

# **Agrégation de modèles et SVM**

`masedki.github.io`

ENSAI, Rennes

jeu. 21 févr. 2019

# Outline

## 1. Introduction

## 2. Régression vs classification supervisée

## 3. Arbres de décision uniques

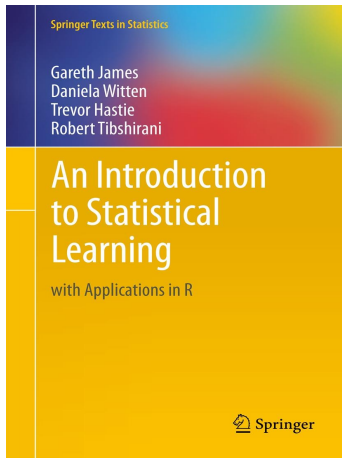
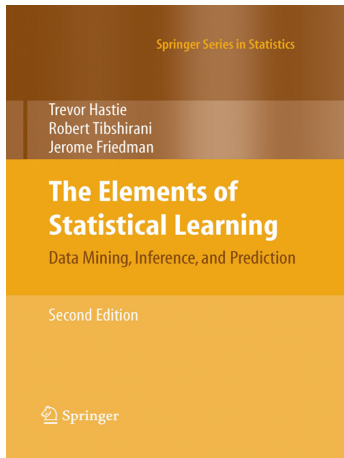
## 4. Agrégation par la moyenne : bagging

## 5. Agrégation séquentielle : boosting

## 6. Justification du boosting : minimisation de risque empirique

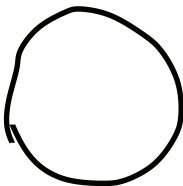
## 7. Méthodes à noyaux : SVM et SVR

# Références



## Problème d'apprentissage

0	0	0	0	0
1	1	1	1	1
2	2	2	2	2
3	3	3	3	3
4	4	4	4	4
5	5	5	5	5
6	6	6	6	6
7	7	7	7	7
8	8	8	8	8
9	9	9	9	9



- Reconnaissance de chiffres manuscrits ?

0, 1, 2 ... ?

# Problèmes d'apprentissage statistique

1. Identifier les facteurs de risque du cancer de la prostate
2. Prédire si une personne est sujette aux crises cardiaques, à partir de mesures cliniques, son régime et des données démographiques
3. Classification d'échantillons de tissus dans différents types de cancer, en fonction de données d'expression de gènes
4. Classer des images de tumeurs

## Question

- 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.

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

# Outline

1. Introduction

**2. Régression vs classification supervisée**

3. Arbres de décision uniques

4. Agrégation par la moyenne : bagging

5. Agrégation séquentielle : boosting

6. Justification du boosting : minimisation de risque empirique

7. Méthodes à noyaux : SVM et SVR

# Réprésentation du problème

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

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

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



# Régression

- Un **échantillon i.i.d**  $(X_1, Y_1), \dots, (X_n, Y_n)$  à valeurs dans  $\mathbb{R}^p \times \mathbb{R}$ .
- **Objectif** : Prédire ou expliquer la variable  $Y$  à partir d'une nouvelle observation  $X$ .
- **Méthode** : construire un **régresseur**

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

- **Critère** de performance pour  $m$  : l'**erreur de prévision**

$$\mathbb{E} \left[ (Y - m(X))^2 \right].$$

# La fonction de régression

- Un **champion**

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

appelé **fonction de régression**.

- Pour toute autre fonction  $m$ , on a

$$\mathbb{E} \left[ (Y - m^*(X))^2 \right] \leq \mathbb{E} \left[ (Y - m(X))^2 \right]$$

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

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

# La classification binaire

- Un **échantillon i.i.d**  $(X_1, Y_1), \dots, (X_n, Y_n)$  à valeurs dans  $\mathbb{R}^p \times \{0, 1\}$ .
- **Objectif** : Prédire ou expliquer la variable  $Y$  à partir d'une nouvelle observation  $X$ .
- **Méthode** : construire une **règle classification**

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

- **Critère** de performance pour  $g$  : **probabilité d'erreur ou de mauvais classement**

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

# La règle de Bayes

- Un autre **champion** :

$$g^*(x) = \begin{cases} 0 & \text{si } \mathbb{P}(Y = 0|X = x) \geq \mathbb{P}(Y = 1|X = x) \\ 1 & \text{sinon,} \end{cases}$$

appelé **règle de Bayes**.

- Quelque soit la règle de décision  $g$ , nous avons

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

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

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

# Notations

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

# Outline

1. Introduction

2. Régression vs classification supervisée

**3. Arbres de décision uniques**

4. Agrégation par la moyenne : bagging

5. Agrégation séquentielle : boosting

6. Justification du boosting : minimisation de risque empirique

7. Méthodes à noyaux : SVM et SVR

# Méthodes basées sur des arbres

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

## Pours et contres

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

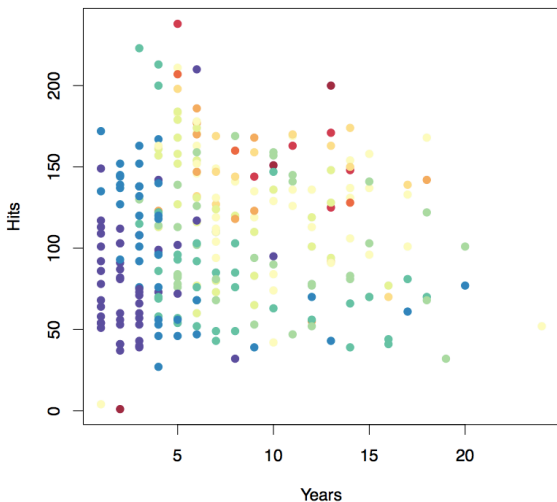


# Les bases des arbres de décision

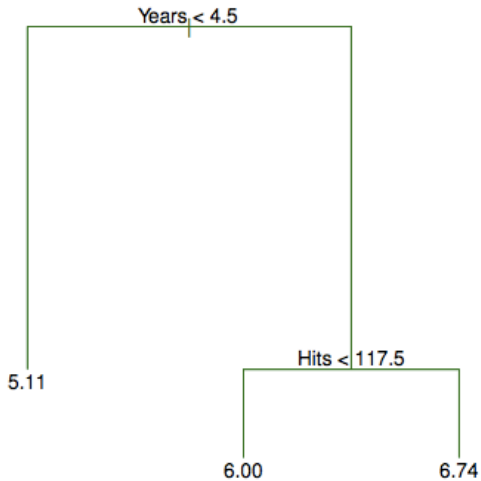
- Les arbres de décision sont utiles aussi bien pour des problèmes de régression que de classification.
- Nous commençons par présenter des problèmes de régression et nous viendrons ensuite à la classification.

## Données de salaire au baseball: comment les stratifier ?

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



## L'arbre de décision sur ces données

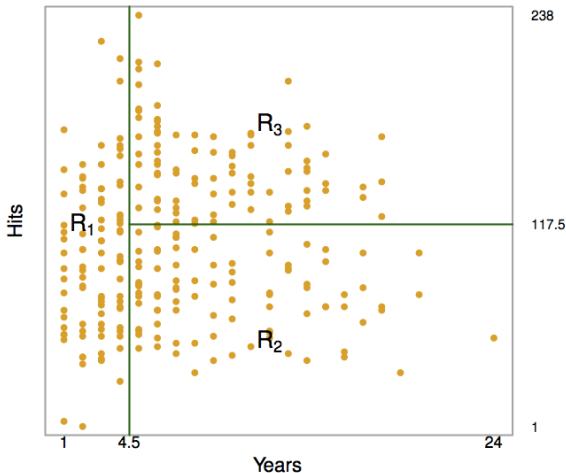


## Détails de la précédente figure

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

## Résultats

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



## Interprétation des résultats

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

# Détails sur la construction de l'arbre

## Algorithme CART (Classification and Regression Trees)

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

## Détails sur la construction de l'arbre (suite)

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

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

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



## Détails sur la construction de l'arbre (suite)

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

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

Initialisation

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

Récurrance sur chaque nœud

On partitionne chaque nœud en deux classes:

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

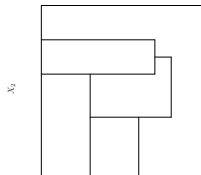
en cherchant  $j$  et  $s$  qui minimisent

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

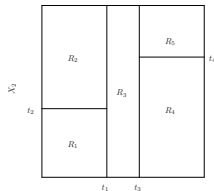
où  $\hat{y}_m = \text{ave}(y_i | x_i \in \mathcal{R}_m(j, s))$  est la réponse moyenne des données d'apprentissage qui tombent dans la région  $\mathcal{R}_m(j, s)$  pour  $m = 1$  ou  $m = 2$ .

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

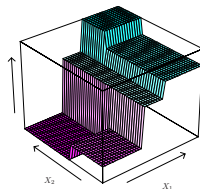
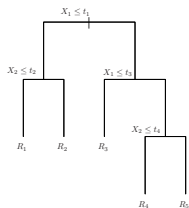
# Exemples de récurrence binaire



$X_1$



$X_1$



## Exemples de récurrence binaire

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

## Algorithme (suite...)

### Phase 1 : Construction de $T_0$

Initialisation

[...]

Récurrence sur chaque nœud

[...]

Terminaison

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

## Critère d'arrêt

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

## Une première idée

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

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

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

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

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

## Une première idée (suite)

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



# Sur-apprentissage

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

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

**Solution :** élagage de  $T_0$  appelé *Cost complexity pruning*

# Élagage

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

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

## Élagage : détails

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

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

où

- $|T|$  est nombre de feuilles de  $T$
- $N_m(T) = \text{Card} \{x_i \in \mathcal{R}_m(T)\}$  et  $Q_m(T) = \frac{1}{N_m(T)} \sum_{x_i \in \mathcal{R}_m(T)} (y_i - \hat{y}_m)^2$
- $\hat{y}_m = \text{ave}(y_i | x_i \in \mathcal{R}_m(T))$
- On notera  $T_\alpha$  le sous-arbre qui minimise  $\mathcal{C}_\alpha(T)$  à  $\alpha$  fixé
- Rôle de  $\alpha$  ? Cas particuliers  $\alpha = 0$  et  $\alpha \rightarrow +\infty$  !!

## Élagage : Calcul des minima $T_\alpha$ du critère pénalisé

1. On construit une suite d'arbres itérativement

- On part de  $T_0$
- À chaque étape, on supprime le nœud interne de tel sorte à produire la plus petite augmentation de

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

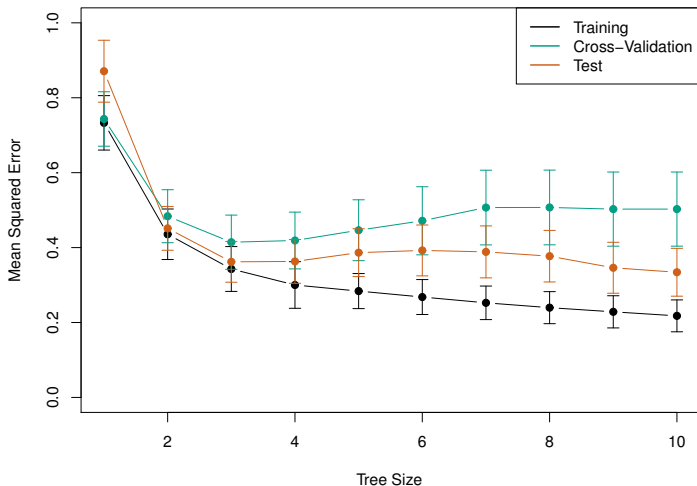
- On s'arrête lorsque l'arbre est réduit à un seul nœud (racine)

2. Tous les minima  $T = T_\alpha$  des fonctions  $T \mapsto \mathcal{C}_\alpha(T)$  sont dans cette suite

## Élagage : Choix de $\alpha$ par validation croisée $K$ -folds

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

## Illustration : *Hitters* dataset



## Exemple : boston dataset

```
rm(list=ls())
require(MASS)
require(rpart)
require(rpart.plot)
data("Boston")
set.seed(1)
train = sample(1:nrow(Boston), round(0.75*nrow(Boston)))
Boston.tr = Boston[train,]
Boston.te = Boston[-train,]
```

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

# Arbres de classification

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



## Classification : différence avec la régression

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

## Taux d'erreur pour la classification

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

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

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

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

## Mesures d'impureté

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

- **Taux de mauvais classement :**

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

- **Indice de Gini :**

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

- **Entropie :**

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

## Mesures d'impureté

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

# Post-election survey data from the 2004 British Election Survey

```
rm(list=ls())
require(rpart)
require(rpart.plot)
require(caret)
require(foreign)
bes <- read.dta("bes.dta")
bes <- na.omit(bes)
head(bes)
table(bes$in_school)
bes$in_school <- ifelse (bes$in_school != 1, 0, bes$in_school)
table(bes$in_school)
# data manipulation
categorical <- c("Turnout", "Vote2001", "Gender", "PartyID", "Telephone", "edu15",
                "edu16", "edu17", "edu18", "edu19plus", "in_school", "in_uni")
# declare factor variables
bes[, categorical] <- lapply(bes[, categorical], factor)
```

- Reprendre les étapes de l'exemple de régression pour ajuster un arbre de décision visant à prédire (participation à l'élection) Turnout en fonction des autres variables présentes dans le jeu de données à l'exception de CivicDutyScores.
- Justifier l'élimination de la variable CivicDutyScores du modèle.
- Attention au calcul de l'erreur de prédiction sur le jeu de données test.

## Avantages et inconvénients des arbres

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

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

# Outline

1. Introduction

2. Régression vs classification supervisée

3. Arbres de décision uniques

**4. Agrégation par la moyenne : bagging**

5. Agrégation séquentielle : boosting

6. Justification du boosting : minimisation de risque empirique

7. Méthodes à noyaux : SVM et SVR

## Agrégation par Bagging

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



## Bagging pour la régression

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

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

## Bagging pour la classification

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

## Intuitivement

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

## Erreur *Out-Of-Bag* (OOB)

- Il y a une façon simple d'estimer l'erreur de test quand on fait du bagging.
- La clé du bagging est l'entraînement de nombreux  $\hat{f}(x)$  sur des échantillons bootstraps. On peut donc utiliser les observations hors du  $b^{\text{ième}}$  bootstrap pour évaluer chaque  $\hat{f}^{*b}(x)$ .
- Ce qui donne l'algorithme ci-dessous.
  1. Pour chaque observation  $(x_i, y_i)$ , calculer  $\hat{y}_i^{\text{OOB}}$  la prédiction en n'utilisant que les estimateurs  $\hat{f}^{*b}(x)$  qui n'ont pas vu cette observation dans leur entraînement
  2. Évaluer l'erreur entre  $\hat{y}_i^{\text{OOB}}$  et les  $y_i$  (erreur quadratique moyenne ou taux de mauvaise classification)

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

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

## Estimation de l'erreur de test par OOB

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

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

- L'erreur de classification OOB

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

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

## Mesurer l'importance des variables

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

## Garanties théoriques : un peu de notations

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

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

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

$$\mathbb{E}[(\hat{m}(x) - m^*(x))^2] = \left(\mathbb{E}(\hat{m}(x)) - m^*(x)\right)^2 + \text{Var}(\hat{m}(x)).$$

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

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

même biais mais la variance diminue



## Garanties théoriques : bootstrap

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

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

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

## Garanties théoriques : premier résultat

- Deux techniques sont généralement utilisées pour générer les échantillons bootstrap
  1.  $\theta_b(\mathcal{D}_n)$  est obtenu en tirant  $n$  observations avec remise dans  $\mathcal{D}_n$ , chaque observation ayant la même probabilité d'être tirée  $\frac{1}{n}$ .
  2.  $\theta_b(\mathcal{D}_n)$  est obtenu en tirant  $\ell$  observations (avec ou sans remise) dans  $\mathcal{D}_n$  avec  $\ell < n$ .
- ***Théorème de Biau & Devroye (2010)*** Si  $\ell = \ell_n$  tel que  $\lim_{n \rightarrow \infty} \ell_n = +\infty$  et  $\lim_{n \rightarrow \infty} \frac{\ell_n}{n} = 0$  alors l'estimateur  $\hat{m}(x) = \mathbb{E}_\theta [\hat{m}(x, \theta) | \mathcal{D}_n]$  est universellement consistant.

## Garanties théoriques : biais et variance

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

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

*Par conséquent*

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

## Exemple : reprendre les deux exemples Boston et bes

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

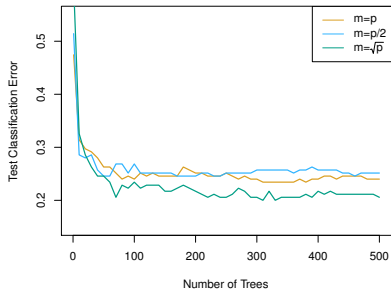
## Forêts aléatoires très proche du bagging

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

# Forêts aléatoires

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

## Illustration : données d'expression de gènes



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

## Exemple d'application

Sur le jeu de données `credit`, ajuster un modèle de forêt aléatoire pour prédire la variable `default.payment.next.month` à l'aide des autres variables

- Identifier les arguments à contrôler dans la fonction `randomForest`
- Identifier les courbes tracées par la fonction `plot`
- La fonction `varImpPlot` permet d'accéder au rôle des variables
- Utiliser les fonctions `tune` et `train` des packages `e1071` et `caret`

```
## load data
# convert categorical variables
load("credit.rda")
credit$SEX <- as.factor(credit$SEX)
credit$EDUCATION <- as.factor(credit$EDUCATION)
credit$MARRIAGE <- as.factor(credit.data$MARRIAGE)
credit$default.payment.next.month <- as.factor(credit$default.payment.next.month)
```



# Outline

1. Introduction

2. Régression vs classification supervisée

3. Arbres de décision uniques

4. Agrégation par la moyenne : bagging

**5. Agrégation séquentielle : boosting**

6. Justification du boosting : minimisation de risque empirique

7. Méthodes à noyaux : SVM et SVR

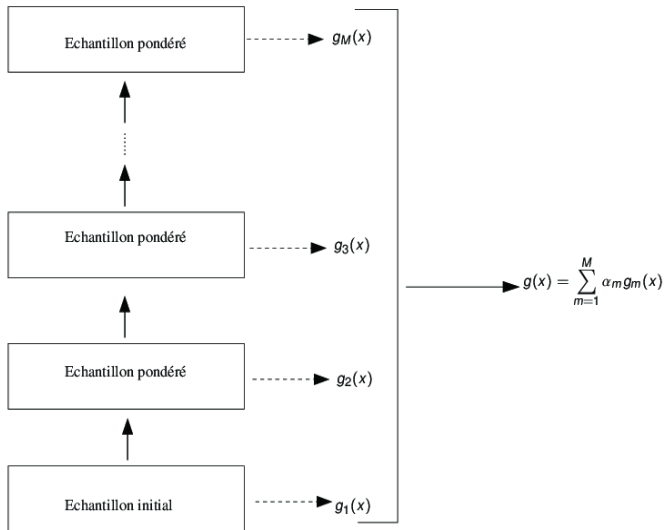
# Principe

- **Premier** algorithme de “boosting” [Freund and Schapire, 1997].
- Construire une famille de **règles** qui sont ensuite agrégées.
- Processus **récuratif** : la règle construite à l’étape  $k$  dépend de celle construite à l’étape  $k - 1$

# Principe

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

# Schéma



# Notion de règle faible

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

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

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

- Exemples :** règle des 1-NN, arbres à 2 feuilles.

# Algorithme dit Adaboost.M1

**Input :** - Une observation  $x$  à prédire et l'échantillon  $d_n = (x_1, y_1), \dots, (x_n, y_n)$  -

Une règle de classification faible et  $M$  le nombre d'itérations

## Algorithm of [Freund and Schapire 1997]:

1. Initialiser les poids  $w_i = \frac{1}{n}, i = 1, \dots, n$
2. **Pour**  $m = 1$  à  $M$ :
  - 2.1 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
  - 2.2 Calcul du taux d'erreur :

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

2.3 Calcul de :  $\alpha_m = \log \left( \frac{1-e_m}{e_m} \right)$

2.4 Réajuster les poids :

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

**Output:**

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

# Commentaires

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

## Quelques garanties théoriques : contrôle de l'erreur empirique

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

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

- $\gamma_m$  désigne le **gain** de la règle  $g_m$  par rapport à une règle **pûrement aléatoire**

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

**Propriété: [Freund and Schapire, 1999]**

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

**Conséquence :**

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



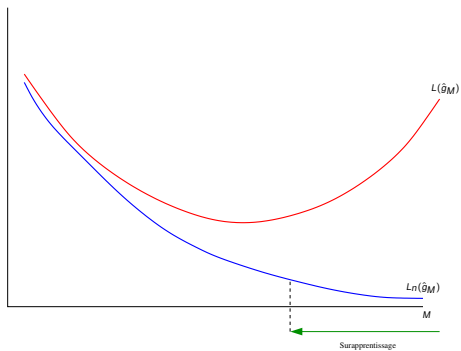
# Controle de l'erreur de généralisation

## Propriétés [Freund and Schapire, 1999]

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

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

# Surapprentissage



**Conséquence :** Il est important de bien choisir  $M$ .

# Outline

1. Introduction
2. Régression vs classification supervisée
3. Arbres de décision uniques
4. Agrégation par la moyenne : bagging
5. Agrégation séquentielle : boosting
- 6. Justification du boosting : minimisation de risque empirique**
7. Méthodes à noyaux : SVM et SVR

## Pertes théorique et empirique

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

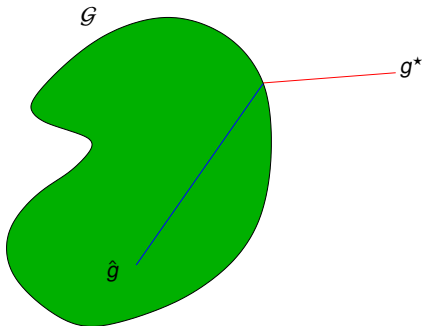
$$L(g) = \mathbb{P}(Y \neq g(X)).$$

**Problème** : la fonction de perte n'est pas calculable

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

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

## Erreurs d'estimation et d'approximation



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

# Risque convexifié

**Problème** : la fonction

$$\mathcal{G} \rightarrow \mathbb{R}$$
$$g \mapsto \frac{1}{n} \sum_{i=1}^n \mathbf{1}_{g(X_i) \neq Y_i}$$

est généralement difficile à minimiser.

**Idée** : trouver une autre fonction de perte  $\ell : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R}$  telle que

$$\mathcal{G} \rightarrow \mathbb{R}$$
$$g \mapsto \frac{1}{n} \sum_{i=1}^n \ell(Y_i, g(X_i))$$

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

## Fonction de perte

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

# Fonction de perte

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



# Fonction de perte

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

## Récapitulatif

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

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

## Récapitulatif

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

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

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

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

## Récapitulatif

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

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

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

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

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

## Récapitulatif

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

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

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

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

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

## Un petit rappel

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

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

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

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

# Newton Raphson

- On note  $\mathbf{g}_m = (g_m(x_1), \dots, g_m(x_n))$ , et

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

# Newton Raphson

- On note  $\mathbf{g}_m = (g_m(x_1), \dots, g_m(x_n))$ , et

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

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

$$\mathbf{g}_m = \mathbf{g}_{m-1} - \lambda \nabla J(\mathbf{g}_{m-1}),$$

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



# Newton Raphson

- On note  $\mathbf{g}_m = (g_m(x_1), \dots, g_m(x_n))$ , et

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

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

$$\mathbf{g}_m = \mathbf{g}_{m-1} - \lambda \nabla J(\mathbf{g}_{m-1}),$$

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

- Inconvénients**

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

# Boosting par descente du gradient

## Entrées :

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

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

2. **Pour**  $m = 1$  à  $M$  :

2.1 Calculer l'opposé du gradient  $-\frac{\partial}{\partial g(x_i)} \ell(y_i, g_m(x_i))$  et l'évaluer aux points  $g_{m-1}(x_i)$  :

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g_m(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)}, \quad i = 1, \dots, n.$$

2.2 Ajuster la règle faible sur l'échantillon  $(x_1, U_1), \dots, (x_n, U_n)$ , on note  $h_m$  la règle ainsi définie.

2.3 Mise à jour :  $g_m(x) = g_{m-1}(x) + \lambda h_m(x)$ .

3. **Sortie** : La règle  $\hat{g}_M(x) = g_m(x)$ .

## Commentaires

- La sortie  $\hat{g}_M(x)$  est un **réel**. Si on cherche à prédire le label de  $x$ , on pourra utiliser la règle  $\hat{y} = \text{signe}(\hat{g}_M(x))$ .

## Commentaires

- La sortie  $\hat{g}_M(x)$  est un **réel**. Si on cherche à prédire le label de  $x$ , on pourra utiliser la règle  $\hat{y} = \text{signe}(\hat{g}_M(x))$ .
- Pour le choix  $\lambda = 1$  et  $\ell(y, g(x)) = \exp(-yg(x))$ , cet **algorithme coïncide (quasiment) avec Adaboost**.

## Commentaires

- La sortie  $\hat{g}_M(x)$  est un **réel**. Si on cherche à prédire le label de  $x$ , on pourra utiliser la règle  $\hat{y} = \text{signe}(\hat{g}_M(x))$ .
- Pour le choix  $\lambda = 1$  et  $\ell(y, g(x)) = \exp(-yg(x))$ , cet **algorithme coïncide (quasiment) avec Adaboost**.
- Le **choix de  $\lambda$**  est lié au choix du nombre d'itérations  $M$ . Il permet de *contrôler* la vitesse à laquelle on minimise la fonction

$$\frac{1}{n} \sum_{i=1}^n \ell(y_i, g(x_i)).$$

$\implies$  lorsque  $\lambda \nearrow M \searrow$  et réciproquement.

## Règles faibles

- Comme pour Adaboost, la règle utilisée dans l'algorithme doit être **faible** (légèrement meilleure que le hasard).

## Règles faibles

- Comme pour Adaboost, la règle utilisée dans l'algorithme doit être **faible** (légèrement meilleure que le hasard).
- **Booster** une règle non faible se révèle généralement **peu performant**.

# Règles faibles

- Comme pour Adaboost, la règle utilisée dans l'algorithme doit être **faible** (légèrement meilleure que le hasard).
- **Booster** une règle non faible se révèle généralement **peu performant**.
- Il est recommandé d'utiliser une règle possédant un **biais élevé** est une **faible variable** (booster permet de réduire le biais, pas la variance).



# Règles faibles

- Comme pour Adaboost, la règle utilisée dans l'algorithme doit être **faible** (légèrement meilleure que le hasard).
- **Booster** une règle non faible se révèle généralement **peu performant**.
- Il est recommandé d'utiliser une règle possédant un **biais élevé** est une **faible variable** (booster permet de réduire le biais, pas la variance).
- On utilise souvent des **arbres** comme règle faible. Pour posséder un biais élevé, on utilisera donc des **arbres avec peu de noeuds terminaux**.

## Paramètres de la fonction gbm du package gbm de [Ridgeway, 2006]

1. fonction de perte `distribution`
2. nombre d'itérations qu'on a noté  $M$  `n.trees`
3. nombre de noeuds terminaux des arbres plus 1 qu'on a noté  $K$  `interaction.depth`
4. paramètre de régularisation  $\lambda$  `shrinkage`

## Logitboost : gradient boosting pour la classification

Si  $Y$  est à valeurs dans  $\{0, 1\}$ . La variable  $Y|X = x$  suit une loi de Bernoulli de paramètre  $p(x) = \mathbb{P}(Y = 1|X = x)$ . La vraisemblance d'une observation  $(x, y)$  s'écrit

$$p(x) = \frac{1}{1 + \exp(-x'\beta)} = \frac{\exp(x'\beta)}{1 + \exp(-x'\beta)}.$$

Souvent on estime  $\beta$  par maximum de vraisemblance.

Le modèle logitboost repose une approche similaire  $g : \mathbb{R}^p \rightarrow \mathbb{R}$ , on pose

$$p(x) = \frac{\exp(g(x))}{\exp(g(x)) + \exp(-g(x))} = \frac{1}{1 + \exp(-2g(x))},$$

ce qui donne

$$g(x) = \frac{1}{2} \log \left( \frac{p(x)}{1 - p(x)} \right).$$

Maximiser la log-vraisemblance revient à minimiser son opposé

$$-\left(y \log(p(x)) + (1 - y) \log(1 - p(x))\right) = \log \left(1 + \exp(-2\tilde{y}g)\right)$$

où  $\tilde{y} = 2y - 1 \in \{-1, 1\}$ .

# Logitboost

- On applique l'algorithme de boosting par descente du gradient à la fonction de perte

$$\ell(y, g) = \log \left( 1 + \exp \left( - 2\tilde{y}g \right) \right).$$

- Après  $M$  itérations, on obtient l'estimateur  $\hat{g}_M$  de

$$g^* = \underset{g}{\operatorname{argmin}} \mathbb{E} \left[ \log \left( 1 + \exp \left( - 2\tilde{Y}g \right) \right) \right].$$

- On peut montrer que  $g^*(x) = \frac{1}{2} \log \left( \frac{p(x)}{1-p(x)} \right)$ , on déduit un estimateur  $\hat{p}_M(x)$  de  $p(x)$  en posant

$$\hat{p}_M(x) = \frac{\exp(\hat{g}_M(x))}{\exp(\hat{g}_M(x)) + \exp(-\hat{g}_M(x))} = \frac{1}{1 + \exp(-2\hat{g}_M(x))}.$$

- On obtient la règle de classification

$$\hat{y} = \begin{cases} 1 & \text{si } \hat{g}_M(x) \geq 0 \iff \hat{p}_M(x) \geq 0.5 \\ 0 & \text{si } \hat{g}_M(x) < 0 \iff \hat{p}_M(x) < 0.5. \end{cases}$$

## $L_2$ boosting : gradient boosting pour la régression

- On s'intéresse à la régression. On désigne par  $f : \mathbb{R}^p \rightarrow \mathbb{R}$  un régresseur *faible*. Plus précisément un régresseur (fortement) biaisé, par exemple :
  - un arbre de décision à deux noeuds terminaux (stumps)
  - un estimateur à noyau avec une grande fenêtre
- Le  $L_2$  boosting consiste à appliquer l'algorithme de boosting par descente du gradient avec la fonction de perte quadratique

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

- Après  $M$  itérations, l'algorithme fournit un estimateur  $\hat{f}_M$  de

$$f^* = \underset{f}{\operatorname{argmin}} \mathbb{E} \left[ \frac{1}{2} (Y - f(X))^2 \right],$$

c'est-à-dire la fonction de régression  $f^*(x) = \mathbb{E}[Y|X = x]$ .

- Les variables  $U_i$  de l'étape 2.1 du boosting par descente du gradient s'écrivent  $U_i = y_i - f_{m-1}(x_i)$ . L'étape 2.2 consiste donc simplement à faire une régression sur les résidus du modèle construit à l'étape  $m - 1$ .

# Bilan

- Les algorithmes adaboost, logitboost et  $L_2$  boosting sont donc construits selon le même schéma : ils fournissent un estimateur de  $f^*$  ou  $g^*$  qui minimise la version empirique de l'espérance d'une fonction de perte  $\ell$ .
- Récapitulatif

$\ell(y, f)$ ou $\ell(y, g)$	$f^*$ ou $g^*$	Algorithme
$\exp [-2\tilde{y}g]$	$\frac{1}{2} \log \left( \frac{p(x)}{1-p(x)} \right)$	Adaboost
$\log \left( 1 + \exp [-2\tilde{y}g] \right)$	$\frac{1}{2} \log \left( \frac{p(x)}{1-p(x)} \right)$	Logitboost
$\frac{1}{2}(y - f)^2$	$\mathbb{E}[Y X = x]$	$L_2$ boosting

# Gradient Boosting Algorithm avec arbres

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

2. **Pour**  $m = 1$  à  $M$  :

- Calculer l'opposé du gradient  $-\frac{\partial}{\partial g(x_i)} \ell(y_i, g_m(x_i))$  et l'évaluer aux points  $g_{m-1}(x_i)$  :

$$U_i = -\frac{\partial}{\partial g(x_i)} \ell(y_i, g_m(x_i)) \Big|_{g(x_i)=g_{m-1}(x_i)}, \quad i = 1, \dots, n.$$

- Ajuster un arbre à  $K$  noeuds terminaux sur l'échantillon  $(x_1, U_1), \dots, (x_n, U_n)$ .
- Calculer la prévision optimale pour chaque noeuds  $\rho_1, \dots, \rho_K$  : Mise à jour :

$$\rho_k = \operatorname{argmin}_{\rho} \sum_{x_i \in R_k} \ell(y_i, g_{m-1}(x_i) + \rho),$$

où  $R_k$  contient l'ensemble des  $x_i$  qui appartiennent au  $k^{\text{ième}}$  noeud de l'arbre.

- Mise à jour :  $g_m(x) = g_{m-1}(x) + \lambda \rho_{k(x)}$ ,  $k(x)$  désignant le numéro du noeud qui contient  $x$ .

3. **Sortie** : la règle  $\hat{g}_M(x)$ .

# Boosting par descente du gradient pour la régression

1. Fixer  $\hat{f}(x) = 0$  et  $r_i = y_i$  pour tout  $i$  de l'ensemble d'entraînement.
2. Pour  $m = 1, \dots, M$  faire
  - Ajuster un arbre  $\hat{f}_m$  à  $d$  nœuds internes ( $d + 1$  feuilles) pour prédire les  $r_i$  avec  $x_i$
  - Mettre à jour  $\hat{f}$  en ajoutant ce nouvel arbre (à un coefficient  $\lambda$  de réduction près)

$$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_m(x)$$

- Mettre à jour les résidus

$$r_i \leftarrow r_i - \lambda \hat{f}_m(x_i)$$

3. Retourner le modèle

$$\hat{f}(x) = \sum_{m=1}^M \lambda \hat{f}_m(x).$$



## Application : jeu de données spam

Lien vers UCI : <https://archive.ics.uci.edu/ml/datasets/spambase>

- Sur 4601 mails, on a pu identifier 1813 spams
- On a également mesuré sur chacun de ces mails un ensemble de 57 variables.

Expliquer la variable  $y$  (spam ou pas) en fonction des 57 variables

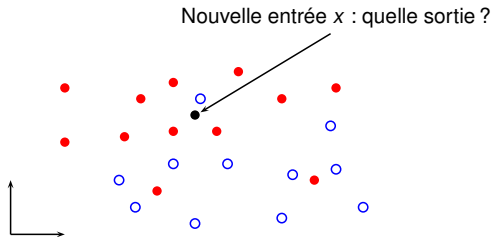
# Outline

1. Introduction
2. Régression vs classification supervisée
3. Arbres de décision uniques
4. Agrégation par la moyenne : bagging
5. Agrégation séquentielle : boosting
6. Justification du boosting : minimisation de risque empirique
- 7. Méthodes à noyaux : SVM et SVR**

# Support Vector Machine : quoi et pourquoi ?

Une **SVM (Support Vector Machine)** ou **Machine à Vecteurs Supports** est une famille d'algorithmes d'apprentissage supervisé pour des problèmes de discrimination (ou de régression).

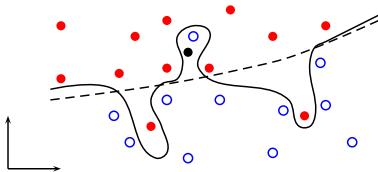
Exemple jouet : classification binaire en dimension 2



# Support Vector Machine : quoi et pourquoi ?

Fondements mathématiques solides  $\Rightarrow$  bonnes **propriétés de généralisation** i.e. bon compromis la classification des données et la prédiction de la sortie associée à une nouvelle entrée  $x$ .

Exemple de règle n'ayant pas de bonnes propriétés de généralisation



**!!! Phénomène de sur-apprentissage (ou overfitting) !!!**

particulièrement présent en grande dimension ou pour des règles de discrimination non linéaires complexes.

# SVM linéaire pour des données séparables

**Données linéairement séparables** : On considère des données à valeurs dans  $\mathbb{R}^p$ , muni du produit scalaire usuel  $\langle \cdot, \cdot \rangle$ .

Les données observées  $(x_1, y_1), \dots, (x_n, y_n)$  sont dites **linéairement séparables** s'il existe  $(w, b)$  tel que pour tout  $i$  :

$$y_i = 1 \quad \text{si} \quad \langle w, x_i \rangle + b > 0,$$

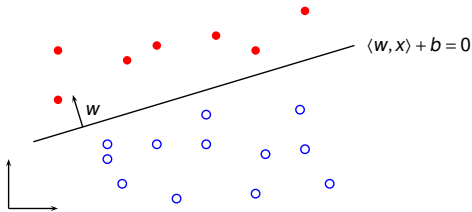
$$y_i = -1 \quad \text{si} \quad \langle w, x_i \rangle + b < 0,$$

Autrement dit

$$\forall i = 1, \dots, n \quad y_i (\langle w, x_i \rangle + b) > 0.$$

# Hyperplan séparateur

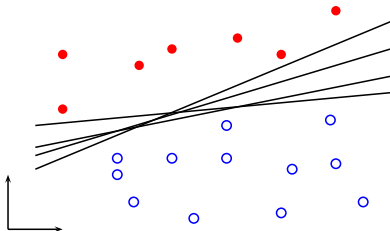
L'équation  $\langle w, x \rangle + b = 0$  définit un hyperplan séparateur de vecteur orthogonal  $w$ .



La règle de classification  $g_{w,b}(x) = \mathbf{1}_{\langle w, x \rangle + b \geq 0} - \mathbf{1}_{\langle w, x \rangle + b < 0}$  est une règle de classification linéaire potentielle.

## Choix de la règle de décision ?

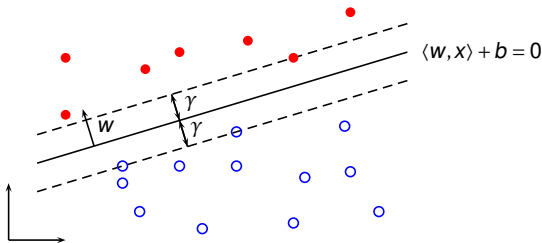
- **Problème:** une infinité d'hyperplans séparateurs  $\Rightarrow$  une infinité de règles de discrimination linéaires potentielles !



- **Laquelle choisir ?**

# Hyperplan séparateur de marge maximale

**La réponse (Vapnik)** : La règle de discrimination linéaire ayant les meilleures propriétés de généralisation correspond à l'hyperplan séparateur de **marge maximale**  $\gamma$ .





## La marge maximale

- Soit deux entrées de l'ensemble d'apprentissage (re)notées  $x_1$  et  $x_{-1}$  de sorties respectives 1 et -1, se situant sur les frontières définissant la marge. L'hyperplan séparateur correspondant se situe à mi-distance entre  $x_1$  et  $x_{-1}$ . La marge s'exprime donc comme suit :

$$\gamma = \frac{1}{2} \frac{\langle w, x_1 - x_{-1} \rangle}{\|w\|}.$$

- **Remarque:** pour tout  $\kappa \neq 0$ , les couples  $(\kappa w, \kappa b)$  et  $(w, b)$  définissent le même hyperplan.
- L'hyperplan  $\langle w, x \rangle + b = 0$  est dit en forme canonique relativement à un ensemble de vecteurs  $x_1, \dots, x_m$  si  $\min_{i=1, \dots, m} |\langle w, x_i \rangle + b| = 1$ .
- L'hyperplan séparateur est en forme canonique relativement aux vecteurs  $\{x_1, x_{-1}\}$  s'il est défini par  $(w, b)$  avec  $\langle w, x_1 \rangle + b = 1$  et  $\langle w, x_{-1} \rangle + b = -1$ . On a alors  $\langle w, x_1 - x_{-1} \rangle = 2$ , d'où

$$\gamma = \frac{1}{\|w\|}.$$

## Problème d'optimisation *primal*

Trouver l'hyperplan séparateur de marge maximale revient à trouver le couple  $(w, b)$  tel que

$$\begin{cases} \text{Minimiser} & \|w\|^2 \quad \text{ou} \quad \frac{1}{2}\|w\|^2 \\ \text{sous la contrainte} & y_i(\langle w, x_i \rangle + b) \geq 1 \quad \text{pour tout } i. \end{cases}$$

- Problème d'optimisation convexe sous contraintes linéaires
- Existence d'un **optimum global**, obtenu par résolution du problème "dual" (méthode des multiplicateurs de Lagrange).

# Multiplicateurs de Lagrange : problème primal

- Minimiser pour  $u \in \mathbb{R}^2$   $h(u)$  sous les contraintes  $g_i(u) \geq 0$  pour  $i = 1 \dots n$ ,  $h$  fonction quadratique,  $g_i$  fonction affines.
- Le **Lagrangien** est défini sur  $\mathbb{R}^2 \times \mathbb{R}^n$  par  $L(u, \alpha) = h(u) - \sum_{i=1}^n \alpha_i g_i(u)$ .  
Les variables  $\alpha_i$  sont appelées les **variables duales**. Soit pour tout  $\alpha \in \mathbb{R}^n$ ,
  - $u_\alpha = \underset{u \in \mathbb{R}^2}{\operatorname{argmin}} L(u, \alpha)$ ,
  - $\theta(\alpha) = L(u_\alpha, \alpha) = \min_{u \in \mathbb{R}^2} L(u, \alpha)$  (**fonction duale**).
- **Multiplicateurs de Lagrange : problème dual**
  - Maximiser  $\theta(\alpha)$  sous les contraintes  $\alpha_i \geq 0$  pour  $i = 1 \dots n$ .
  - La solution du problème dual  $\alpha^*$  donne la solution du problème primal :  
 $u^* = u_{\alpha^*}$ .

# Conditions de Karush-Kuhn-Tucker

- $\alpha^* \geq 0$  pour tout  $i = 1 \dots n$ .
- $g_i(u_{\alpha^*}) \geq 0$  pour tout  $i = 1 \dots n$ .
- **Retour sur le problème dual :**

On doit minimiser  $L(u, \alpha) = h(u) - \sum_{i=1}^n \alpha_i g_i(u)$  par rapport à  $u$  et maximiser  $L(u_{\alpha}, \alpha)$  correspondant par rapport aux variables duales  $\alpha_i$ .

$\Rightarrow$  Si  $g_i(u_{\alpha^*}) > 0$ , alors nécessairement  $\alpha_i^* = 0$ .

- **Condition complémentaire de Karush-Kuhn-Tucker** qui s'exprime sous la forme  $\alpha_i^* g_i(u_{\alpha^*}) = 0$ .

## Lagrangien pour SVM en classification binaire

$$\text{Lagrangien : } L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^n \alpha_i (y_i (\langle w, x_i \rangle + b) - 1)$$

Fonction duale :

$$\begin{cases} \frac{\partial L(w, b, \alpha)}{\partial w} = w - \sum_{i=1}^n \alpha_i y_i x_i = 0 \iff w = \sum_{i=1}^n \alpha_i y_i x_i \\ \frac{\partial L(w, b, \alpha)}{\partial b} = - \sum_{i=1}^n \alpha_i y_i = 0 \iff \sum_{i=1}^n \alpha_i y_i = 0 \end{cases}$$

$$\theta(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle.$$

La solution du problème d'optimisation primal est donnée par :

- $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i,$
- $b^* = -\frac{1}{2} \left\{ \min_{y_i=1} \langle w^*, x_i \rangle + \min_{y_i=-1} \langle w^*, x_i \rangle \right\},$  où  $\alpha^* = (\alpha_1^*, \dots, \alpha_n^*)$  est la solution du problème d'optimisation dual :

$$\begin{aligned} \text{Maximiser } \theta(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle. \\ \text{S.C. } \sum_{i=1}^n \alpha_i y_i &= 0 \text{ et } \alpha_i \geq 0 \quad \forall i = 1 \dots n. \end{aligned}$$

La solution  $\alpha^*$  du problème dual est indépendante de la dimension  $p$  : SVM linéaire ne souffre pas du *fléau de la dimension*.

# Conditions de Karush-Kuhn-Tucker

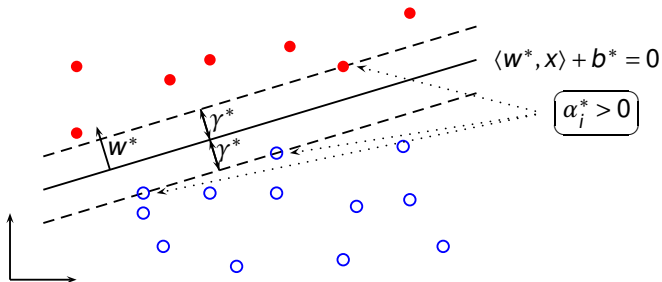
- $\alpha_i^* \geq 0 \quad \forall i = 1 \dots n.$
- $y_i(\langle w^*, x_i \rangle + b^*) \geq 1 \quad \forall i = 1 \dots n.$
- $\alpha_i^* \left( y_i(\langle w^*, x_i \rangle + b^*) - 1 \right) = 0 \quad \forall i = 1 \dots n.$  (condition complémentaire)

1. Le nombre de  $\alpha_i^* > 0$  peut être petit : on dit que la solution du problème dual est **parcimonieuse (sparse)**.
2. Efficacité algorithmique.

Les  $x_i$  tels que  $\alpha_i^* > 0$  sont appelés les **vecteurs supports**. Ils sont situés sur les frontières définissant la marge maximale i.e.

$$y_i(\langle w^*, x_i \rangle + b^*) = 1 \quad (\text{c.f. condition complémentaire de KKT}).$$

## Représentation des vecteurs supports



## En conclusion

La règle de classification obtenue

$$\hat{g}_n(x) = \mathbf{1}_{\langle w^*, x \rangle + b^* \geq 0} - \mathbf{1}_{\langle w^*, x \rangle + b^* < 0},$$

où

- $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i,$
- $b^* = -\frac{1}{2} \left\{ \min_{y_i=1} \langle w^*, x_i \rangle + \min_{y_i=-1} \langle w^*, x_i \rangle \right\},$

ou encore :

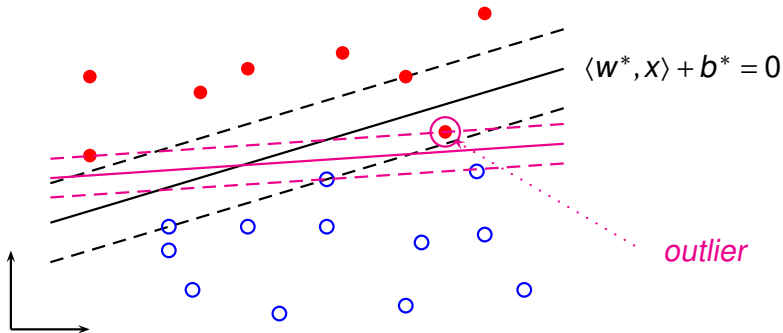
$$\hat{g}_n(x) = \mathbf{1}_{\sum_{x_i \in S} \alpha_i^* y_i \langle x_i, x \rangle + b^* \geq 0} - \mathbf{1}_{\sum_{x_i \in S} \alpha_i^* y_i \langle x_i, x \rangle + b^* < 0}.$$

La marge maximale vaut  $\gamma^* = \frac{1}{\|w^*\|} = \left( \sum_{i=1}^n (\alpha_i^*)^2 \right)^{-\frac{1}{2}}.$



## SVM pour des données non séparables

- La méthode précédente ne peut être appliquée si les données ne sont pas linéairement séparables
- Sensibilité aux *outliers*



## La solution

- Autoriser quelques vecteurs à être bien classés mais dans la région définie par la marge, voire mal classés.
- La contrainte  $y_i(\langle w, x_i \rangle + b) \geq 1$  devient  $y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i$ , avec  $\xi_i \geq 0$ .
  - $\xi_i \in [0, 1] \leftrightarrow$  bien classé, mais région définie par la marge.
  - $\xi_i > 1 \leftrightarrow$  mal classé.
- On parle de **marge souple** ou marge relaxée.
- Les variables  $\xi_i$  sont appelées les **variables ressort** (slacks).
- Un souci : les contraintes relaxées ne peuvent pas être utilisées sans contrepartie sous peine d'obtenir une marge maximale infinie (en prenant des valeurs de  $\xi_i$  suffisamment grandes).

# Pénaliser les grandes valeurs des $\xi_i$

## Problème d'optimisation primal

$$\begin{aligned} &\text{Minimiser en } (w, b, \xi) \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \\ &\text{s.c. } \begin{cases} y_i (\langle w, x_i \rangle + b) \geq 1 - \xi_i \quad \forall i \\ \xi_i \geq 0 \end{cases} \end{aligned}$$

$\hookrightarrow C > 0$  paramètre (**constante de tolérance**) à ajuster.

La solution du problème d'optimisation primal est donnée par :

- $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i$ ,
- $b^*$  tel que  $y_i (\langle w^*, x_i \rangle + b^*) = 1 \quad \forall x_i, 0 < \alpha_i^* < C$ ,

où  $\alpha^* = (\alpha_1^*, \dots, \alpha_n^*)$  est la solution du **problème d'optimisation dual** :

$$\begin{aligned} &\text{Maximiser } \theta(\alpha) = \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle x_i, x_j \rangle \\ &\text{s.c. } \sum_{i=1}^n \alpha_i y_i = 0 \text{ et } 0 \leq \alpha_i \leq C \quad \forall i. \end{aligned}$$

# Conditions de Karush-Kuhn-Tucker

- $0 \leq \alpha_i^* \leq C \quad \forall i = 1 \dots n.$
- $y_i(\langle w^*, x_i \rangle + b^*) \geq 1 - \xi_i^* \quad \forall i = 1 \dots n.$
- $\alpha_i^* (y_i(\langle w^*, x_i \rangle + b^*) + \xi_i^* - 1) = 0 \quad \forall i = 1 \dots n.$
- $\xi_i^* (\alpha_i^* - C) = 0.$

Les  $x_i$  tels que  $\alpha_i^* > 0$  sont les **vecteurs supports**.

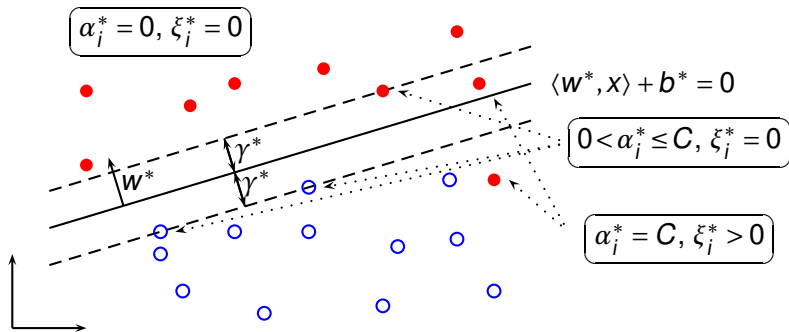
Deux types de vecteurs supports :

- Les vecteurs correspondant à des variables ressort nulles. Ils sont situés sur les frontières de la région définissant la marge.
- Les vecteurs correspondant à des variables ressort non nulles :

$\xi_i^* > 0$  et dans ce cas  $\alpha_i^* = C.$

Les vecteurs qui ne sont pas supports vérifient  $\alpha_i^* = 0$  et  $\xi_i^* = 0$

## Représentation des vecteurs supports



## En conclusion

La règle de classification obtenue

$$\hat{g}_n(x) = \mathbf{1}_{\langle w^*, x \rangle + b^* \geq 0} - \mathbf{1}_{\langle w^*, x \rangle + b^* < 0},$$

où

- $w^* = \sum_{i=1}^n \alpha_i^* y_i x_i,$
- $b^*$  tel que  $y_i (\langle w^*, x_i \rangle + b^*) = 1 \quad \forall x_i, 0 < \alpha_i^* < C,$

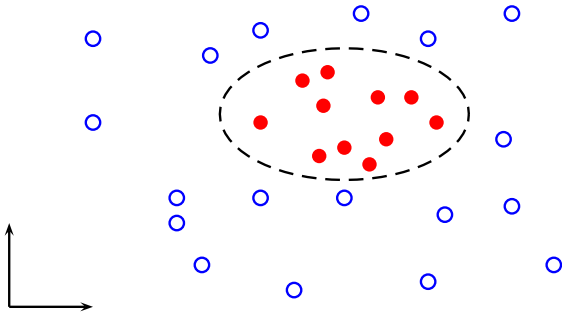
ou encore :

$$\hat{g}_n(x) = \mathbf{1}_{\sum_{x_i \in S} \alpha_i^* y_i \langle x_i, x \rangle + b^* \geq 0} - \mathbf{1}_{\sum_{x_i \in S} \alpha_i^* y_i \langle x_i, x \rangle + b^* < 0}.$$

La marge maximale vaut  $\gamma^* = \frac{1}{\|w^*\|} = \left( \sum_{i=1}^n (\alpha_i^*)^2 \right)^{-\frac{1}{2}}.$

## SVM non linéaire : astuce du noyau

Exemple de données difficiles à discriminer linéairement :



Une règle SVM linéaire donnera une très mauvaise classification avec un nombre de vecteurs supports très élevé  $\Rightarrow$  SVM non linéaire ?

## L'idée (Boser, Guyon, Vapnik, 1992)

Envoyer les entrées  $\{x_i, i = 1 \dots n\}$  dans un espace de Hilbert  $\mathcal{H}$  (muni d'un produit scalaire  $\langle \cdot, \cdot \rangle_{\mathcal{H}}$ ), de grande dimension, voire de dimension infinie, via une fonction  $\varphi$ , et appliquer une SVM linéaire aux nouvelles données  $\{(\varphi(x_i), y_i), i = 1 \dots n\}$ .  
La sortie attribuée à l'entrée  $x$  est celle attribuée à son image  $\varphi(x)$ .

- La fonction  $\varphi$  est appelée la **fonction de représentation (feature function)**.
- L'espace  $\mathcal{H}$  est appelé l'**espace de représentation (feature space)**.

Dans l'exemple précédent, pour  $\varphi(x) = (x_1^2, x_2^2, x_1, x_2)$ , les données deviennent linéairement séparables dans  $\mathbb{R}^4$ .



## Comment choisir $\mathcal{H}$ et $\varphi$ ?

La règle de discrimination de la SVM non linéaire est définie par :

$$\hat{g}_n(x) = \mathbf{1}_{\sum y_i \alpha_i^* \langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{H}} + b^* \geq 0} - \mathbf{1}_{\sum y_i \alpha_i^* \langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{H}} + b^* < 0},$$

où

$$\begin{aligned} \text{Maximiser } \theta(\alpha) &= \sum_{i=1}^n \alpha_i - \frac{1}{2} \sum_{i=1}^n \sum_{j=1}^n \alpha_i \alpha_j y_i y_j \langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}} \\ \text{s.c. } \sum_{i=1}^n \alpha_i y_i &= 0 \text{ et } 0 \leq \alpha_i \leq C \quad \forall i. \end{aligned}$$

- Remarque fondamentale :

La règle de discrimination de la SVM non linéaire ne dépend de  $\varphi$  qu'à travers des produits scalaires de la forme  $\langle \varphi(x_i), \varphi(x) \rangle_{\mathcal{H}}$  ou  $\langle \varphi(x_i), \varphi(x_j) \rangle_{\mathcal{H}}$ .

- **Astuce du noyau (kernel trick)** : La connaissance de la seule fonction  $k$  définie par  $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$  permet de lancer la SVM dans  $\mathcal{H}$ , sans déterminer explicitement  $\mathcal{H}$  et  $\varphi$ .

# Le noyau

- Une fonction  $k : \mathcal{X}, \mathcal{X} \rightarrow \mathbb{R}$  telle que  $k(x, x') = \langle \varphi(x), \varphi(x') \rangle_{\mathcal{H}}$  pour une fonction  $\varphi : \mathcal{X} \rightarrow \mathcal{X}$  donnée est appelée un **noyau**.
- Un noyau est souvent plus facile à calculer que la fonction  $\varphi$ . Par exemple, si pour  $x = (x_1, x_2) \in \mathbb{R}^2$ ,  $\varphi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2)$ , alors que  $k(x, x') = \langle x, x' \rangle^2$ .
- Quelques noyaux classiques pour  $\mathcal{X} = \mathbb{R}^p$ 
  - Noyau **polynomial** :  $k(x, x') = (\langle x, x' \rangle + c)^p$   
 $\hookrightarrow \varphi(x) = (\varphi_1(x), \dots, \varphi_k(x))$  avec  $\varphi_i(x)$  = monôme de degré inférieur à  $p$  de certains composantes de  $x$ .
  - Noyau **gaussien** ou **radial** (RBF) :  $k(x, x') = e^{-\frac{\|x-x'\|^2}{2\sigma^2}}$   
 $\hookrightarrow$  à valeurs dans un espace de dimension infinie.
  - Noyau **laplacien** :  $k(x, x') = e^{-\frac{\|x-x'\|}{\sigma}}$ .

# Agrégation de noyaux

- Soit  $k_1$  et  $k_2$  des noyaux,  $f$  une fonction  $\mathbb{R}^p \rightarrow \mathbb{R}$ ,  $\psi : \mathbb{R}^p \rightarrow \mathbb{R}^{p'}$ ,  $B$  une matrice définie positive,  $P$  un polynôme à coefficients positifs,  $\lambda \geq 0$ .
- La fonction définie  $k(x, x') = k_1(x, x') + k_2(x, x')$ ,  $\lambda k_1(x, x')$ ,  $k_1(x, x')k_2(x, x')$ ,  $f(x)f(x')$ ,  $k(\psi(x), \psi(x'))$ ,  $x^T B x'$ ,  $P(k_1(x, x'))$  ou  $e^{k_1(x, x')}$  est encore un noyau.
- **Noyaux pour  $\mathcal{X} \neq \mathbb{R}^p$**  : quelques noyaux ont été proposés pour d'autres types d'objets comme des
  - ensembles,
  - arbres,
  - graphes,
  - chaînes de symboles,
  - documents textuels...

# Éléments de théorie

$$\text{Minimiser } \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \xi_i \quad \text{s.c.} \quad \begin{cases} y_i(\langle w, x_i \rangle + b) \geq 1 - \xi_i & \forall i \\ \xi_i \geq 0 \end{cases}$$

est équivalent à minimiser

$$\frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \left(1 - y_i(\langle w, x_i \rangle + b)\right)_+,$$

ou encore

$$\frac{1}{n} \sum_{i=1}^n \left(1 - y_i(\langle w, x_i \rangle + b)\right)_+ + \frac{1}{2Cn} \|w\|^2.$$

$\gamma(w, b, x_i, y_i) = \left(1 - y_i(\langle w, x_i \rangle + b)\right)_+$  est une majorant convexe de l'erreur empirique  $\mathbf{1}_{y_i(\langle w, x_i \rangle + b) \leq 0}$

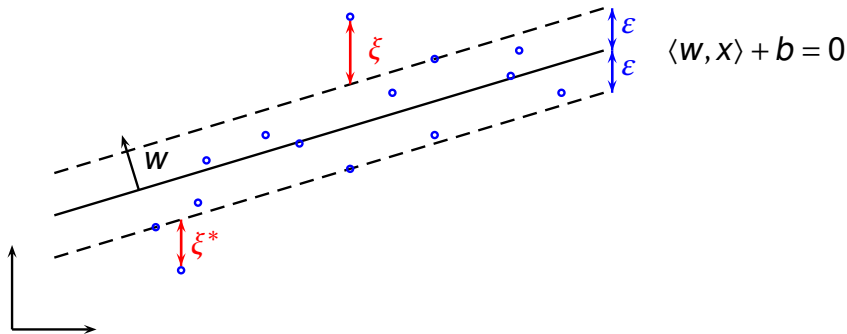
↪ SVM = Minimisation de risque empirique convexifié régularisé

## Conclusion et questions ouvertes

- Le rééquilibrage des données
- La renormalisation des données
- Les réglages à effectuer :
  - Le noyau et ses paramètres (validation croisée, bootstrap ?)
  - La constante de tolérance  $C$  (validation croisée, bootstrap ?)
  - L'algorithme d'optimisation (SMO ou variantes par exemple)
- La sélection de variables
- Généralisation à la discrimination multiclassées : one-versus-all, one-versus-one

# SVM pour la régression (SVR) : un mot

La dite  $\varepsilon$ —SVR linéaire en dimension 2



# Le principe

Trouver la règle de régression linéaire la plus *plate* possible, sous certaines contraintes.

**Problème d'optimisation prima pour la  $\varepsilon$ -SVR linéaire**

$$\text{Minimiser en } (w, b, \xi, \xi^*) \quad \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n (\xi_i + \xi_i^*)$$

$$\text{s.c.} \quad \begin{cases} y_i - \langle w, x_i \rangle - b \leq \varepsilon + \xi_i & \forall i \\ \langle w, x_i \rangle + b - y_i \leq \varepsilon + \xi_i^* & i \\ \xi_i, \xi_i^* \geq 0 & \forall i \end{cases}$$

$\hookrightarrow C > 0$  paramètre à ajuster.

$\hookrightarrow$  SVR non linéaire : astuce du noyau !

## **Application : le package e1071**

- La fonction svm du package e1071
- La fonction ksvm du package kernlab
- Le package caret pour le choix des paramètres



# Table of Contents

1. Introduction
2. Régression vs classification supervisée
3. Arbres de décision uniques
4. Agrégation par la moyenne : bagging
5. Agrégation séquentielle : boosting
6. Justification du boosting : minimisation de risque empirique
7. Méthodes à noyaux : SVM et SVR