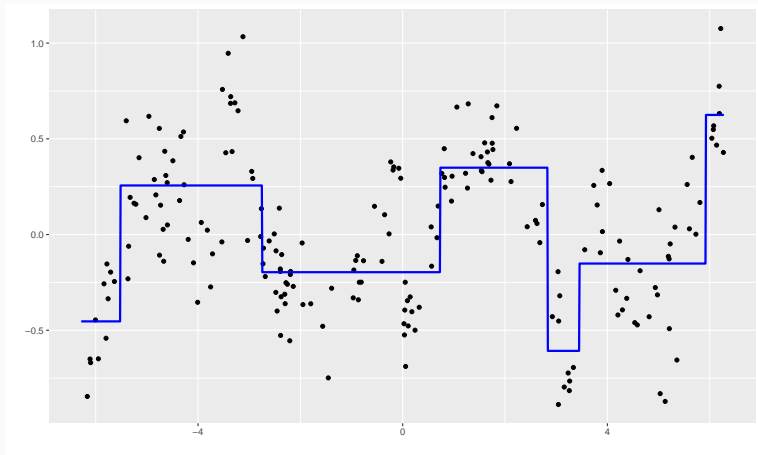
On ajuste un premier arbre simple :



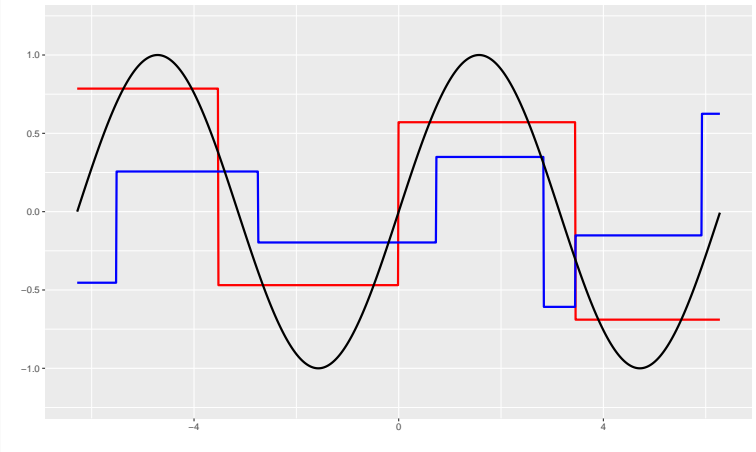
On calcule les résidus :



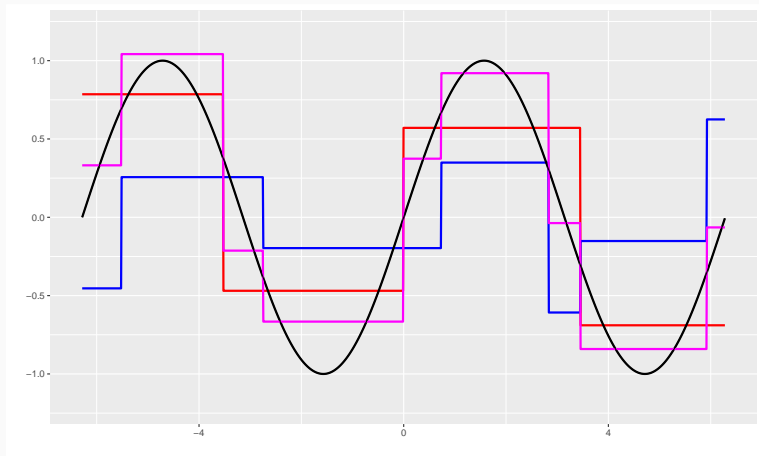
On ajuste un nouvel arbre sur les résidus :



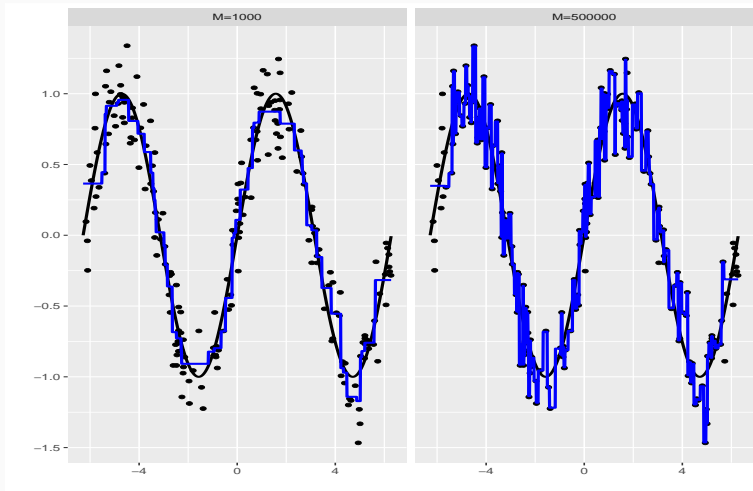
On obtient ainsi 2 arbres :



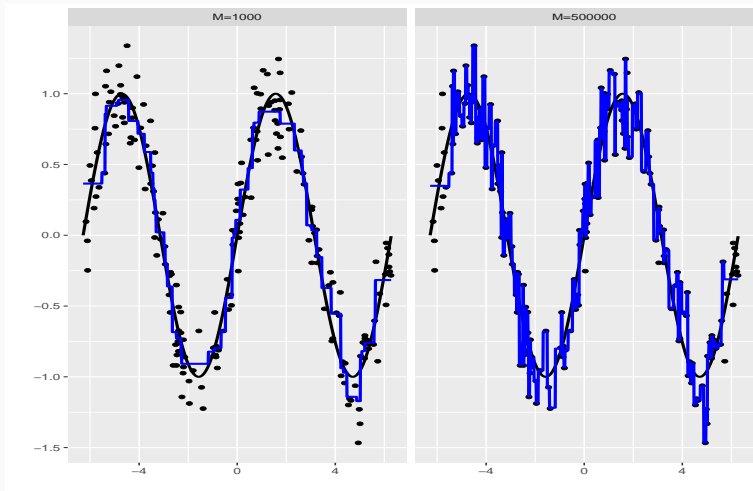
Que l'on ajoute pour déduire un nouvel estimateur (2 itérations)...



- Pour 1,000 et 500,000 itérations, on obtient :



- Pour 1,000 et 500,000 itérations, on obtient :



Remarque importante

L'algorithme **sur-ajuste** si le **nombre d'itérations est (trop) grand**.

Choix de m et λ

- Le choix du nombre d'itérations est **crucial** pour les estimateurs boosting.

Choix de m et λ

- Le choix du nombre d'itérations est **crucial** pour les estimateurs boosting.
- Si m est trop **grand** on **sur-ajuste** (estimateurs avec peu de biais mais beaucoup de variance) et réciproquement si m est trop petit.

Choix de m et λ

- Le choix du nombre d'itérations est **crucial** pour les estimateurs boosting.
- Si m est trop **grand** on **sur-ajuste** (estimateurs avec peu de biais mais beaucoup de variance) et réciproquement si m est trop petit.
- Le **paramètre de régularisation** λ représente le **pas de la descente de gradient**.
- Ce paramètre est **lié à m** : un λ grand nécessitera peu d'itérations et réciproquement.

Choix de m et λ

- Le choix du nombre d'itérations est **crucial** pour les estimateurs boosting.
- Si m est trop **grand** on **sur-ajuste** (estimateurs avec peu de biais mais beaucoup de variance) et réciproquement si m est trop petit.
- Le **paramètre de régularisation** λ représente le **pas de la descente de gradient**.
- Ce paramètre est **lié à m** : un λ grand nécessitera peu d'itérations et réciproquement.

En pratique

- On considère 2 ou 3 valeurs (petites) pour λ (0.1, 0.01) ;
- Pour chaque λ , on choisit le meilleur m en utilisant des techniques de type **validation croisée**.

- L'algorithme étant construit pour une fonction de perte ℓ donnée, il est d'usage d'utiliser la même fonction de perte pour sélectionner m .
- On va donc chercher le nombre d'itérations qui minimise la perte :

$$\hat{m} = \operatorname{argmin}_{m \leq M} \mathbf{E}[\ell(Y, g_m(X))].$$

- L'algorithme étant construit pour une fonction de perte ℓ donnée, il est d'usage d'utiliser la même fonction de perte pour sélectionner m .
- On va donc chercher le nombre d'itérations qui minimise la perte :

$$\hat{m} = \operatorname{argmin}_{m \leq M} \mathbf{E}[\ell(Y, g_m(X))].$$

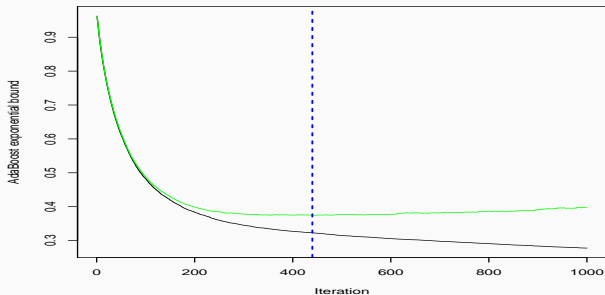
- L'espérance ci-dessus étant inconnue en pratique, elle est approchée par des algorithmes de validation croisée.

- La fonction `gbm` du package `gbm` [Ridgeway, 2006] permet de faire du gradient boosting. Elle admet notamment comme paramètres :
 1. fonction de perte (`distribution`)
 2. nombre d'itérations maximal M (`n.trees`)
 3. nombre de nœuds terminaux des arbres plus 1 (`interaction.depth`)
 4. paramètre de régularisation λ (`shrinkage`)
 5. paramètres pour la validation croisée (`train.fraction` pour la validation hold out ou `cv.folds` pour la validation croisée).

- La fonction `gbm` du package `gbm` [Ridgeway, 2006] permet de faire du gradient boosting. Elle admet notamment comme paramètres :
 1. fonction de perte (`distribution`)
 2. nombre d'itérations maximal M (`n.trees`)
 3. nombre de nœuds terminaux des arbres plus 1 (`interaction.depth`)
 4. paramètre de régularisation λ (`shrinkage`)
 5. paramètres pour la validation croisée (`train.fraction` pour la validation hold out ou `cv.folds` pour la validation croisée).
- La fonction `gbm.perf` permet de sélectionner le nombre d'itérations.

Exemple

```
> library(gbm)
> set.seed(1234)
> spam1 <- spam
> spam1$type <- as.numeric(spam1$type)-1
> ada <- gbm(type~.,data=spam1,distribution="adaboost",cv.folds=5,
+           n.trees=1000,shrinkage=0.05)
> mopt <- gbm.perf(ada)
> mopt
## [1] 440
```



Conclusion

- Les algorithmes **randomforest** et **boosting** agrègent des arbres :

$$\hat{g}_m(x) = \sum_{k=1}^m \alpha_k h_k(x).$$

- Pour être efficace les arbres h_k doivent être des **règles faibles** (**weaklearner**), donc des arbres **peu performants** :

Conclusion

- Les algorithmes **randomforest** et **boosting** agrègent des arbres :

$$\hat{g}_m(x) = \sum_{k=1}^m \alpha_k h_k(x).$$

- Pour être efficace les arbres h_k doivent être des **règles faibles** (**weaklearner**), donc des arbres **peu performants** :
 - **randomforest** : arbres **très profonds** avec beaucoup de variance et peu de biais ;
 - **boosting** : arbres **peu profonds** avec peu de variance et beaucoup de biais.

Conclusion

- Les algorithmes **randomforest** et **boosting** agrègent des arbres :

$$\hat{g}_m(x) = \sum_{k=1}^m \alpha_k h_k(x).$$

- Pour être efficace les arbres h_k doivent être des **règles faibles** (**weaklearner**), donc des arbres **peu performants** :
 - **randomforest** : arbres **très profonds** avec beaucoup de variance et peu de biais ;
 - **boosting** : arbres **peu profonds** avec peu de variance et beaucoup de biais.

Résumé

- Agrégation RF : réduction de **variance** ;
- Agrégation **boosting** : réduction de **biais**.

Bagging et forêts aléatoires

Bagging

Forêts aléatoires

Boosting

Algorithmes de gradient boosting

Choix des paramètres

Bibliographie



Aronszajn, N. (1950).

Theory of reproducing kernels.

Transactions of the American Mathematical Society, 68 :337–404.



Breiman, L. (1996).

Bagging predictors.




Machine Learning, 26(2) :123–140.



Bühlmann, P. and Hothorn, T. (2007).

Boosting algorithms : regularization, prediction and model fitting.

Statistical Science, 22 :477–505.

-  Bühlmann, P. and Yu, B. (2003).
Boosting with the l_2 loss : regression and classification.
Journal of American Statistical Association, 98 :324–339.
-  Cornillon, P., Hengartner, N., and Matzner-Løber, E. (2014).
Recursive bias estimation for multivariate regression smoothers.
ESAIM : Probability and Statistics, 18(483-502).
-  Devroye, L. and Krzyżak, A. (1989).
An equivalence theorem for l_1 convergence of the kernel regression estimate.
Journal of statistical Planning Inference, 23 :71–82.



Freund, Y. and Schapire, R. (1996).

Experiments with a new boosting algorithm.

In *Proceedings of the Thirteenth International Conference on Machine Learning*.



Freund, Y. and Schapire, R. (1997).

A decision-theoretic generalization of online learning and an application to boosting.

Journal of Computer and System Sciences, 55 :119–139.



Freund, Y. and Schapire, R. (1999).

A short introduction to boosting.

Journal of Japanese Society for Artificial Intelligence, 14(5) :771–780.



Friedman, J. H. (2001).

Greedy function approximation : A gradient boosting machine.
Annals of Statistics, 29 :1189–1232.



Fromont, M. (2015).

Apprentissage statistique.





Université Rennes 2, diapos de cours.



Genuer, R. (2010).

Forêts aléatoires : aspects théoriques, sélection de variables et applications.

PhD thesis, Université Paris XI.

-  Györfi, L., Kohler, M., Krzyzak, A., and Harro, W. (2002).
A Distribution-Free Theory of Nonparametric Regression.
Springer.
-  Hastie, T., Tibshirani, R., and Friedman, J. (2009).
The Elements of Statistical Learning : Data Mining, Inference, and Prediction.
Springer, second edition.
-  Ridgeway, G. (2006).
Generalized boosted models : A guide to the gbm package.
-  Stone, C. J. (1977).
Consistent nonparametric regression.
Annals of Statistics, 5 :595–645.



Vapnik, V. (2000).

The Nature of Statistical Learning Theory.

Springer, second edition.



Vert, J. (2014).

Support vector machines and applications in computational biology.

disponible à l'url <http://cbio.ensmp.fr/~jvert/svn/kernelcourse/slides/kernel2h/kernel2h.pdf>.

Cinquième partie V

Réseaux de neurones

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

- Wikistat : Neural networks and introduction to deep learning
- Eric Rakotomalala : Deep learning : Tensorflow et Keras sous R
- Rstudio : R interface to Keras

- Modélisation du **neurone formel** [McCulloch and Pitts, 1943].
- Concept mis en **réseau** avec **une couche d'entrée et une sortie** [Rosenblatt, 1958].
 - Origine du **perceptron**
 - Approche **connexioniste** (atteint ses limites technologiques et théoriques au début des années 70)
- Relance de l'approche connexioniste au début des années 80 avec l'essor technologique et quelques avancées théoriques
- Estimation du **gradient** par **rétro-propagation de l'erreur** [Rumelhart et al., 1986].

- Développement considérable (au début des années 90)
- Remis en veilleuse au milieu des années 90 au profit d'autres algorithmes d'apprentissage : boosting, support vector machine...
- Regain d'intérêt dans les années 2010, énorme battage médiatique sous l'appellation d'apprentissage profond/deep learning.
- Résultats spectaculaires obtenus par ces réseaux en reconnaissance d'images, traitement du langage naturel...

Il existe différents types de réseaux neuronaux :

- **perceptron multicouches** : les plus anciens et les plus simples ;
- **réseaux de convolution** : particulièrement efficaces pour le traitement d'images ;
- **réseaux récurrents** : adaptés à des données séquentielles (données textuelles, séries temporelles).

Différentes architectures

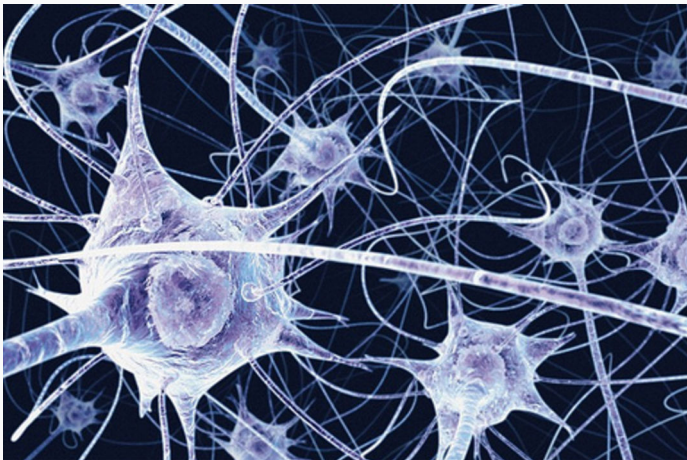
Il existe différents types de réseaux neuronaux :

- **perceptron multicouches** : les plus anciens et les plus simples ;
- **réseaux de convolution** : particulièrement efficaces pour le traitement d'images ;
- **réseaux récurrents** : adaptés à des données séquentielles (données textuelles, séries temporelles).

Dans cette partie

nous nous intéresserons uniquement au **perceptron multicouches**.

Neurone : vision biologique



Définition : neurone biologique

Un neurone biologique est une cellule qui se caractérise par

- des **synapses** : les points de **connexion** avec les autres neurones ;
- **dentrées** : **entrées** du neurones ;
- les **axones** ou **sorties** du neurone vers d'autres neurones ;
- le **noyau** qui **active** les sorties.

Définition : neurone biologique

Un neurone biologique est une cellule qui se caractérise par

- des **synapses** : les points de **connexion** avec les autres neurones ;
- des **dendrites** : **entrées** du neurones ;
- les **axones** ou **sorties** du neurone vers d'autres neurones ;
- le **noyau** qui **active** les sorties.

Définition : neurone formel

Un **neurone formel** est un modèle qui se caractérise par

- des **entrées** x_1, \dots, x_p ;
- des **poids** w_0, w_1, \dots, w_p ;
- une **fonction d'activation** $\sigma : \mathbb{R} \rightarrow \mathbb{R}$;
- une **sortie** :

$$\hat{y} = \sigma(w_0 + w_1x_1 + \dots + x_px_p).$$

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

- Le problème : expliquer une sortie $y \in \mathbb{R}$ par des entrées $x = (x_1, \dots, x_p)$.

- Le problème : expliquer une sortie $y \in \mathbb{R}$ par des entrées $x = (x_1, \dots, x_p)$.

Définition

Le perceptron simple est une fonction f des entrées x

- pondérées par un vecteur $w = (w_1, \dots, w_p)$,
- complétées par un neurone de biais w_0 ,
- et une fonction d'activation $\sigma : \mathbb{R} \rightarrow \mathbb{R}$

$$\hat{y} = f(x) = \sigma(w_0 + w_1x_1 + \dots + x_px_p).$$

Plusieurs fonctions d'activation peuvent être utilisées :

- **Identité** : $\sigma(x) = x$;
- **sigmoïde** ou **logistique** : $\sigma(x) = 1/(1 + \exp(-x))$;
- **seuil** : $\sigma(x) = \mathbf{1}_{x \geq 0}$;
- **ReLU** (Rectified Linear Unit) : $\sigma(x) = \max(x, 0)$;
- **Radiale** : $\sigma(x) = \sqrt{1/2\pi} \exp(-x^2/2)$.

Fonction d'activation

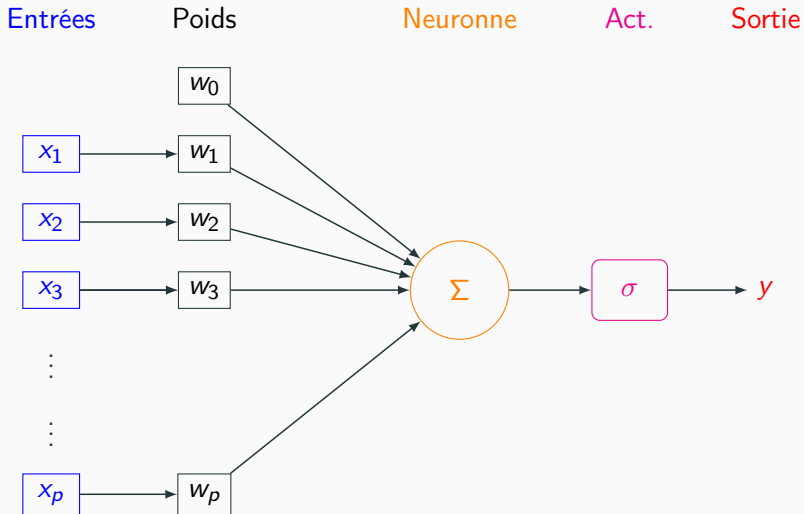
Plusieurs fonctions d'activation peuvent être utilisées :

- **Identité** : $\sigma(x) = x$;
- **sigmoïde** ou **logistique** : $\sigma(x) = 1/(1 + \exp(-x))$;
- **seuil** : $\sigma(x) = \mathbf{1}_{x \geq 0}$;
- **ReLU** (Rectified Linear Unit) : $\sigma(x) = \max(x, 0)$;
- **Radiale** : $\sigma(x) = \sqrt{1/2\pi} \exp(-x^2/2)$.

Remarque

Les **poids** w_j sont estimés à partir des **données** (voir plus loin).

Représentation graphique



- Plusieurs **packages R** permettent d'ajuster des réseaux de neurones : **nnet**, **deepnet**...
- Nous présentons ici le package **keras**, initialement programmé en **Python** et qui a été "traduit" récemment en **R**.

```
> library(keras)
> install_keras()
```

Exemple

- On veut expliquer une variable Y binaire par 4 variables d'entrées X_1, \dots, X_4 .
- On dispose d'un échantillon d'apprentissage de taille 300 :

```
> head(dapp)
```

##		X1	X2	X3	X4	Y
## 1		0.5855288	-1.4203239	1.67751179	-0.1746226	1
## 2		0.7094660	-2.4669386	0.07947405	-0.6706167	1
## 3		-0.1093033	0.4847158	-0.85642750	0.5074258	0
## 4		-0.4534972	-0.9379723	-0.77877729	1.2474343	0
## 5		0.6058875	3.3307333	-0.38093608	-1.2482755	1
## 6		-1.8179560	-0.1629455	-1.89735834	-1.9347187	1

Définition du modèle

- Elle s'effectue à l'aide des fonctions `keras_model_sequential` et `layer_dense`.

```
> model <- keras_model_sequential()  
> model %>% layer_dense(units=1,input_shape=c(4),  
+                       activation="sigmoid")
```

Définition du modèle

- Elle s'effectue à l'aide des fonctions `keras_model_sequential` et `layer_dense`.

```
> model <- keras_model_sequential()  
> model %>% layer_dense(units=1,input_shape=c(4),  
+                        activation="sigmoid")
```

- `units` : nombre de neurones souhaités ;
- `activation` : choix de la fonction d'activation.

Summary

- Un **summary** du modèle permet de visualiser le **nombre de paramètres** à estimer.

```
> summary(model)
## -----
## Layer (type)                Output Shape          Param #
## =====
## dense (Dense)              (None, 1)             5
## =====
## Total params: 5
## Trainable params: 5
## Non-trainable params: 0
## -----
```

- On indique dans la fonction `compile` la `fonction de perte` pour l'estimation des paramètres du modèle et le `critère de performance`

```
> model %>% compile(  
+   loss="binary_crossentropy",  
+   optimizer="adam",  
+   metrics="accuracy"  
+ )
```


Estimation

- On utilise la fonction `fit` pour entrainer le modèle

```
> Xtrain <- as.matrix(dapp[,1:4])  
> Ytrain <- dapp$Y  
> model %>% fit(x=Xtrain,y=Ytrain,epochs=300,batch_size=5)
```

- Et on obtient les poids avec `get_weights` :

```
> W <- get_weights(model)  
> W  
## [[1]]  
##           [,1]  
## [1,]  0.250867128  
## [2,]  0.092339918  
## [3,] -0.162947521  
## [4,] -0.005261241  
##  
## [[2]]  
## [1]  0.1739036
```

Visualisation du réseau

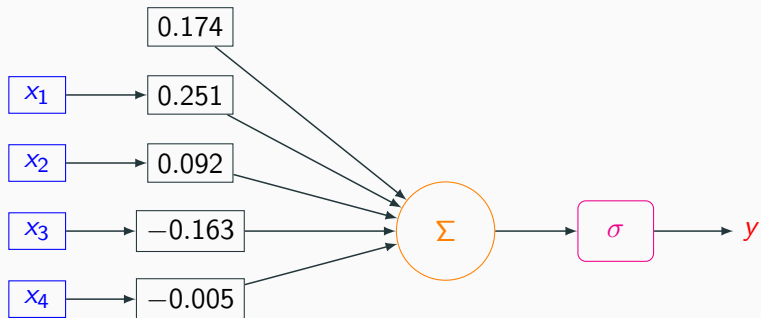
Entrées

Poids

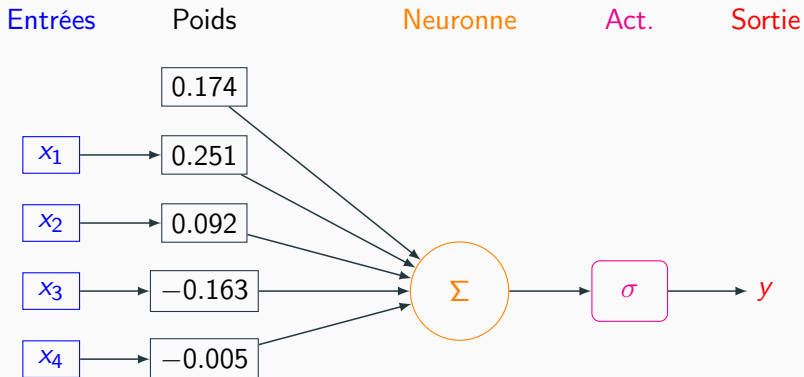
Neuronne

Act.

Sortie



Visualisation du réseau



Estimation

$$\hat{P}(Y = 1|X = x) = \frac{1}{1 + \exp(-(0.174 + 0.251x_1 + \dots - 0.005x_4))}$$

- On calcule la **prévision de la probabilité** $P(Y = 1|X = x)$ pour le premier individu de l'échantillon test :

```
> w <- W[[1]]  
> w0 <- W[[2]]  
> Xtest <- as.matrix(dtest[,1:4])  
> sc1 <- w0+sum(w*Xtest[1,])  
> 1/(1+exp(-sc1))  
## [1] 0.6209704
```

- On calcule la **prévision de la probabilité** $P(Y = 1|X = x)$ pour le premier individu de l'échantillon test :

```
> w <- W[[1]]
> w0 <- W[[2]]
> Xtest <- as.matrix(dtest[,1:4])
> sc1 <- w0+sum(w*Xtest[1,])
> 1/(1+exp(-sc1))
## [1] 0.6209704
```

- que l'on retrouve avec **predict_proba** :

```
> prev <- model %>% predict_proba(Xtest)
> prev[1]
## [1] 0.6209704
```

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

Constat

- Règle de classification : le **preceptron simple** affecte un individu dans le groupe 1 si

$$P(Y = 1|X = x) \geq 0.5 \iff w_0 + w_1x_1 + \dots + w_px_p \geq 0.$$

- Il s'agit donc d'une **règle linéaire**.

Constat

- Règle de classification : le preceptron simple affecte un individu dans le groupe 1 si

$$P(Y = 1|X = x) \geq 0.5 \iff w_0 + w_1x_1 + \dots + w_px_p \geq 0.$$

- Il s'agit donc d'une règle linéaire.

\implies peu efficace pour représenter des phénomènes "complexes".

Constat

- Règle de classification : le **preceptron simple** affecte un individu dans le groupe 1 si

$$P(Y = 1|X = x) \geq 0.5 \iff w_0 + w_1x_1 + \dots + w_px_p \geq 0.$$

- Il s'agit donc d'une **règle linéaire**.

\implies **peu efficace** pour représenter des **phénomènes "complexes"**.

Idée

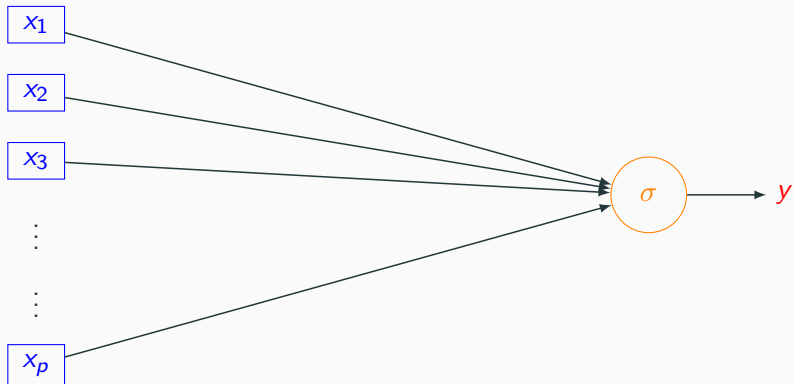
Conserver cette structure de réseau en considérant **plusieurs couches** de **plusieurs neurones**.

Perceptron simple

Entrées

C. Sortie

Sortie



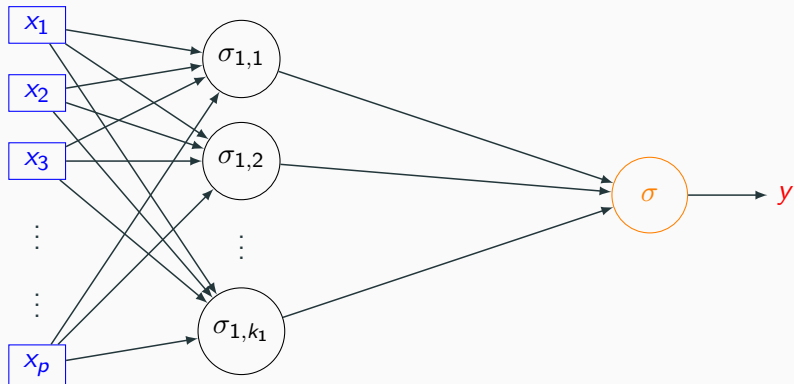
Une couche cachée

Entrées

C. cachée

C. Sortie

Sortie



Deux couches cachées

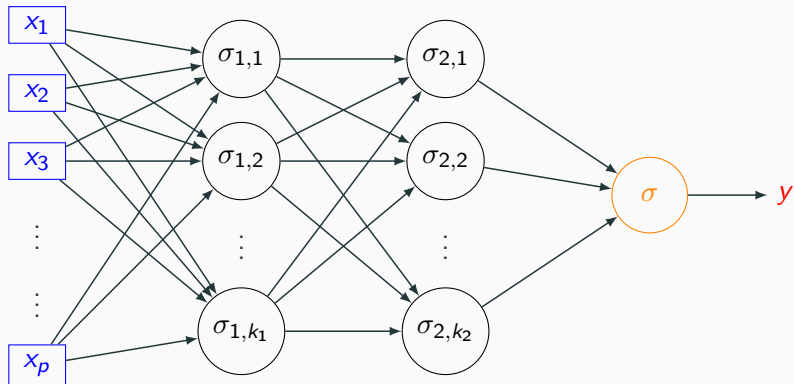
Entrées

C. cachée 1

C. cachée 2

C. Sortie

Sortie



- Les neurones de la première couche (cachée) calculent des combinaisons linéaires des entrées.
- Ces combinaisons linéaires sont ensuite activées par une fonction d'activation, produisant une sortie par neurone.

- Les neurones de la première couche (cachée) calculent des combinaisons linéaires des entrées.
- Ces combinaisons linéaires sont ensuite activées par une fonction d'activation, produisant une sortie par neurone.
- Chaque neurone de la deuxième couche (cachée) est une combinaison linéaire des sorties de la couche précédente...
- activées par une fonction d'activation, produisant une sortie par neurone...

Commentaires

- Les neurones de la **première couche (cachée)** calculent des **combinaisons linéaires des entrées**.
- Ces combinaisons linéaires sont ensuite **activées par une fonction d'activation**, produisant **une sortie par neurone**.
- Chaque neurone de la **deuxième couche (cachée)** est une combinaison linéaire des **sorties de la couche précédente...**
- **activées par une fonction d'activation**, produisant **une sortie par neurone...**

Remarque

Le nombre de neurones dans la **couche finale** est définie par la **dimension de la sortie y** :

- Régression ou classification binaire \implies 1 neurone.
- Classification multiclasse (K) $\implies K$ (ou $K - 1$) neurones.

- L'ajout de couches cachées dans `keras` est relativement simple.
- Il suffit de définir ces couches au moment de la spécification du modèle.

- L'ajout de couches cachées dans **keras** est relativement simple.
- Il suffit de définir ces couches au moment de la spécification du modèle.
- Par exemple, pour **deux couches cachées** avec 10 et 5 neurones, on utilisera :

```
> model <- keras_model_sequential()
> model %>% layer_dense(units=10,input_shape=c(4),activation="sigmoid") %>%
+   layer_dense(units=5,activation="sigmoid") %>%
+   layer_dense(units=1,activation="sigmoid")
```

```
> summary(model)
```

```
## -----  
## Layer (type)                Output Shape                Param #  
## =====  
## dense_1 (Dense)             (None, 10)                 50  
## -----  
## dense_2 (Dense)             (None, 5)                  55  
## -----  
## dense_3 (Dense)             (None, 1)                   6  
## =====  
## Total params: 111  
## Trainable params: 111  
## Non-trainable params: 0  
## -----
```

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

- L'utilisateur doit choisir le nombre de couches, le nombre de neurones par couche, les fonctions d'activation de chaque neurone.

- L'utilisateur doit choisir le nombre de couches, le nombre de neurones par couche, les fonctions d'activation de chaque neurone.
- Une fois ces paramètres choisis, il faut calculer (estimer) tous les vecteurs de poids dans tous les neurones.

- L'utilisateur doit choisir le nombre de couches, le nombre de neurones par couche, les fonctions d'activation de chaque neurone.
- Une fois ces paramètres choisis, il faut calculer (estimer) tous les vecteurs de poids dans tous les neurones.

L'approche

- On désigne par θ l'ensemble des paramètres à estimer $\implies f(x, \theta)$ la règle associée au réseau.
- Minimisation de risque empirique : minimiser

$$\mathcal{R}_n(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i, \theta))$$

où ℓ est une fonction de perte (classique).

Fonctions de perte

- Erreur quadratique (régression) :

$$\ell(y, f(x)) = (y - f(x))^2.$$

- Cross-entropy ou log-vraisemblance négative (classification binaire 0/1) :

$$\ell(y, p(x)) = -(y \log(p(x)) + (1 - y) \log(1 - p(x)))$$

où $p(x) = \mathbf{P}(Y = 1|X = x)$.

- Cross-entropy ou log-vraisemblance négative (classification multi-classes) :

$$\ell(y, p(x)) = - \sum_{k=1}^K \mathbf{1}_{y=k} \log(p_k(x))$$

où $p_k(x) = \mathbf{P}(Y = k|X = x)$.

Descente de gradient

- La solution s'obtient à l'aide de méthodes de type **descente de gradient** :

$$\theta^{\text{new}} = \theta^{\text{old}} - \varepsilon \nabla_{\theta} \mathcal{R}_n(\theta^{\text{old}}).$$

- Le réseau étant **structuré en couches**, la mise à jour des paramètres n'est pas directe.

Descente de gradient

- La solution s'obtient à l'aide de méthodes de type **descente de gradient** :

$$\theta^{\text{new}} = \theta^{\text{old}} - \varepsilon \nabla_{\theta} \mathcal{R}_n(\theta^{\text{old}}).$$

- Le réseau étant **structuré en couches**, la mise à jour des paramètres n'est pas directe.

Algorithme de rétropropagation (voir [ici](#))

1. **Etape forward** : calculer tous les poids associés à θ^{old} et stocker toutes les valeurs intermédiaires.
2. **Etape backward** :
 - 2.1 Calculer le gradient dans la couche de sortie.
 - 2.2 En déduire les gradients des couches cachées.

- L'algorithme de rétropropagation n'est généralement **pas appliqué sur l'ensemble des données**, mais sur des sous-ensemble de cardinaux m appelés **batch**.
- Cette approche est classique sur les **gros volumes de données** et permet de prendre en compte des **données séquentielles**.

Batch et epoch

- L'algorithme de rétropropagation n'est généralement **pas appliqué sur l'ensemble des données**, mais sur des sous-ensemble de cardinaux m appelés **batch**.
- Cette approche est classique sur les **gros volumes de données** et permet de prendre en compte des **données séquentielles**.
- Pour prendre en compte **toutes les données** sur une étape de la descente de gradient, on va donc appliquer n/m fois **l'algorithme de rétropropagation**.
- Une itération sur l'ensemble des données est appelée **epoch**.

Algorithme de rétropropagation stochastique

Algorithme

Entrées : ε (learning rate), m (taille des batches), nb (nombre d'epochs).

1. Pour $\ell = 1$ à nb
2. Partitionner aléatoire les données en n/m batch de taille $m \implies B_1, \dots, B_{n/m}$.
 - 2.1 Pour $j = 1$ à n/m
 - 2.1.1 Calculer les gradients sur le batch j avec l'algorithme de **rétropropagation** : ∇_{θ} .
 - 2.1.2 Mettre à jour les paramètres

$$\theta^{\text{new}} = \theta^{\text{old}} - \varepsilon \nabla_{\theta^{\text{old}}}.$$

Sorties : θ^{new} et $f(x, \theta^{\text{new}})$.

Choix des paramètres

- ϵ (pas de la descente de gradient), généralement petit. Existence de versions améliorées de l'algorithme précédent moins sensible à ce paramètre (RMSProp, Adam...).
- m (taille des batch) : généralement petit (pas trop en fonction du temps de calcul). L'utilisateur peut (doit) faire plusieurs essais.
- nb (nombre d'epoch), proche du nombre d'itérations en boosting \implies risque de surapprentissage si trop grand.

Choix des paramètres

- ϵ (pas de la descente de gradient), généralement petit. Existence de versions améliorées de l'algorithme précédent moins sensible à ce paramètre (RMSProp, Adam...).
- m (taille des batch) : généralement petit (pas trop en fonction du temps de calcul). L'utilisateur peut (doit) faire plusieurs essais.
- nb (nombre d'epoch), proche du nombre d'itérations en boosting \implies risque de surapprentissage si trop grand.

En pratique

Il est courant de visualiser l'évolution de la fonction de perte et/ou d'un critère de performance en fonction du nombre d'epoch.

Un exemple

- On considère un réseau à 2 couches cachées comportant 50 nœuds (2851 paramètres).

```
> model1 <- keras_model_sequential()
> model1 %>% layer_dense(units=50,input_shape=c(4),
+                       activation="sigmoid") %>%
+   layer_dense(units = 50,activation = "sigmoid") %>%
+   layer_dense(units = 1,activation = "sigmoid")
```

- On utilise
 - `crossentropy` comme perte.
 - `Adam` comme algorithme d'optimisation.
 - `accuracy` (taux de bien classés) comme mesure de performance.

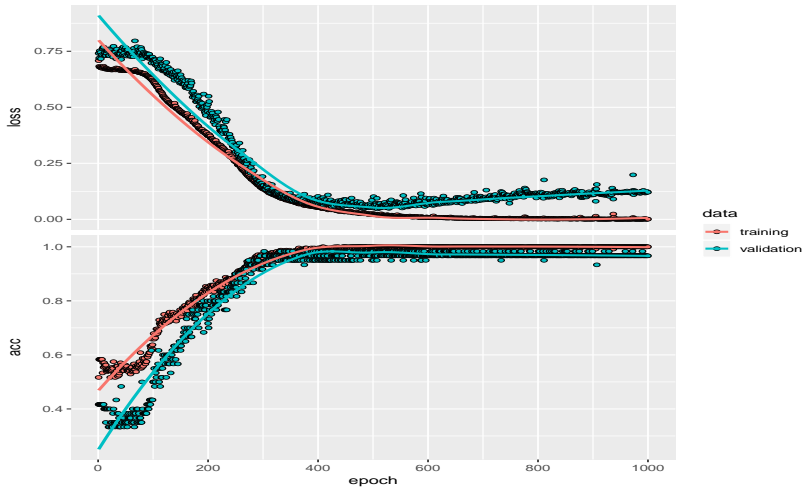
```
> model1 %>% compile(  
+   loss="binary_crossentropy",  
+   optimizer="adam",  
+   metrics="accuracy"  
+ )
```


- On **estime les paramètres** avec $m = 5$ et $nb = 1000$ et utilise 20% des données dans l'échantillon de validation.

```
> history <- model1 %>% fit(  
+   x=Xtrain,  
+   y=Ytrain,  
+   epochs=1000,  
+   batch_size=5,  
+   validation_split=0.2  
+ )
```

Erreur et perte

```
> plot(history)
```



- On compare ce nouveau réseau avec le perceptron simple construit précédemment.

```
> Xtest <- as.matrix(dtest[,1:4])
> Ytest <- dtest$Y
> model %>% evaluate(Xtest,Ytest)
## $loss
## [1] 0.7259337
##
## $acc
## [1] 0.39
> model1 %>% evaluate(Xtest,Ytest)
## $loss
## [1] 0.3290039
##
## $acc
## [1] 0.935
```

Nombre de couches et de neurones

- A choisir par l'utilisateur.
- Il est généralement mieux d'en avoir trop que pas assez \implies plus "facile" de capter des non linéarités complexes avec beaucoup de couches et de neurones.
- On fait généralement plusieurs essais que l'on compare (avec caret par exemple).
- Voir par exemple l'appli suivante :

<http://playground.tensorflow.org/>

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie

Surapprentissage

- Plusieurs paramètres peuvent causer du surapprentissage, notamment les nombres de couches cachées, de neurones et d'epoch.

Surapprentissage

- Plusieurs paramètres peuvent causer du surapprentissage, notamment les nombres de couches cachées, de neurones et d'epoch.

Plusieurs solutions

1. Régularisation de type ridge/lasso :

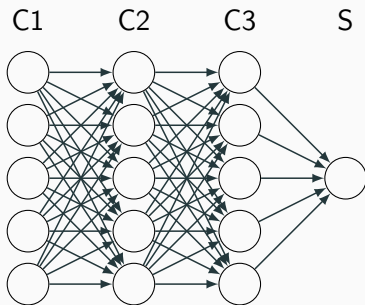
$$\mathcal{R}_n(\theta) = \frac{1}{n} \sum_{i=1}^n \ell(y_i, f(x_i, \theta)) + \lambda \Omega(\theta).$$

⇒ ajouter `kernel_regularizer = regularizer_l2(l = 0.001))` dans la fonction `layer_dense` par exemple.

2. Early stopping : on stoppe l'algorithme lorsque l'ajout d'epoch n'améliore pas suffisamment un critère donné.
3. Dropout : suppression (aléatoire) de certains neurones dans les couches
⇒ souvent la solution privilégiée.

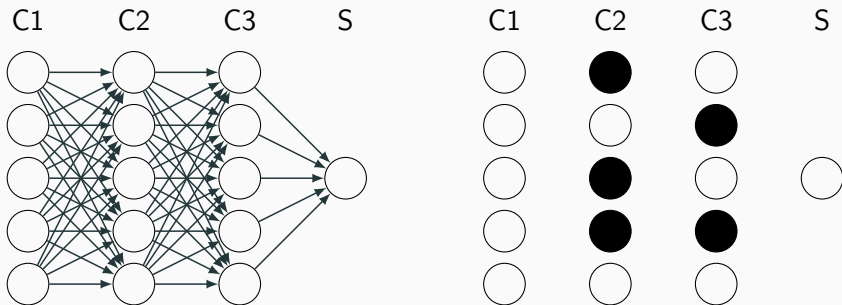
Dropout

- A chaque étape de la phase d'entraînement, on **supprime un nombre de neurones** (selon une Bernoulli de paramètre p).



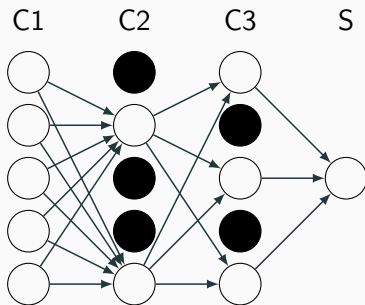
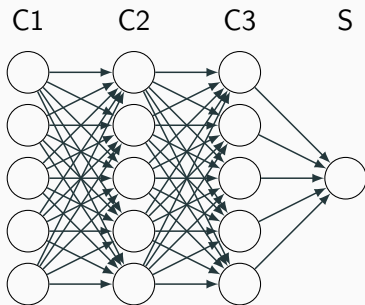
Dropout

- A chaque étape de la phase d'entraînement, on **supprime un nombre de neurones** (selon une Bernoulli de paramètre p).



Dropout

- A chaque étape de la phase d'entraînement, on **supprime un nombre de neurones** (selon une Bernoulli de paramètre p).



- Il suffit d'ajouter `layer_dropout` après les couches cachées.

```
> model3 <- keras_model_sequential()
> model3 %>% layer_dense(units=50,input_shape=c(4),activation="sigmoid") %>%
+   layer_dropout(0.5) %>%
+   layer_dense(units = 50,activation = "sigmoid") %>%
+   layer_dropout(0.5) %>%
+   layer_dense(units = 1,activation = "sigmoid")
```

Sélection avec caret

- On peut sélectionner la plupart des paramètres avec **caret**.
- On propose par exemple, pour un réseau avec une couche cachée, de choisir
 1. le nombre de **neurones dans la couche cachée** parmi 10, 50, 100
 2. la **fonction d'activation** : sigmoïde ou relu.

Sélection avec caret

- On peut sélectionner la plupart des paramètres avec **caret**.
- On propose par exemple, pour un réseau avec une couche cachée, de choisir
 1. le nombre de **neurones dans la couche cachée** parmi 10, 50, 100
 2. la **fonction d'activation** : sigmoïde ou relu.
- On définit d'abord les **paramètres du modèle**

```
> library(caret)
> dapp1 <- dapp
> dapp1$Y <- as.factor(dapp1$Y)
> param_grid <- expand.grid(size=c(10,50,100),
+                           lambda=0,batch_size=5,lr=0.001,
+                           rho=0.9,decay=0,
+                           activation=c("relu","sigmoid"))
```

- On calcule ensuite les taux de bien classés par validation croisée 5 blocs pour chaque combinaison de paramètres.

```
> caret_mlp <- train(Y~.,data=dapp1,method="mlpKerasDecay",  
+                   tuneGrid=param_grid,epoch=500,verbose=0,  
+                   trControl=trainControl(method="cv",number=5))
```

```

> caret_mlp
## Multilayer Perceptron Network with Weight Decay
## 300 samples
## 4 predictor
## 2 classes: '0', '1'
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 240, 240, 240, 240, 240
## Resampling results across tuning parameters:
##   size  activation  Accuracy  Kappa
##   10    relu       0.9200000  0.8394122
##   10    sigmoid    0.8966667  0.7913512
##   50    relu       0.9266667  0.8515286
##   50    sigmoid    0.9066667  0.8127427
##   100   relu       0.9366667  0.8722974
##   100   sigmoid    0.9300000  0.8595025
## Tuning parameter 'lambda' was held constant at a value of 0
## Tuning parameter 'rho' was held constant at a value of 0.9
## Tuning parameter 'decay' was held constant at a value of 0
## Accuracy was used to select the optimal model using the largest value.
## The final values used for the model were size = 100, lambda =
## 0, batch_size = 5, lr = 0.001, rho = 0.9, decay = 0 and activation = relu.

```

Conclusion

- **Avantages :**
 - Méthode connue pour être efficace pour (quasiment) tous les problèmes.
 - Plus particulièrement sur des architectures particulières : images, données textuelles.

Conclusion

- **Avantages :**
 - Méthode connue pour être efficace pour (quasiment) tous les problèmes.
 - Plus particulièrement sur des architectures particulières : images, données textuelles.
- **Inconvénients :**
 - Gain plus discutable sur des problèmes standards.
 - (Beaucoup) plus difficile à calibrer que les autres algorithmes ML.
 - Niveau d'expertise important.

Introduction

Le perceptron simple

Perceptron multicouches

Estimation

Choix des paramètres et surapprentissage

Bibliographie



McCulloch, W. and Pitts, W. (1943).

A logical calculus of ideas immanent in nervous activity.

Bulletin of Mathematical Biophysics, 5 :115–133.



Rosenblatt, F. (1958).

The perceptron : a probabilistic model for information storage and organization in the brain.

Psychological Review, 65 :386–408.



Rumelhart, D. E., Hinton, G. E., and R. J. Williams, R. J. (1986).

Learning representations by back-propagating errors.

Nature, pages 533–536.