

# Machine learning

L. Rouvière  
*laurent.rouviere@univ-rennes2.fr*

OCTOBRE 2020

## Présentation

- *Objectifs* : comprendre les aspects **théoriques** et **pratiques** des algorithmes machine learning de référence.
- *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** : energie, finance, marketing, sport.

## Programme

- *Matériel* :
  - slides : <https://lrouviere.github.io/INP-HB/>
  - Tutoriel (sans correction) : [https://lrouviere.github.io/TUTO\\_ML/](https://lrouviere.github.io/TUTO_ML/)
  - Tutoriel (avec corrections) : [https://lrouviere.github.io/TUTO\\_ML/correction/](https://lrouviere.github.io/TUTO_ML/correction/)
- *2 parties*
  1. **Rappels machine learning** : notions de risque, estimation de risque, complexité/surapprentissage...
  2. **Support vector machine**.

## Table des matières

<b>I Apprentissage : contexte et formalisation</b>	<b>3</b>
<b>1 Motivations</b>	<b>3</b>
<b>2 Cadre mathématique pour l'apprentissage supervisé</b>	<b>6</b>
<b>3 Exemples de fonction de perte</b>	<b>8</b>
<b>4 Estimation du risque</b>	<b>9</b>
<b>5 Le sur-apprentissage</b>	<b>11</b>

<b>6</b>	<b>Le package caret</b>	<b>14</b>
<b>7</b>	<b>Annexe : compléments sur les scores</b>	<b>17</b>
<b>8</b>	<b>Bibliographie</b>	<b>19</b>
<b>II Support vector machine</b>		<b>20</b>
<b>1</b>	<b>SVM - cas séparable</b>	<b>21</b>
<b>2</b>	<b>SVM : cas non séparable</b>	<b>25</b>
<b>3</b>	<b>SVM non linéaire : astuce du noyau</b>	<b>31</b>
<b>4</b>	<b>Scores et probabilités</b>	<b>34</b>
<b>5</b>	<b>Compléments : SVM multi-classes et SVR</b>	<b>36</b>
5.1	SVM multiclassess . . . . .	36
5.2	Support vector regression (SVR) . . . . .	38
<b>6</b>	<b>Bibliographie</b>	<b>42</b>
<b>III Agrégation</b>		<b>43</b>
<b>1</b>	<b>Bagging et forêts aléatoires</b>	<b>43</b>
1.1	Bagging . . . . .	43
1.2	Forêts aléatoires . . . . .	46
<b>2</b>	<b>Boosting</b>	<b>51</b>
2.1	Algorithmes de gradient boosting . . . . .	51
2.2	Choix des paramètres . . . . .	53
2.3	Compléments . . . . .	59
<b>3</b>	<b>Bibliographie</b>	<b>60</b>

## Première partie

# Apprentissage : contexte et formalisation

## 1 Motivations

### 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, 2018]

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...

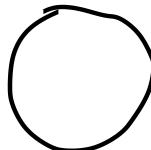
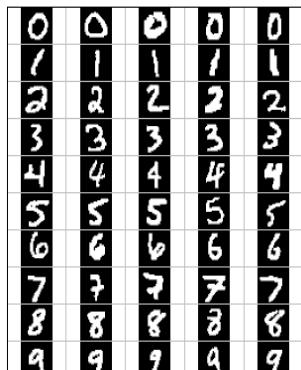
#### Conclusion

Capacités informatiques  $\Rightarrow$  Data Mining  $\Rightarrow$  Apprentissage statistique  $\Rightarrow$  Big Data  $\Rightarrow$  Intelligence artificielle...

#### Reconnaissance de l'écriture

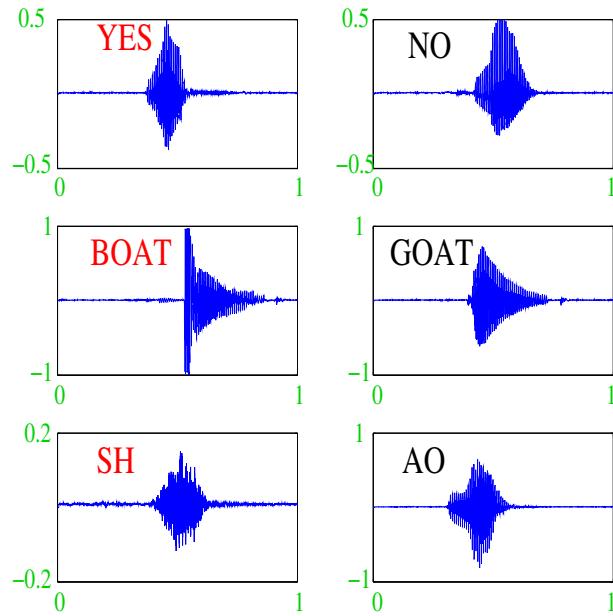
### Apprentissage statistique

Comprendre et apprendre un comportement à partir d'exemples.

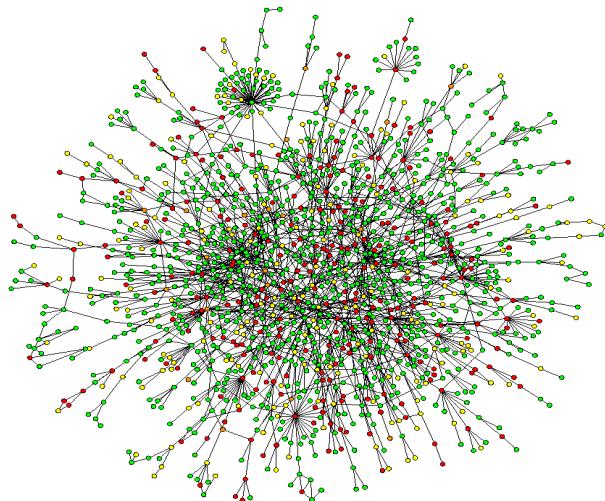


Qu'est-ce qui est écrit ? 0, 1, 2... ?

## 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
```

## 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
```

## Question

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

## Estimation vs apprentissage

- *Estimation* : objectifs **explicatifs**
  - notion de **modèle** ;
  - décisions prises à l'aide de **tests statistiques**.
- *Apprentissage* : objectifs **prédictifs**
  - **complexité des modèles** "peu" importantes ;
  - décisions prises à l'aide de **critères de prévisions**.

## 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...

## Théorie de l'apprentissage statistique

### 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/>

## Wikistat

- Page de cours et tutoriels très bien faits sur la *statistique classique et moderne*.
- On pourra notamment regarder les *vignettes* sur la partie **apprentissage** :
  - [Wikistat, 2020a]
  - [Wikistat, 2020b]
  - ...
- Plusieurs parties de ce cours sont *inspirées de ces vignettes*.

## 2 Cadre mathématique pour l'apprentissage supervisé

### 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

### Remarque

La nature des variables associées aux *entrées*  $x_i$  est *variée* (quanti, quali, 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 sorties  $y_i \in \mathcal{Y}$  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

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

## 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  $\mathbf{P}$ .

## 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  $f$ .

## Fonction de prévision optimale

### 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  $\ell$ .

### 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  $f_n = f_n(\cdot, \mathcal{D}_n)$  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*  $\ell$ ).
- 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.

## 3 Exemples de fonction de perte

### Régression

- 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

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

### Classification binaire

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

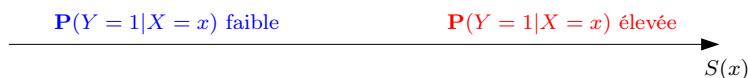
$$\begin{aligned}\ell : \{-1, 1\} \times \{-1, 1\} &\rightarrow \mathbb{R}^+ \\ (y, y') &\mapsto \mathbf{1}_{y \neq y'}\end{aligned}$$

- 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



- 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 ;

	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		$AUC(S)$	$\mathbf{P}(Y = 1 X = x)$

## Courbe ROC et AUC

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

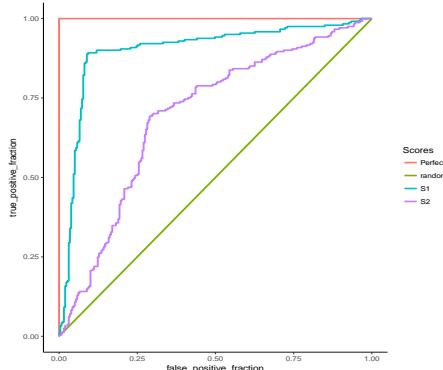
$$\begin{cases} x(s) = \alpha(s) = 1 - sp(s) = \mathbf{P}(S(X) > s|Y = -1) \\ y(s) = 1 - \beta(s) = se(s) = \mathbf{P}(S(X) \geq s|Y = 1) \end{cases}$$

- 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

## 4 Estimation du risque

### 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

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

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*  $\mathcal{R}(f_n) = \mathbf{E}[\ell(Y, f_n(X))]$ .
- **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  $f_n$ .

### 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  $\widehat{\mathcal{R}}_n(f_n) = \frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \ell(Y_i, f_{n,app}(X_i))$ .

### 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 *differentes 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  $k = 1, \dots, K$ 
  - (a)  $\mathcal{I}_{app} = \{1, \dots, n\} \setminus \mathcal{I}_k$  et  $\mathcal{I}_{test} = \mathcal{I}_k$  ;
  - (b) Construire l'algorithme de prédiction sur  $\mathcal{D}_{n,app} = \{(X_i, Y_i) : i \in \mathcal{I}_{app}\}$ , on le note  $f_{n,k}$  ;
  - (c) En déduire  $f_n(X_i) = f_{n,k}(X_i)$  pour  $i \in \mathcal{I}_{test}$  ;
3. **Retourner**

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

## Commentaires

- 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

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

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

## Autres approches

- *Estimation par pénalisation* : critère **ajustement/complexité**,  $C_p$  de Mallows, AIC-BIC...
- *Validation croisée Monte-Carlo* : répéter plusieurs fois la validation hold out ;
- *Bootstrap* : notamment **Out Of Bag** ;
- voir [Wikistat, 2020b].

## 5 Le sur-apprentissage

- La plupart des modèles statistiques renvoient des estimateurs qui dépendent de *paramètres  $\lambda$  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**.

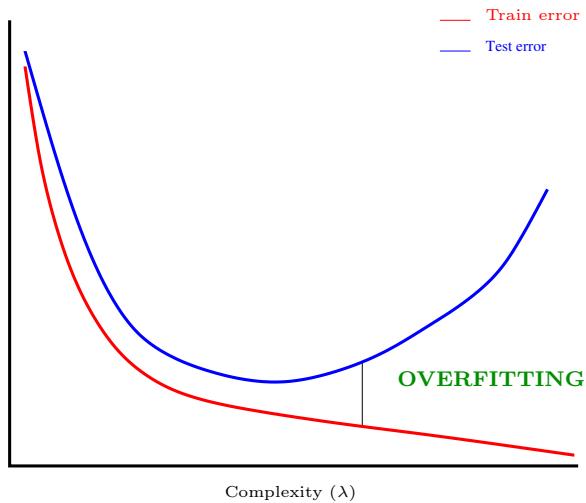
- Le paramètre  $\lambda$  à sélectionner représente le plus souvent la *complexité du modèle* :

### **Complexité $\Rightarrow$ compromis biais/variance**

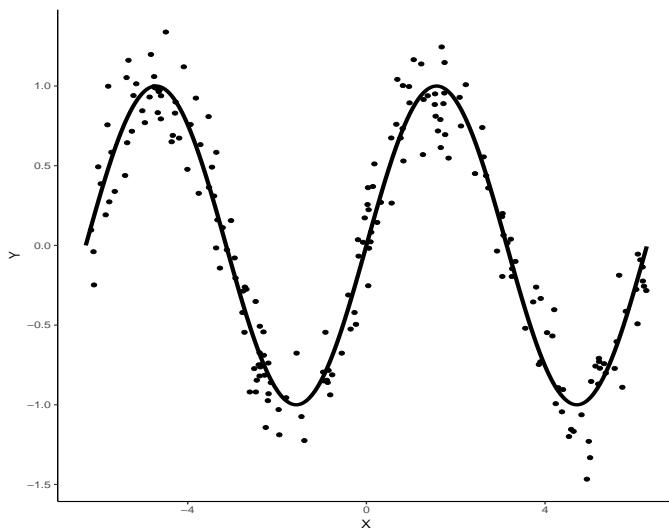
- $\lambda$  petit  $\Rightarrow$  modèle peu flexible  $\Rightarrow$  mauvaise adéquation sur les données  $\Rightarrow$  biais  $\nearrow$ , variance  $\searrow$ .
- $\lambda$  grand  $\Rightarrow$  modèle trop flexible  $\Rightarrow$  **sur-ajustement**  $\Rightarrow$  biais  $\searrow$ , variance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

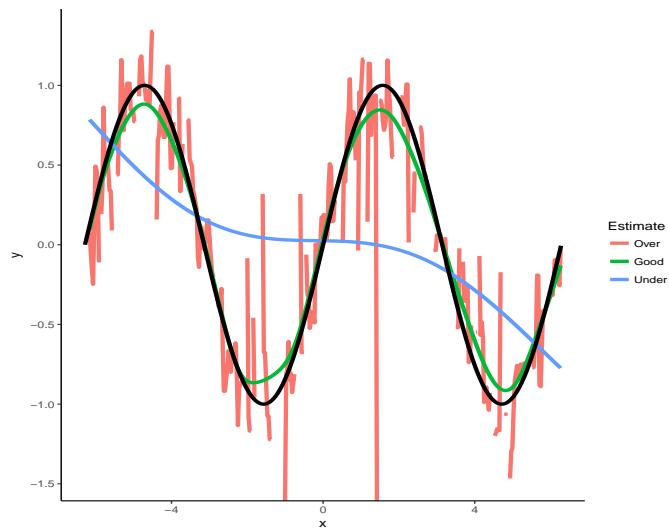
### **Overfitting**

*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.

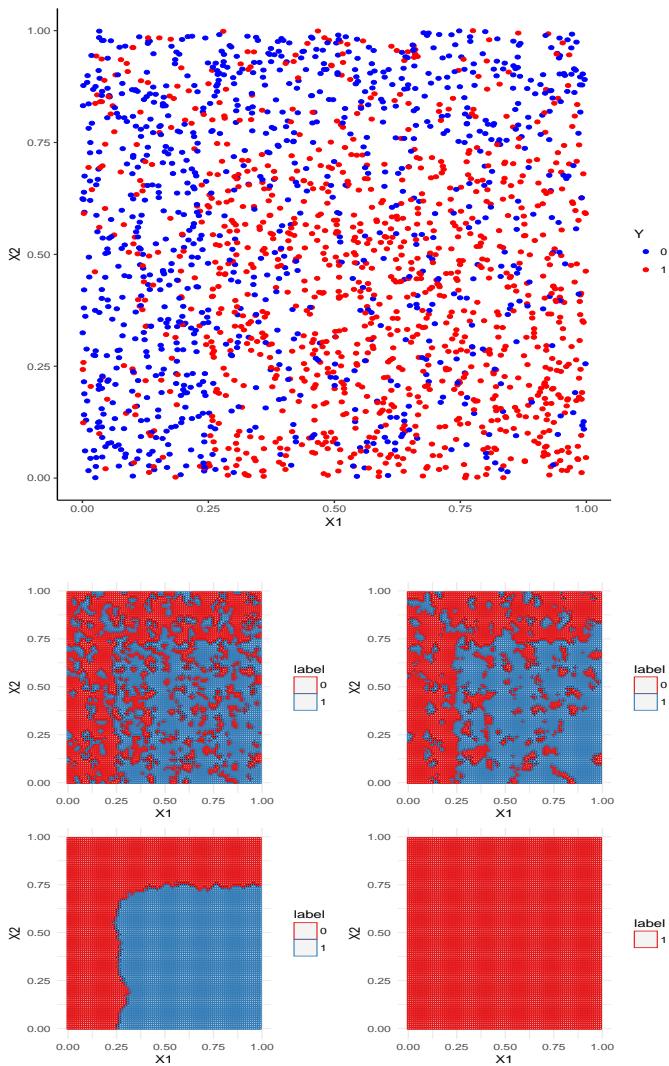


### **Overfitting en régression**





## Overfitting en classification supervisée



## Application shiny

[https://lrouviere.shinyapps.io/overfitting\\_app/](https://lrouviere.shinyapps.io/overfitting_app/)

## 6 Le package caret

### 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...)

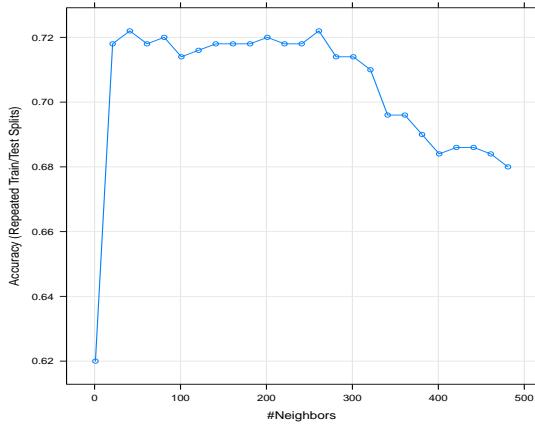
### Apprentissage-validation

```
> 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, ...
## 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

```

> ctrl13 <- trainControl(method="repeatedcv",repeats=5,number=10)
> e3 <- train(Y~.,data=dapp,method="knn",trControl=ctrl13,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, ...
## 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")
> ctrl111 <- trainControl(method="LGOCV", number=1, index=list(1:1500),
+                           classProbs=TRUE, summary=twoClassSummary)
> e4 <- train(Class~., data=donnees1, method="knn", trControl=ctrl111,
+               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

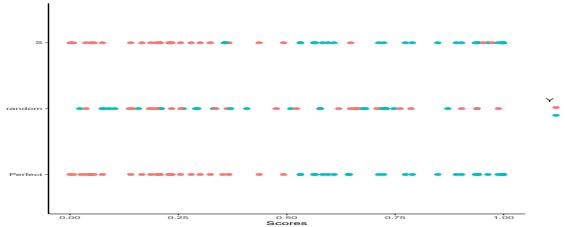
## 441 0.7258933 0.6192469 0.7471264
## 461 0.7220619 0.6150628 0.7471264
## 481 0.7236330 0.6108787 0.7432950
##
## ROC was used to select the optimal model using the largest value.
## The final value used for the model was k = 121.

```

⇒ Partie 1 du tuto

## 7 Annexe : compléments sur les scores

### Scores parfait et aléatoire



**Définition 7.1.** — *Score parfait* : il est tel qu'il existe un seuil  $s^*$  tel que

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

— *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

— 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 :  $se(s) = \mathbf{P}(S(X) \geq s | Y = 1) = 1 - \beta(s)$

### 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

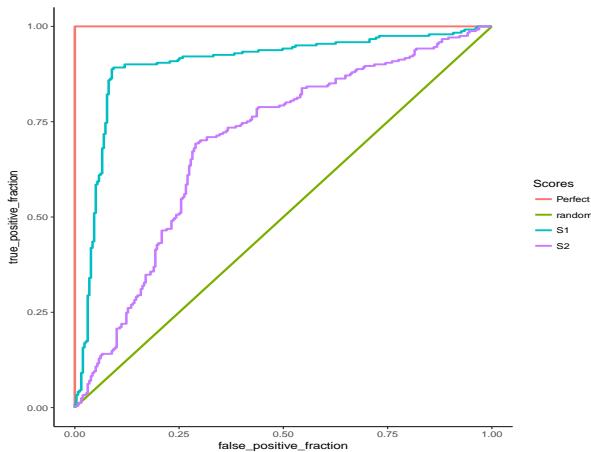
- Idée : représenter sur un graphe 2d les deux types d'erreur pour tous les seuils  $s$ .

**Définition 7.2.** 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.

### AUC

**Définition 7.3.** — 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 7.1.** — Etant données deux observations  $(X_1, Y_1)$  et  $(X_2, Y_2)$  indépendantes et de même loi que  $(X, Y)$ , on a

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

### AUC

```

> 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.

**Théorème 7.1** ([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

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

## 8 Bibliographie

### Références

#### Biblio1

- [Besse, 2018] Besse, P. (2018). *Science des données - Apprentissage Statistique*. INSA - Toulouse. [http://www.math.univ-toulouse.fr/~besse/pub/Appren\\_stat.pdf](http://www.math.univ-toulouse.fr/~besse/pub/Appren_stat.pdf).
- [Bousquet et al., 2003] Bousquet, O., Boucheron, S., and Lugosi, G. (2003). *Introduction to Statistical Learning Theory*, chapter Advanced Lectures on Machine Learning. Springer.
- [Cléménçon et al., 2008] Cléménçon, S., Lugosi, G., and Vayatis, N. (2008). Ranking and empirical minimization of u-statistics. *The Annals of Statistics*, 36(2) :844–874.
- [Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer, second edition.
- [James et al., 2015] James, G., Witten, D., Hastie, T., and Tibshirani, R. (2015). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer.
- [Vapnik, 2000] Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. Springer, second edition.
- [Wikistat, 2020a] Wikistat (2020a). Apprentissage machine — introduction. <http://wikistat.fr/pdf/st-m-Intro-ApprentStat.pdf>.
- [Wikistat, 2020b] Wikistat (2020b). Qualité de prévision et risque. <http://wikistat.fr/pdf/st-m-app-risque.pdf>.

## Deuxième partie

# Support vector machine

### Cadre et notation

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

### Objectif

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

### 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* :

$$\text{logit} \frac{p(x)}{1-p(x)} = \beta_0 + \beta_1x_1 + \dots + \beta_px_p$$

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

$$g(x) = \begin{cases} 1 & \text{si } \beta_0 + \beta_1x_1 + \dots + \beta_px_p \geq 0 \\ -1 & \text{sinon.} \end{cases}$$

### Exemple 2 : LDA

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

- *Règle de classification* :

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

- 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 $p = 2$

- 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.



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

## 1 SVM - cas séparable

### 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 en trouver dans la [partie 2.4 du tutoriel](#).

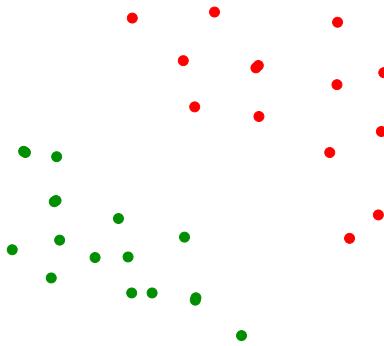
### Présentation

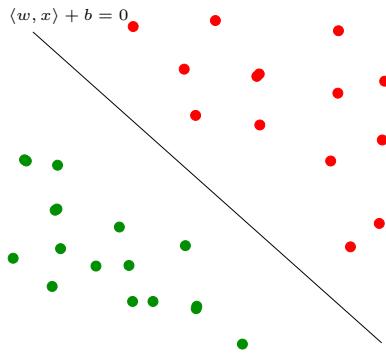
- L'approche SVM [Vapnik, 2000] peut être vue comme une *généralisation* de "recherche d'hyperplan optimal".

### 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  $i$  :

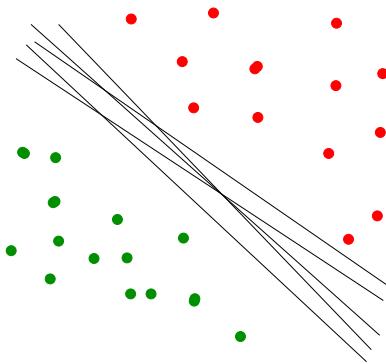
- $y_i = 1$  si  $\langle w, x_i \rangle + b = w^t x_i + b > 0$  ;
- $y_i = -1$  si  $\langle w, x_i \rangle + b = w^t x_i + b < 0$ .





### Vocabulaire

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

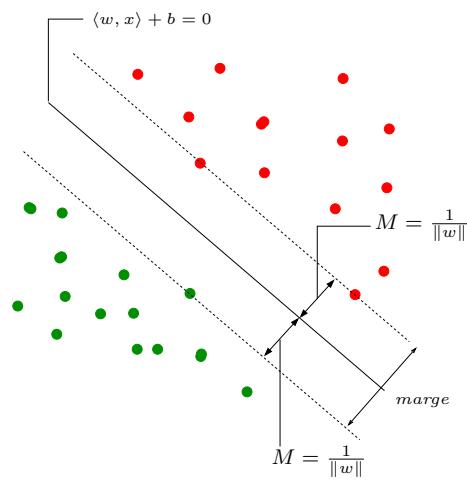


### Problème

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.

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

## Solutions

— On obtient

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

où les  $\alpha_i^*$  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  $\alpha_i^*$  non nul.

### Remarque

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

## Vecteurs supports

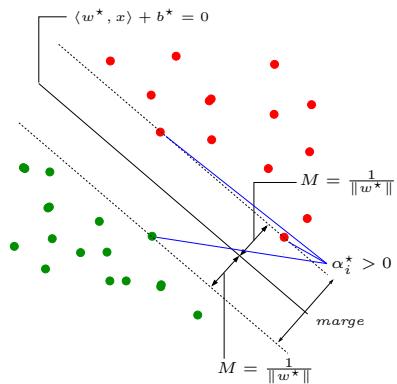
### Propriété (conditions KKT)

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

### Conséquence (importante)

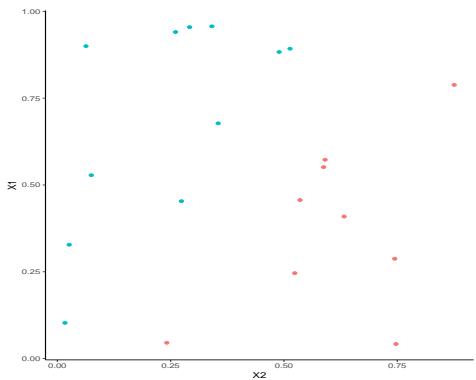
- Si  $\alpha_i^* \neq 0$  alors  $y_i(x_i^t w^* + b) = 1$  et  $x_i$  est **sur la marge**.
- $w^*$  se calcule **uniquement** à partir de ces points là.
- Ces points sont appelés les **vecteurs supports** de la SVM.

## Représentation



## Le coin R

- 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
```

- $mod.svm\$coefs = \alpha^* u_i$  pour chaque vecteur support

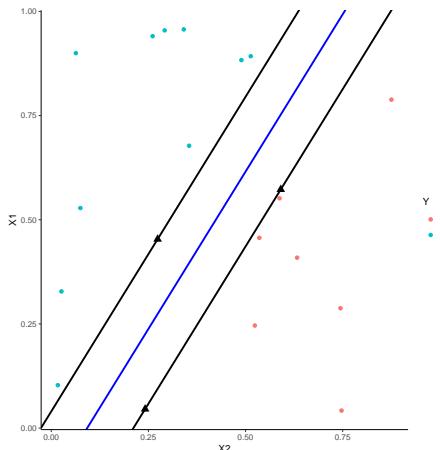
```
> mod.svm$coefs
##          [,1]
## [1,] 1.898982
## [2,] 1.905497
## [3,] -3.804479
```

- 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
```

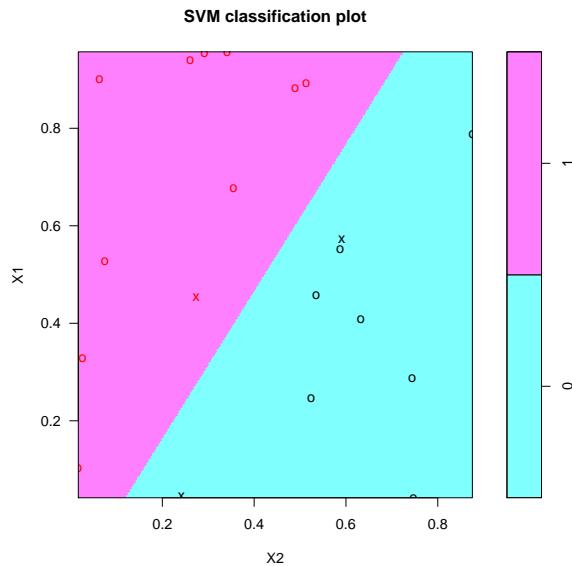
On peut ainsi visualiser

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



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

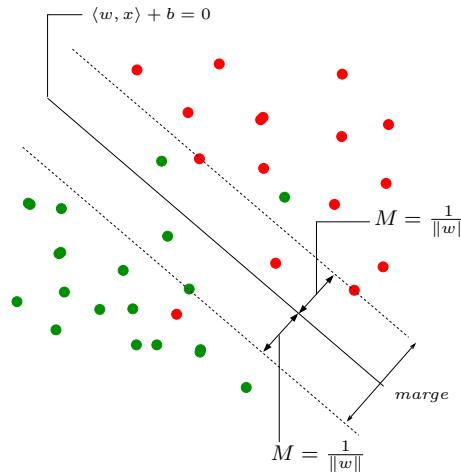
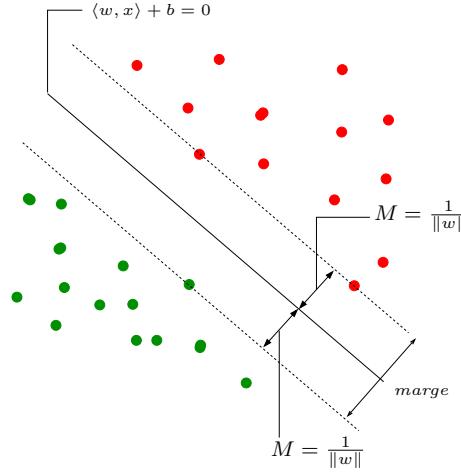
```
> plot(mod.svm,data=df,fill=TRUE,grid=500)
```



## 2 SVM : cas non séparable

### *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*.

## 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  $\xi_1, \dots, \xi_n$  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  $\xi_i$  *nulles*;
- Lorsque  $\xi_i > 0$ , on souhaite que  $\xi_i$  soit le *plus petit possible*.

### Cas non séparable : problème d'optimisation (primal)

- Il s'agit de minimiser en  $(w, b, \xi)$

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

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

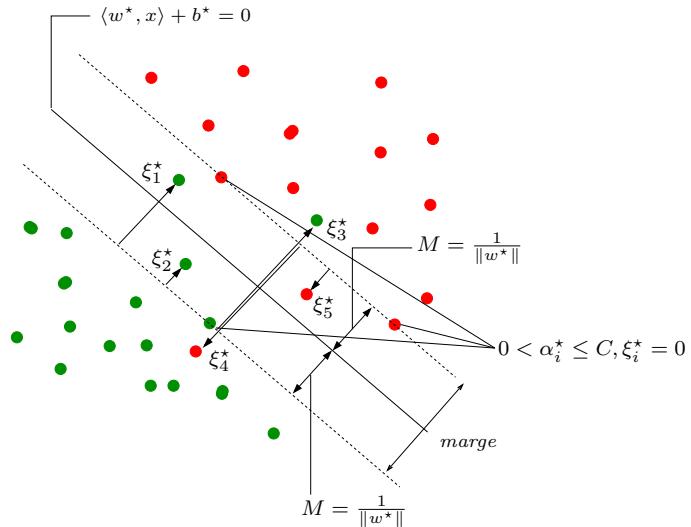
- $C > 0$  est un paramètre à calibrer (**paramètre de coût**).
- Le **cas séparable** correspond à  $C \rightarrow \infty$ .
- Les *solutions* de ce nouveau problème d'optimisation s'obtiennent de la *même façon* que dans le cas séparable (Lagrangien, problème dual...).
- L'*hyperplan optimal* est défini 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  $0 < \alpha_i^* < C$ .

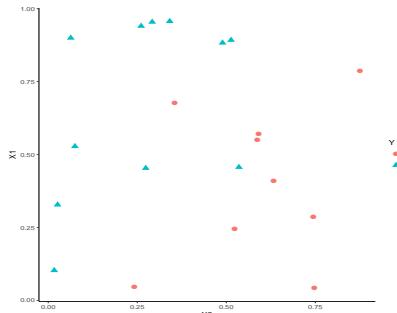
### Vecteurs supports

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

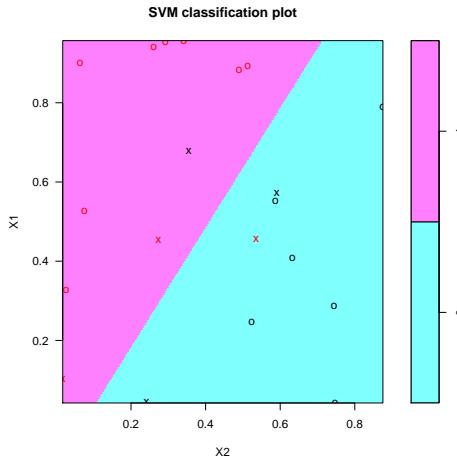
- On utilise la même fonction que dans le *cas séparable* (**svm** du package **e1071**) ;
- L'argument **cost** correspond à la *constante de régularisation*  $C$ .



```
> 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)
```



## Choix de $C$

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

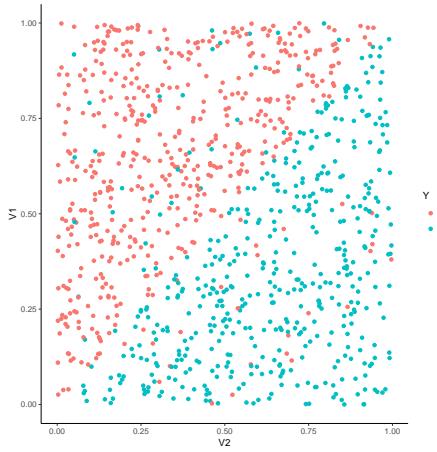
- $C \searrow$  : la marge est privilégiée et les  $\xi_i \nearrow \Rightarrow$  beaucoup d'observations dans la marge ou **mal classées** (et donc beaucoup de vecteurs supports).
- $C \nearrow \Rightarrow \xi_i \searrow$  donc moins d'observations mal classées  $\Rightarrow$  **meilleur ajustement**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

## Conclusion

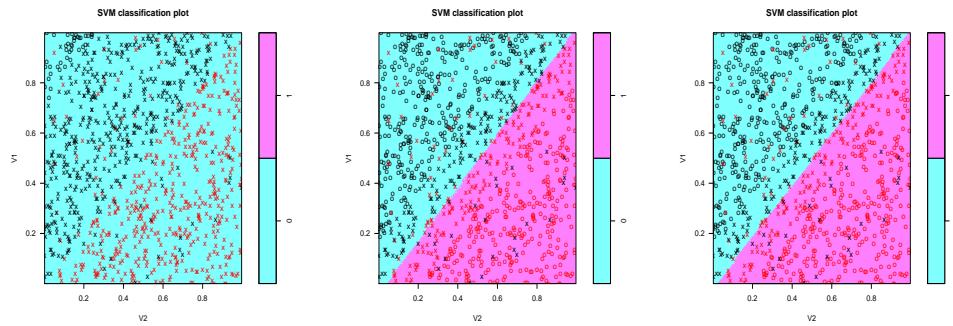
Il est donc très important de bien choisir ce paramètre.

- 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*.
- 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



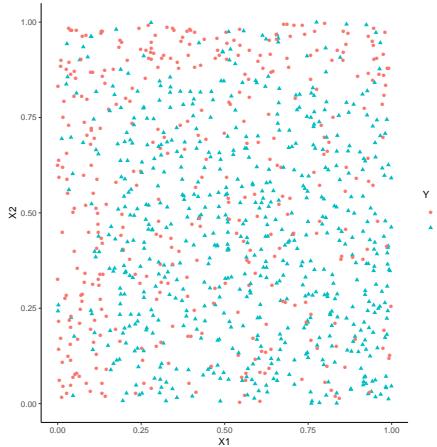
```
> mod.svm1 <- svm(Y~., data=df3, kernel="linear", cost=0.000001)
> mod.svm2 <- svm(Y~., data=df3, kernel="linear", cost=0.1)
> mod.svm3 <- svm(Y~., data=df3, kernel="linear", cost=5)
```



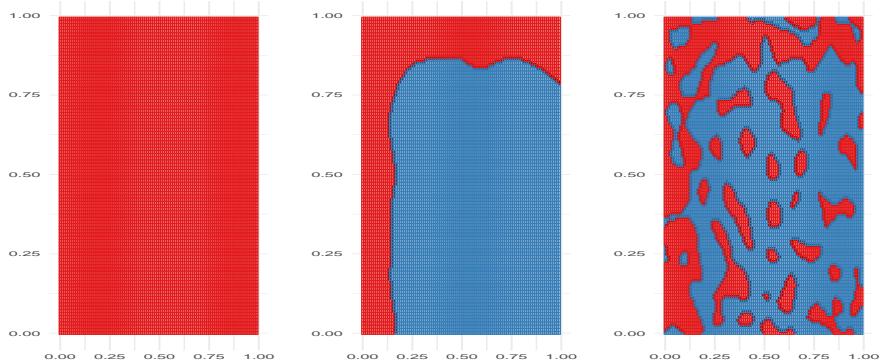
```
> mod.svm1$nSV
## [1] 480 480
> mod.svm2$nSV
## [1] 190 190
> mod.svm3$nSV
## [1] 166 165
```

## 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)
```

```

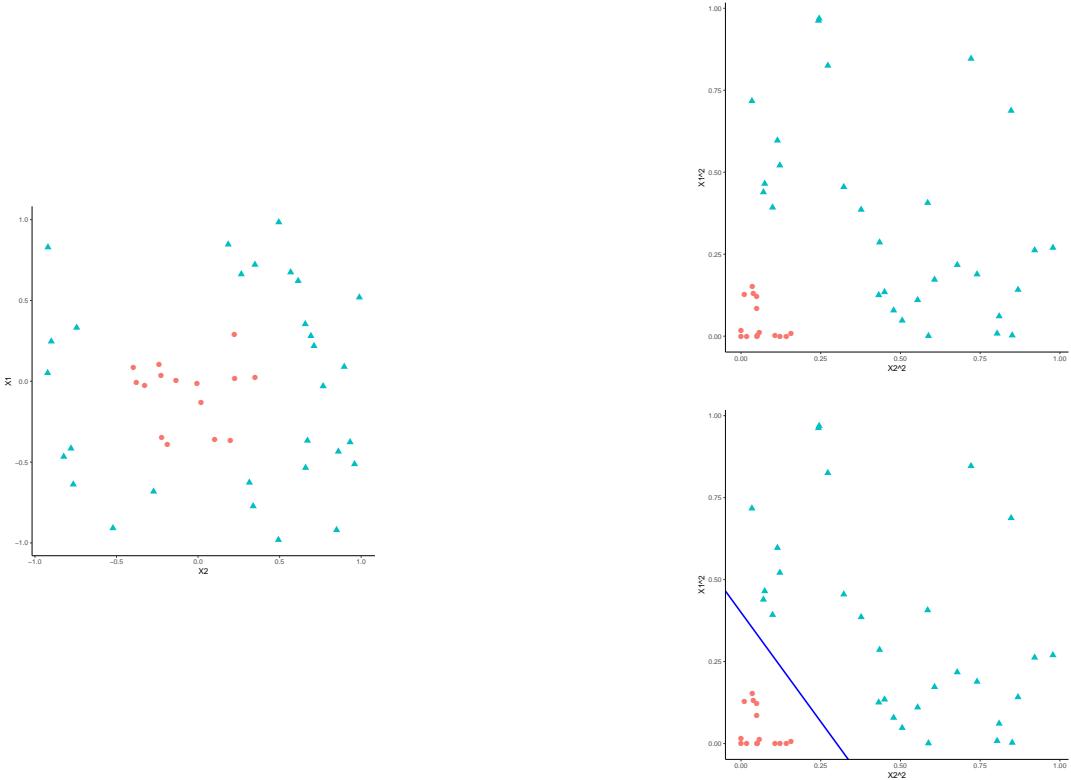
## Support Vector Machines with Linear Kernel
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 900, 900, 900, 900, 900, ...
## Resampling results across tuning parameters:
##   C      Accuracy   Kappa
##   1e-03  0.8700    0.7377051
##   1e-02  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.

```

⇒ Parties 2.1 et 2.2 du tuto

### 3 SVM non linéaire : astuce du noyau

- Les *solutions linéaires* ne sont pas toujours intéressantes.



#### *Idée*

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

#### Noyau

**Définition 3.1.** 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  $\Phi$  est le produit scalaire entre  $\Phi(x)$  et  $\Phi(x')$  :

$$K : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$$

$$(x, x') \mapsto \langle \Phi(x), \Phi(x') \rangle_{\mathcal{H}}.$$

### 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*...
- 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  $\Phi$  ! 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  $\Phi$ , il suffit de se donner un noyau  $K$  !

### SVM dans l'espace original

- Le *problème dual* consiste à maximiser

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

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

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

### SVM dans le feature space

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

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

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

### SVM dans le feature space avec un noyau

- 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 connaitre*  $\mathcal{H}$  ou  $\Phi$ , il suffit de connaitre  $K$  !

## Questions

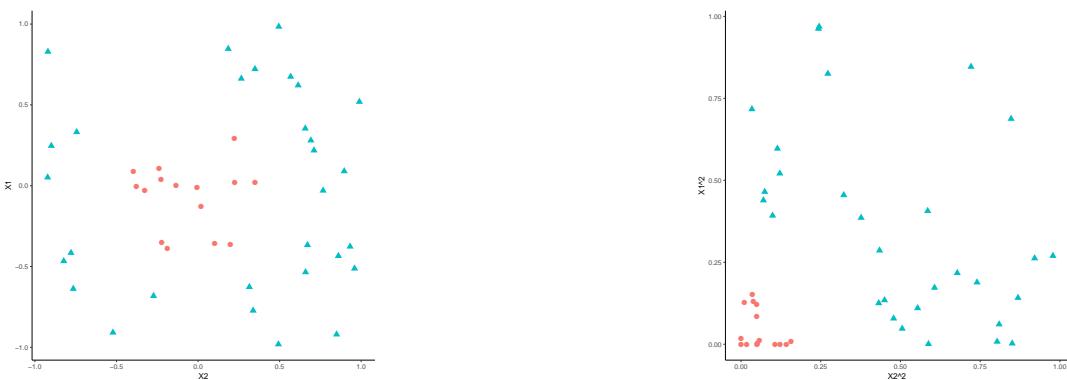
Qu'est-ce qu'un **noyau**? Comment construire un noyau?

**Théorème 3.1** ([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.  $K(x, x') = K(x', x) \forall (x, x') \in \mathcal{X}^2$  ;
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  $\mathbb{R}^d$ )

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

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

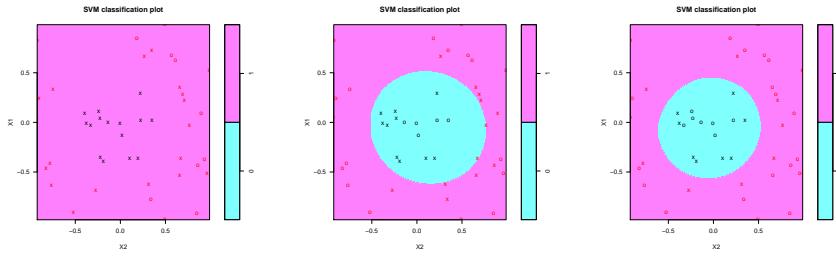
## Remarque

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

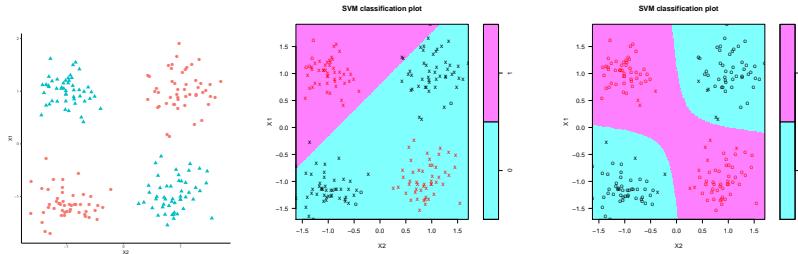
— Argument *kernel* dans la fonction **svm**.

```
> 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

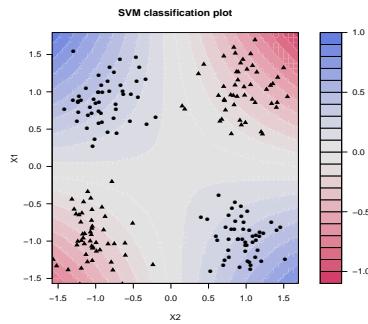
```
> svm(Y~, data=donnees, kernel="linear", cost=1)
> svm(Y~, data=donnees, kernel="polynomial", degree=2, cost=1)
```



### 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)
```



⇒ Partie 2.3 du tuto

## 4 Scores et probabilités

- Jusqu'à présent nous avons utiliser la SVM uniquement pour *classer* :
  - 1 si on est d'un **côté de l'hyperplan**  $\Rightarrow \sum_{i=1}^n \alpha_i^* y_i K(x_i, x) + b^* \geq 0$ ;
  - 1 si on est de l'**autre côté**  $\Rightarrow \sum_{i=1}^n \alpha_i^* y_i K(x_i, x) + b^* < 0$ .

- *Rappel* : dans le cas linéaire la fonction

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

mesure la **distance entre  $x$  et l'hyperplan séparateur**.

- *Conclusion* : cette fonction peut être utilisée comme un **score**, puisque sa valeur (absolue) traduit une **confiance** que l'on a dans la prévision.

## Probabilités

- La valeur de  $f(x)$  est *difficilement interprétable* en tant que telle.
- Il peut être intéressant de la "ramener" entre 0 et 1 pour l'interpréter comme une **estimation de  $\mathbf{P}(Y = 1|X = x)$** .

## Une solution

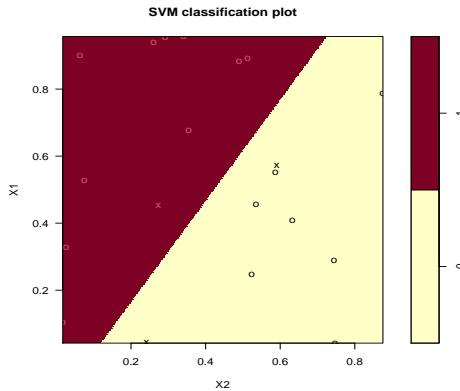
- Considérer un **modèle logistique** :

$$\mathbf{P}(Y = 1|X = x) = \frac{1}{1 + \exp(a f(x) + b)}$$

- et d'estimer  $a$  et  $b$  par **maximum de vraisemblance** sur les données  $(f(x_i), y_i), i = 1, \dots, n$ .

## Le coin R

```
> mod.svm <- svm(Y~., data=df, kernel="linear", cost=10000000000, probability=TRUE)
> plot(mod.svm, data=df, fill=TRUE, grid=500)
```



- Nouvelle observation :

```
> newX <- data.frame(X1=0.2, X2=0.6)
```

- Calcul du *score* et de la *proba* :

```
> predict(mod.svm, newdata=newX, decision.values = TRUE, probability=TRUE)
## 1
## 0
## attr(,"decision.values")
##          0/1
## 1 3.153157
## attr(,"probabilities")
##          0          1
## 1 0.9634021 0.0365979
## Levels: 0 1
```

- On peut retrouver cette proba avec :

```
> a <- mod.svm$probA
> b <- mod.svm$probB
> 1/(1+exp(a*3.153157+b))
## [1] 0.9634021
```

## 5 Compléments : SVM multi-classes et SVR

### Cible multi-classes ou quantitative

- On a abordé ici uniquement le problème de la *classification binaire* :  $y_i \in \{-1, 1\}$ .
- Les SVM se généralisent aux cas *multi-classes* :  $y_i \in \{1, \dots, M\}$
- et à la *régression* :  $y_i \in \mathbb{R}$ .

### 5.1 SVM multiclasses

- On suppose ici que  $y_i \in \{1, \dots, M\}$
- Il existe plusieurs approches pour généraliser les SVM à ce contexte, notamment :
- **One against one**

#### *Idée*

Faire une SVM binaire sur **toutes les paires**  $(j, k) \in \{1, \dots, M\}^2$  avec  $j \neq k$  et choisir le **groupe qui gagne le plus souvent**.

- **One against all**

#### *Idée*

Faire une SVM binaire de **chaque groupe contre les autres** et choisir le **groupe qui a la "plus belle victoire"**.

#### One against one

##### *Algorithme*

1. Pour chaque paire  $(j, k)$ , faire la SVM **binaire** avec uniquement les individus des groupes  $k$  et  $j$  ;
2. On obtient ainsi  $M(M - 1)/2$  **règles "linéaires"**  $f_{j,k}(x)$ .
3. On calcule pour  $j = 1, \dots, M$

$$V(j) = \sum_{k \neq j} \text{signe}(f_{j,k}(x))$$

qui représente le **nombre de fois où on a voté  $j$  contre les autres groupes**.

4. On classe un nouvel individu  $x$  dans le groupe qui a **remporté le plus de suffrage** :

$$f(x) = \underset{j}{\operatorname{argmax}} V(j).$$

#### One against all

##### *Algorithme*

1. Faire une SVM binaire avec **tous les individus** de chaque groupe contre les autres.
2. On obtient ainsi  $M$  **règles "linéaires"**  $f_j(x)$  (groupe  $j$  contre les autres).
3. On classe un nouvel individu dans la classe qui a le score le plus élevé :

$$f(x) = \underset{j}{\operatorname{argmax}} f_j(x).$$

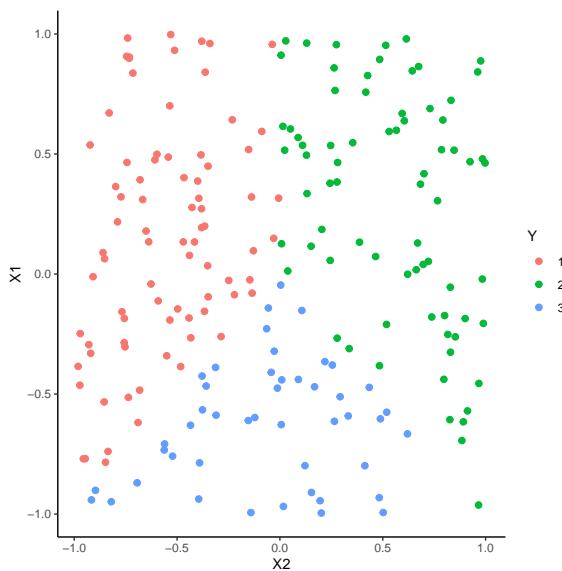
## Comparaison

- $M$  SVM binaire avec l'approche **one against all** contre  $M(M - 1)/2$  avec le **one against one** mais...
- moins d'individus dans les **one against one**.
- Risque de déséquilibre plus fort avec le **one against all** (mais généralement plus rapide).
- Comme dans le cas binaire, il faut *sélectionner* le paramètre de complexité, le noyau, les paramètres du noyau...

## Le coin R

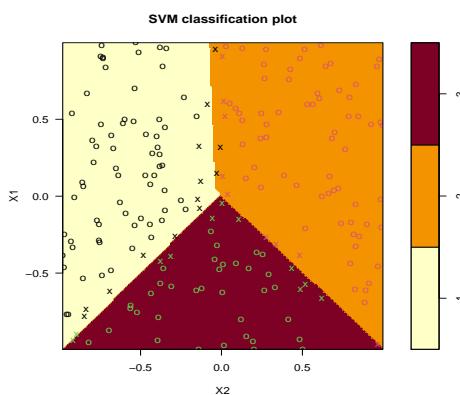
- L'approche **one against one** est plus souvent utilisée.
- C'est le cas par défaut avec **svm** de *e1071* et **ksvm** de *kernlab*.

## Exemple



## SVM linéaire multi classes

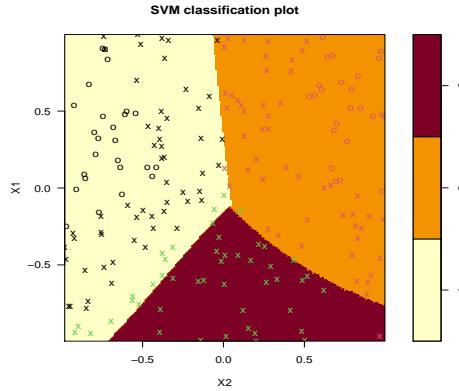
```
> multi <- svm(Y~, data=df, cost=10, kernel="linear")
> plot(multi, data=D, grid=200)
```



## SVM non linéaire multi classes

- Il "suffit" d'utiliser un *noyau*.

```
> multi2 <- svm(Y~, data=df, cost=0.1, kernel="sigmoid")
> plot(multi, data=D, grid=200)
```



## 5.2 Support vector regression (SVR)

- On suppose ici que les \$y\_i\$ sont dans \$\mathbb{R}\$.
- On ne va plus chercher l'hyperplan qui sépare au mieux les groupes mais
- l'hyperplan \$(w, b)\$ qui "*approche au mieux*" les valeurs \$y\_i\$

$$|\langle w, x_i \rangle + b - y_i| \text{ petits.}$$

### Comparaison avec les MCO

- *Approche MCO* (rappel) : on cherche \$(w, b)\$ qui minimise

$$\sum_{i=1}^n (y_i - \langle w, x_i \rangle - b)^2$$

- *Approche SVR* : on veut
  1. tous les points à distance de moins de \$\varepsilon\$ de \$(w, b)\$ ;
  2. \$(w, b)\$ de marge maximale (\$\|w\|\$ minimale).

### Optimisation SVR

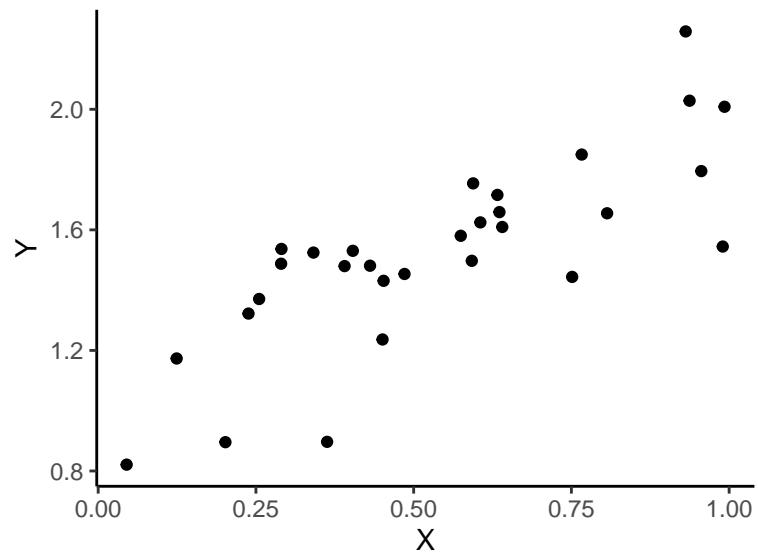
On va chercher à **minimiser la norme de \$w\$** en se fixant comme contrainte que les \$y\_i\$ ne soient pas "trop loin" de l'hyperplan :

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

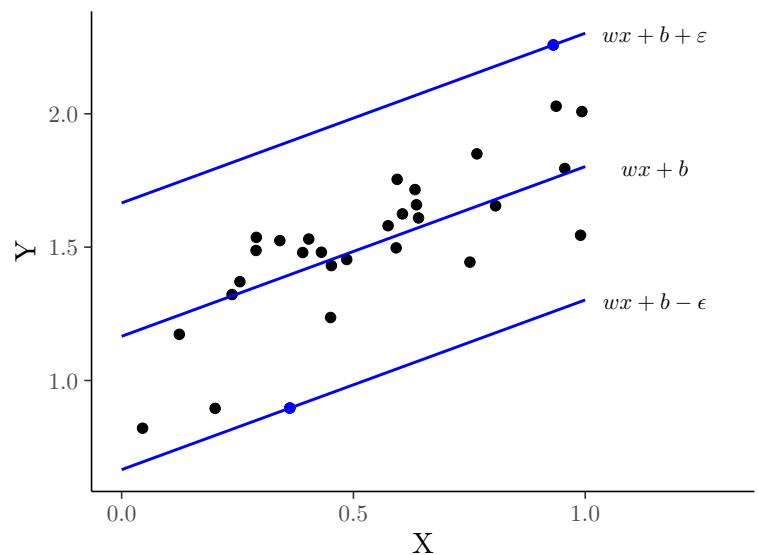
sous les contraintes \$|y\_i - \langle w, x\_i \rangle - b| \leq \varepsilon\$, \$i = 1, \dots, n\$,

où \$\varepsilon > 0\$ est un paramètre à calibrer par l'utilisateur.

### Un exemple en dimension 1

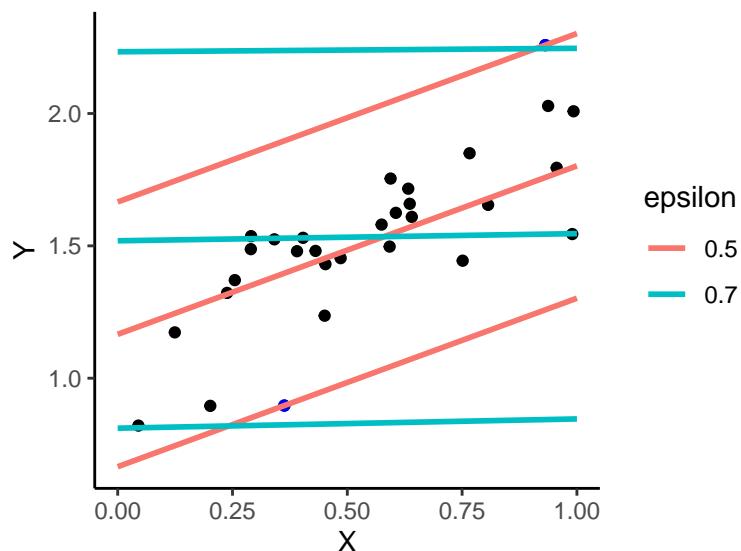


### Un exemple en dimension 1



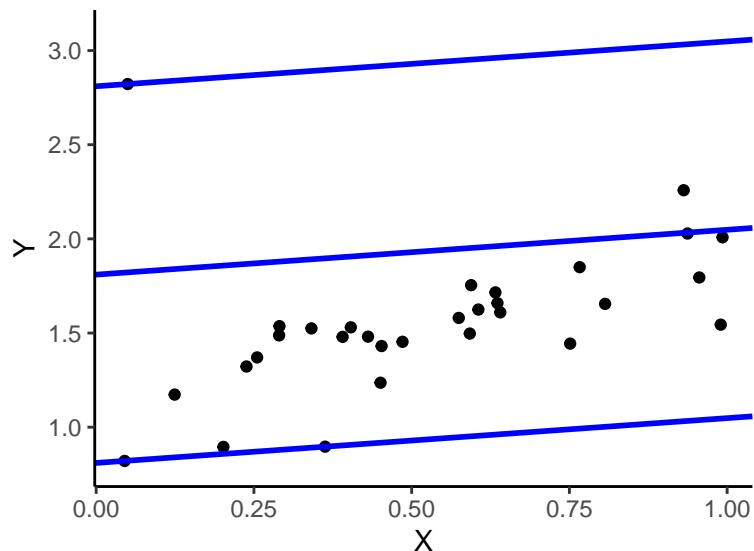
### Influence de $\varepsilon$

- Il contrôle le *niveau de tolérance* que l'on se donne.



### Alléger la contrainte...

- La contrainte nécessite souvent de prendre des grandes valeurs de  $\varepsilon$ ...



- Clairement pas satisfaisant de prendre  $\varepsilon$  trop grand.

### *Idée*

- Comme pour la **SVM binaire**, autoriser des observations à se situer en dehors de la marge !
- *Comment ?* En introduisant des **slack variables** !

### SVR cas général

#### *Le problème d'optimisation*

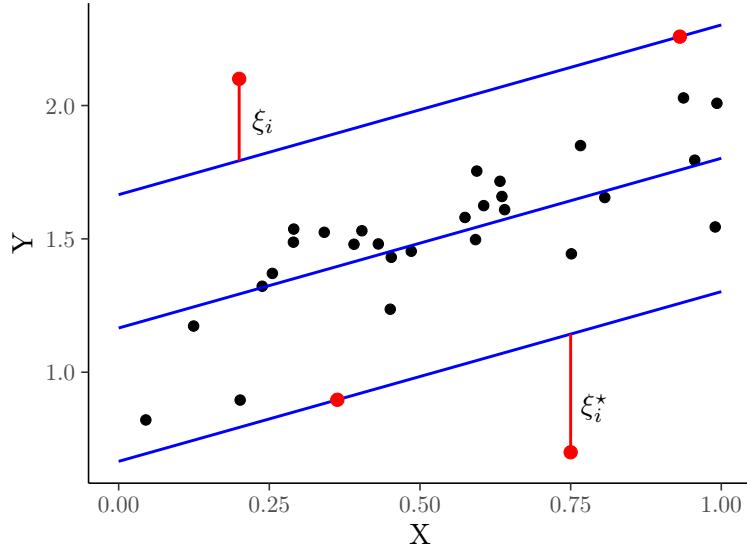
On cherche  $(w, b, \xi, \xi^*)$  qui minimise

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

sous les contraintes

$$\begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i, & i = 1, \dots, n, \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^*, & i = 1, \dots, n, \\ \xi_i \geq 0, \xi_i^* \geq 0, & i = 1, \dots, n \end{cases}$$

### Slack variables en régression



### Rien ne change après...

- Les solutions s'obtiennent en résolvant le *problème dual*  $\Rightarrow \alpha_i, \alpha_i^*$ .
- Les données (les  $X$ ) sont généralement *centrées-réduites* pour éviter les problèmes d'échelle.
- Les observations vérifiant  $\alpha_i^* - \alpha_i \neq 0$  sont les **vecteurs supports**.
- L'hyperplan optimal se déduit des **vecteurs supports** :

$$w^* = \sum_{i=1}^n (\alpha_i^* - \alpha_i) x_i.$$

- L'*astuce du noyau* reste d'actualité pour prendre en compte de la **non linéarité**.
- Il faut *sélectionner C*, le noyau (et ses paramètres) ainsi que  $\varepsilon$ ...

### Le coin R

- Là aussi, pas grand chose ne change.

```
> svm(Y~., data=df, kernel="linear", epsilon=0.5, cost=100)
##
## Call:
## sum(formula = Y ~ ., data = df, kernel = "linear", epsilon = 0.5,
##      cost = 100)
##
## Parameters:
##   SVM-Type:  eps-regression
##   SVM-Kernel: linear
##   cost: 100
##   gamma: 1
##   epsilon: 0.5
##
## Number of Support Vectors:  14
```

## Conclusion

- Algorithme machine learning pouvant être utilisé en *régression* et en *classification supervisée*.
- Méthode *linéaire* mais prise en compte possible de la **non linéarité** grâce à l'**astuce du noyau**.
- *Calibration difficile* : beaucoup de paramètres
  1. paramètre de cout  $C$
  2. noyau
  3. paramètres du noyau
  4. seuil de tolérance  $\varepsilon$  pour la régression
- et souvent **peu d'information a priori** sur la valeur de ces paramètres...

## 6 Bibliographie

### Références

#### Biblio3

- [Aronszajn, 1950] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68 :337–404.
- [Fromont, 2015] Fromont, M. (2015). Apprentissage statistique. Université Rennes 2, diapos de cours.

# Troisième partie

# Aggrégation

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

1. construire un grand nombre de *classificateurs "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** ?

## 1 Bagging et forêts aléatoires

### Cadre

- 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 multiconnexion*.
- *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  $(X, Y)$ .

### 1.1 Bagging

- Le *bagging* désigne un ensemble de méthodes introduit par Léo Breiman [Breiman, 1996].
- *Bagging* : vient de la contraction de **Bootstrap Aggregating**.
- *Idée* : plutôt que de constituer 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 *agrégant*  $B$  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).

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

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	$m_B$

## 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 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 :

— Echantillons bootstrap :

— A la fin, on agrège :

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

## 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  $\theta_k, k = 1, \dots, B$ .
- 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  $\theta$ .
- 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.
- 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.

## 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

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

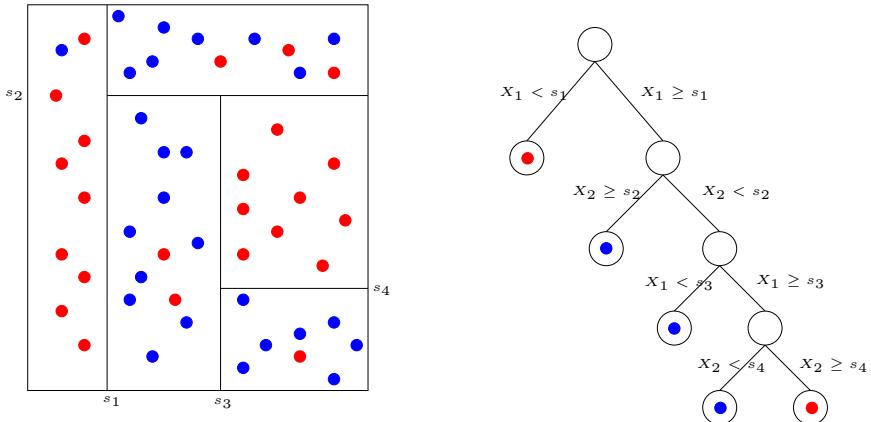
où  $\rho(x) = \text{corr}(m(x, \theta_k, \mathcal{D}_n), m(x, \theta_{k'}, \mathcal{D}_n))$  pour  $k \neq k'$ .

## Conclusion

- Bagger 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.

## 1.2 Forêts aléatoires

### Rappels sur les arbres



### Paramètre à calibrer

Profondeur

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

### Définition

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

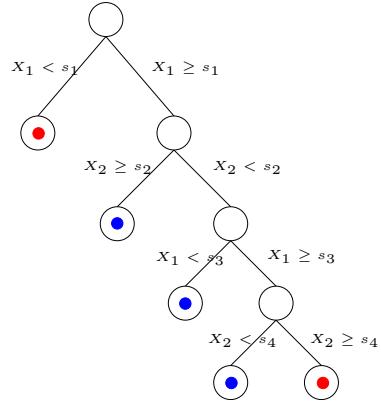
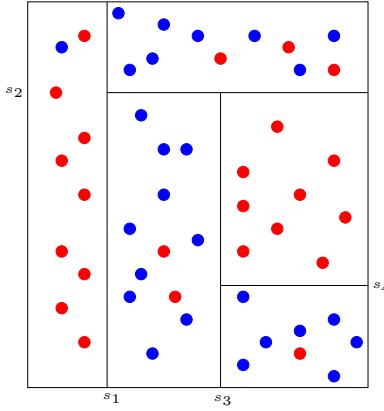
### Définition

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

- 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.
- **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 noeud
- $m \in \{1, \dots, d\}$  le nombre de variables candidates pour découper un noeud.

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  $T(\cdot, \theta_k, \mathcal{D}_n)$  l'arbre construit.

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

### Commentaires

- 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  $\theta_k$  : 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.

## 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.

## 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 classificateurs 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 algorithme bagging permet de définir une nouvelle méthode d'estimation de ces critères : méthode **OOB (Out Of Bag)**.

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	$m_6$

### Erreur Ouf 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 Our 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

- 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
```

### Remarque

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

## 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  $E_{OOB_k}$  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

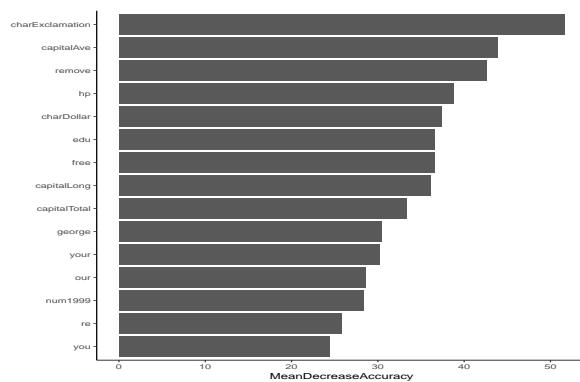
$$Imp(X_j) = \frac{1}{B} \sum_{k=1}^B (E_{OOB_k^j}^j - E_{OOB_k}).$$

### 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()
```



⇒ Partie 4.1 du tuto

## 2 Boosting

- 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*  $\Rightarrow$  *gradient boosting*.
- C'est cette *famille d'algorithmes* que nous présentons dans cette partie.

### 2.1 Algorithmes de gradient boosting

- $(X, Y)$  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 :

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

- *Remarque* :  $\mathcal{R}_n(g)$  nécessite uniquement  $g$  aux points  $X_i \Rightarrow \mathbf{g} = (g(X_1), \dots, g(X_n))$ .

*Minimisation de risque empirique*

Minimiser

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

sur une *classe de fonctions* donnée.

- *Exemple de classes* : l'ensemble des combinaisons d'arbres binaires

$$\mathcal{G}(M, J) = \left\{ \sum_{m=1}^M T(x, \Theta_m), \quad T(x, \Theta) = \sum_{j=1}^J \gamma_j \mathbf{1}_{x \in R_j} \right\}.$$

- Le problème précédent n'admet pas de *solution explicite*.
- Impossible de *minimiser sur l'ensemble des combinaisons d'arbres*...
- *Solution* : utiliser un *algorithme*.
- Par exemple construite la combinaison d'arbres de façon *récursive* à l'aide d'une *descente de gradient*

$$g_{m+1}(x) = g_m(x) + \lambda f_{m+1}(x).$$

#### Descente de gradient

- *Idée* : obtenir la combinaison d'arbres à l'aide d'une *descente de gradient* :

$$\mathbf{g}_{m+1} = \mathbf{g}_m + \lambda \mathbf{f}_{m+1}$$

où  $\mathbf{f}_{m+1}$  est l'opposé du gradient de  $\mathcal{R}_n(\mathbf{g})$  évalué en  $\mathbf{g}_m$ .

- La  $i$ ème composante de ce gradient est

$$-\left[ \frac{\partial \ell(y_i, g(x_i))}{\partial g(x_i)} \right]_{g(x_i)=g_m(x_i)}.$$

- **Exemple** : si  $\ell(y, g(x)) = 1/2(y - g(x))^2$  alors

$$-\left[ \frac{\partial \ell(y_i, g(x_i))}{\partial g(x_i)} \right]_{g(x_i)=g_m(x_i)} = y_i - g_m(x_i).$$

## Récapitulatif

- *Algorithme récursif* :
  - **itération  $m$**  :  $g_m$  (une combinaison d'arbres)
  - **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

$$\frac{1}{n} \sum_{i=1}^n (f_{m+1}(X_i) - h(X_i))^2.$$

- **Problème de régression**  $\Rightarrow$  si on désigne par

$$U_i = f_{m+1}(X_i) = -\nabla \mathcal{R}_n(g_m)(X_i) = -\frac{\partial}{\partial g(X_i)} \ell(Y_i, g(X_i)) \Big|_{g(X_i)=g_{m-1}(X_i)},$$

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

**Entrées :**

- $d_n = (x_1, y_1), \dots, (x_n, y_n)$  l'échantillon,  $\lambda$  un paramètre de régularisation tel que  $0 < \lambda \leq 1$ .
- $M$  le *nombre d'itérations*.
- paramètres de l'arbre (nombre de coupures...)

1. Initialisation :  $g_0(\cdot) = \operatorname{argmin}_c \frac{1}{n} \sum_{i=1}^n \ell(y_i, c)$

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 d'*algorithmes de prévision*  $(g_m(x))_m$ .

## Ajustement de l'arbre

- Un arbre = des régions  $R_{jm}$  et des constantes  $\gamma_{jm}$ .
- L'arbre est ajusté sur l'échantillon  $(x_i, u_i), i = 1, \dots, n$ .
- $\Rightarrow$  un arbre de régression pour obtenir les régions  $R_{jm}, j = 1, \dots, J_m$ .
- Les constantes s'obtiennent en utilisant la fonction de perte :

$$\gamma_{jm} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{jm}} \ell(y_i, g_{m-1}(x_i) + \gamma).$$

- On obtient ainsi

$$h_m(x) = \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}_{x \in R_{jm}}.$$

## Commentaires

- 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.  $\Rightarrow$  ces règles sont donc bien des combinaisons d'arbres.

- Plusieurs paramètres sont à choisir ou à calibrer par l'utilisateur :
  - fonction de perte  $\ell$  ;
  - nombre d'itérations  $M$  ;
  - paramètre de régularisation  $\lambda$  ;
  - paramètres de l'arbre (nombre de coupures notamment).

## 2.2 Choix des paramètres

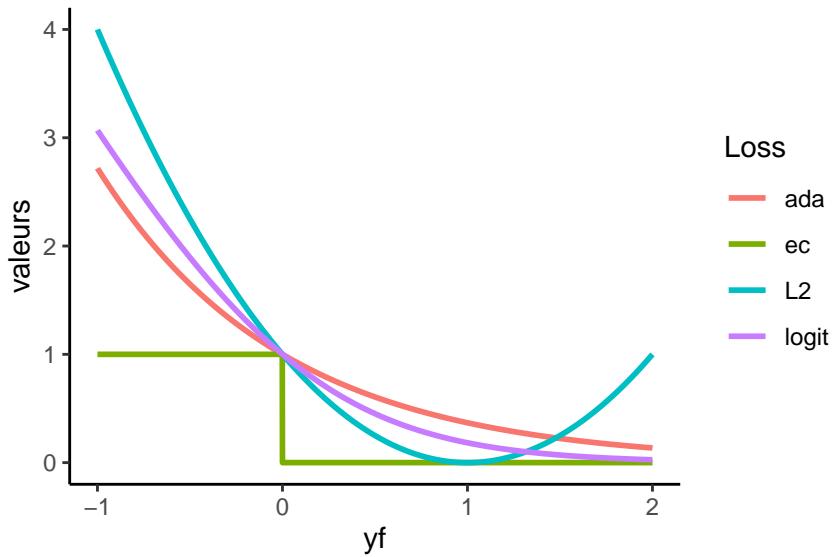
### Choix de la fonction de perte

- 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.  $\ell(y, g(x)) = \exp(-yg(x)) \Rightarrow$  adaboost ;
  2.  $\ell(y, g(x)) = \log(1 + \exp(-2yg(x))) \Rightarrow$  logitboost ;
- Régression avec  $Y$  dans  $\mathbb{R}$  :  $\ell(y, g(x)) = 0.5(y - g(x))^2 \Rightarrow$   $L_2$ -boosting.

## Visualisation des pertes



## Adaboost et logitboost

- Les risques considérés pour *adaboost* et *logitboost* sont

$$\mathcal{R}_{\text{ada}}(g) = \mathbf{E}[\exp(-Yg(X))] \quad \text{et} \quad \mathcal{R}_{\text{logit}}(g) = \mathbf{E}[\log(1 + \exp(-2Yg(X)))].$$

### Propriété

Les risques précédents sont **minimaux** en

$$g^*(x) = \frac{1}{2} \log \frac{\mathbf{P}(Y = 1|X = x)}{\mathbf{P}(Y = -1|X = x)}.$$

### Conséquence

- L'algorithme **FGD** renverra en sortie une suite d'*estimateurs de  $g^*$* .
- Facile d'en déduire des **scores** et **règles de prévision**.

## $L_2$ boosting

- Le risque pour le  $L_2$  *boosting* est

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

- Il est donc minimal en la *fonction de régression*

$$g^*(x) = \mathbf{E}[Y|X = x].$$

- L'algorithme **FGD** renverra en sortie une suite d'*estimateurs de la fonction de régression  $g^*$* .

## 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  $(x_1, u_1), \dots, (x_n, u_n) \Rightarrow h_m$ .
  - c) Mise à jour :  $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$ .

## Adaboost - version 1

- Une étude identique avec la perte *adaboost* montre que l'on retombe sur une algorithme très voisin de l'algorithme adaboost de [Freund and Schapire, 1996].
- *Interprétation similaire* : augmenter le poids des individus mal classés à l'itération précédente pour la mise à jour.

## Adaboost - version 1

### Algorithme [Freund and Schapire, 1996]

**Entrées** : une règle faible,  $M$  nombre d'itérations.

1. Initialiser les poids  $w_i = 1/n$ ,  $i = 1, \dots, n$
2. Pour  $m = 1$  à  $M$  :
  - a) Ajuster la règle faible sur l'échantillon  $d_n$  pondéré par les poids  $w_1, \dots, w_n$ , on note  $g_m(x)$  l'estimateur issu de cet ajustement
  - b) Calculer le taux d'erreur :  $e_m = \frac{\sum_{i=1}^n w_i \mathbf{1}_{y_i \neq g_m(x_i)}}{\sum_{i=1}^n w_i}$ .
  - c) Calculer :  $\alpha_m = \log((1 - e_m)/e_m)$
  - d) Réajuster les poids :  $w_i = w_i \exp(\alpha_m \mathbf{1}_{y_i \neq g_m(x_i)})$ ,  $i = 1, \dots, n$

**Sorties** : l'algorithme de prévision  $\sum_{m=1}^M \alpha_m g_m(x)$ .

## Biais/variance

*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).

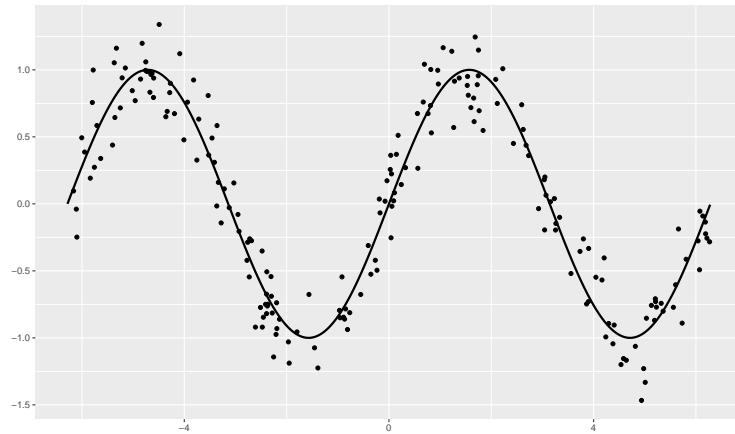
## Comparaison avec random forest

- *Boosting* : réduction de biais.
- *Random Forest* : réduction de variance.

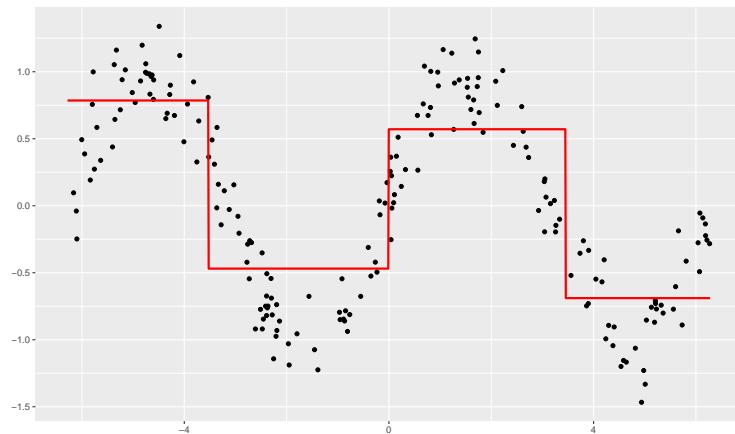
⇒ les arbres sont ajustés de **façon différente** pour ces deux algorithmes. ⇒ dans les deux cas, il faut des **arbres "mauvais"**.

## Illustration

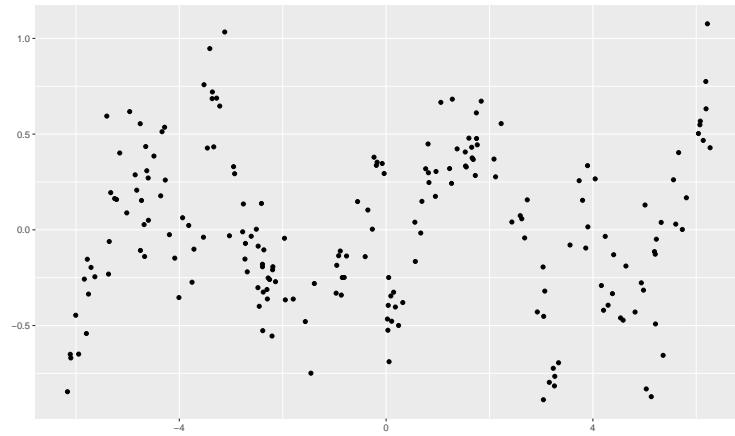
On considère l'échantillon suivant



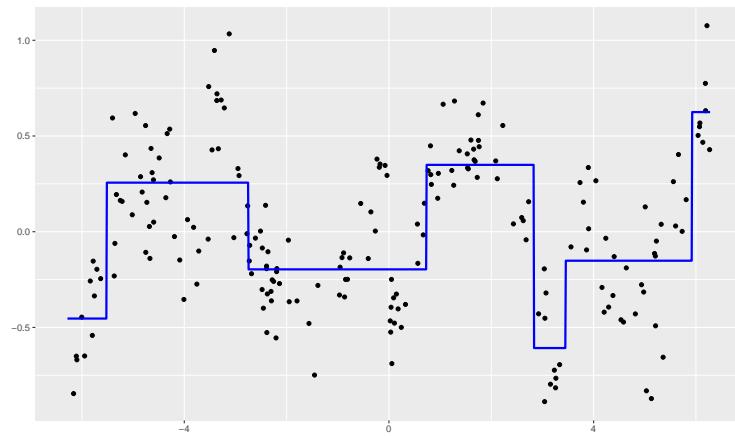
On ajuste un premier arbre simple :



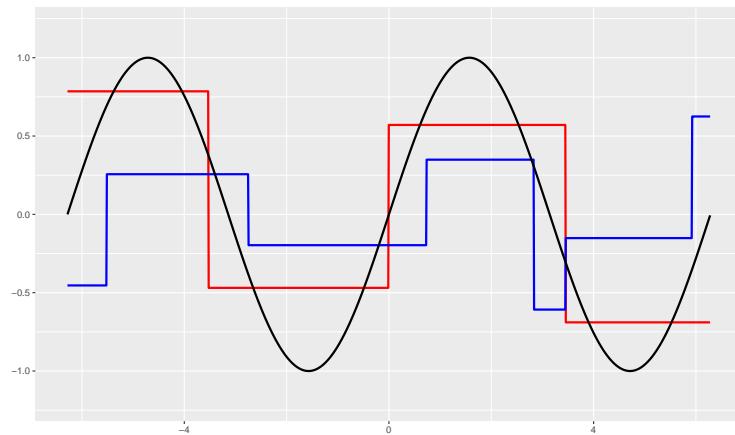
On calcule les résidus :



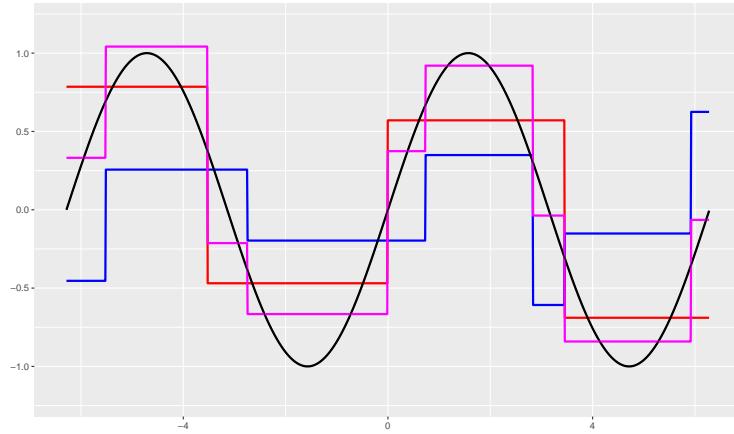
On ajuste un nouvel arbre sur les résidus :



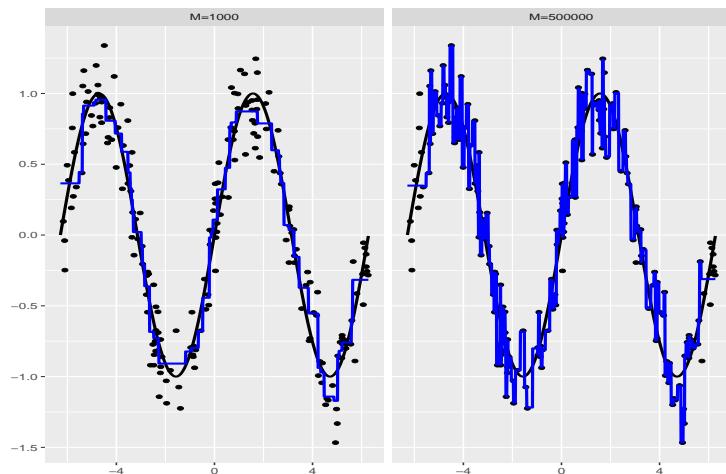
On obtient ainsi 2 arbres :



Que l'on ajoute pour déduire un nouvel estimateur (2 itérations)...



- Pour 1,000 et 500,000 itérations, on obtient :



### *Remarque importante*

L'algorithme *sur-ajuste* si le **nombre d'itérations** est (trop) grand.

### **Choix de $m$ et $\lambda$**

- 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*  $\lambda$  représente le **pas de la descente de gradient**.
- Ce paramètre est **lié à  $m$**  : un  $\lambda$  grand nécessitera peu d'itérations et réciproquement.

### *En pratique*

- On considère 2 ou 3 valeurs (petites) pour  $\lambda$  (0.1, 0.01) ;
- Pour chaque  $\lambda$ , on choisit le meilleur  $m$  en utilisant des techniques de type **validation croisée**.

## En pratique

- L'algorithme étant construit pour une *fonction de perte*  $\ell$  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* :

$$\hat{m} = \operatorname{argmin}_{m \leq M} \mathbf{E}[\ell(Y, g_m(X))].$$

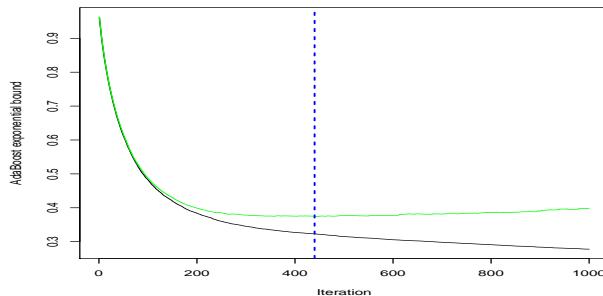
- L'espérance ci-dessus étant *inconnue* en pratique, elle est approchée par des algorithmes de **validation croisée**.

## Le coin R

- La fonction *gbm* du package **gbm** [Ridgeaway, 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 noeuds terminaux des arbres plus 1 (**interaction.depth**)
  4. paramètre de régularisation  $\lambda$  (**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
```



## 2.3 Compléments

### Stochastic gradient boosting

- [Friedman, 1999] propose d'ajouter un caractère *stochastique* à l'algorithme.
- A chaque itération, l'arbre est ajusté sur un échantillon *bootstrap* (tirage d'une proportion  $\eta$  des observations sans remise).
- Option **bag.fraction** dans *gbm*.
- Possibilité d'utiliser des techniques *Out Of Bag* pour estimer les risques (comme pour les forêts aléatoires).

## Importance des variables

- Comme pour les *forêts aléatoire* : **noter** les variables en fonction de leur importance.
- Basé sur la *mesure d'importance des variables* pour une arbre  $T$  de [Breiman et al., 1984] (voir également [Hastie et al., 2009, page 368]) :

$$\mathcal{I}_\ell^2(T) = \sum_{\text{nœuds internes } t} \hat{i}_t^2 \mathbf{1}_{v(t)=\ell}.$$

- Pour le *boosting*, on mesurera l'**importance** en moyennant l'importance pour chaque arbre :

$$\mathcal{I}_\ell^2 = \frac{1}{M} \sum_{m=1}^M \mathcal{I}_\ell^2(h_m).$$

### Remarque

Mesure connue pour être **plus pertinente** que dans le cas d'un seul arbre.

## 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**.
- $\Rightarrow$  Partie 4.2 du tuto

## 3 Bibliographie

### Références

#### Biblio4

- [Aronszajn, 1950] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American Mathematical Society*, 68 :337–404.
- [Breiman, 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 26(2) :123–140.
- [Breiman et al., 1984] Breiman, L., Friedman, J., Olshen, R., and Stone, C. (1984). *Classification and regression trees*. Wadsworth & Brooks.
- [Bühlmann and Hothorn, 2007] Bühlmann, P. and Hothorn, T. (2007). Boosting algorithms : regularization, prediction and model fitting. *Statistical Science*, 22 :477–505.
- [Bühlmann and Yu, 2003] Bühlmann, P. and Yu, B. (2003). Boosting with the  $l_2$  loss : regression and classification. *Journal of American Statistical Association*, 98 :324–339.
- [Cornillon et al., 2014] Cornillon, P., Hengartner, N., and Matzner-Løber, E. (2014). Recursive bias estimation for multivariate regression smoothers. *ESAIM : Probability and Statistics*, 18(483-502).
- [Devroye and Krzyżak, 1989] 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 and Schapire, 1996] Freund, Y. and Schapire, R. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*.
- [Freund and Schapire, 1997] 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 and Schapire, 1999] Freund, Y. and Schapire, R. (1999). A short introduction to boosting. *Journal of Japanese Society for Artificial Intelligence*, 14(5) :771–780.
- [Friedman, 1999] Friedman, J. H. (1999). Stochastic gradient boosting. Technical report, Stanford University.
- [Friedman, 2001] Friedman, J. H. (2001). Greedy function approximation : A gradient boosting machine. *Annals of Statistics*, 29 :1189–1232.
- [Fromont, 2015] Fromont, M. (2015). Apprentissage statistique. Université Rennes 2, diapos de cours.
- [Genuer, 2010] Genuer, R. (2010). *Forêts aléatoires : aspects théoriques, sélection de variables et applications*. PhD thesis, Université Paris XI.
- [Györfi et al., 2002] Györfi, L., Kohler, M., Krzyzak, A., and Harro, W. (2002). *A Distribution-Free Theory of Nonparametric Regression*. Springer.
- [Hastie et al., 2009] Hastie, T., Tibshirani, R., and Friedman, J. (2009). *The Elements of Statistical Learning : Data Mining, Inference, and Prediction*. Springer, second edition.
- [Ridgeway, 2006] Ridgeway, G. (2006). Generalized boosted models : A guide to the gbm package.
- [Stone, 1977] Stone, C. J. (1977). Consistent nonparametric regression. *Annals of Statistics*, 5 :595–645.
- [Vapnik, 2000] Vapnik, V. (2000). *The Nature of Statistical Learning Theory*. Springer, second edition.
- [Vert, 2014] 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>.